

Projet de deuxième année MT

Conception d'un module zigbee (2022)



Encadrant partie analogique : Sylvain Bourdel

Ecrandrant partie système : Yannis Le Guennec

Encadrants partie numérique : Laurent Fesquet

Michele Portolan

Encadrant partie management : Tarik Larja

Introduction	2
Etude système	3
LNA	18
LNA Single (Romain HUC)	19
LNA Diff (Manon TABARDEL)	23
PA	28
PA single (Manon CREVEL)	30
PA Différentiel (Nathan LOISY)	33
Mixer	35
Mixer Passif (Yann ZYZELEWICZ)	36
Mixer Actif (Pierre-Olivier Guessard)	41
PLL (Guillaume Monti - Pierre-Louis Hellier - Pierre Bordère)	49
Récepteur RF (Yann ZYZELEWICZ et Manon TABARDEL)	64
Emetteur RF (Pierre-Olivier Guessard & Nathan Loisy)	69
ADC (Antoine Leone et Brian Martinez)	75
WP3 - Section numérique :	91
Interfaces externes Rx/Tx (Axel Baldacchino, Tom Désesquelle)	92
Démodulateur IQ et filtres à réjection d'image (Mamadou Hawa DIALLO, Matisse REBOUD)	113
CORDIC (Paul-Arthur MEHL, Megi MYFTARAJ)	125
CDR (Alexandre Scouarnec, Elisabeth Porret)	133
Wrappers (Axel Baldacchino, Mamadou Hawa Diallo)	139
TOP (Tom Désesquelle)	144
PLACE & ROUTE TOP (Anthony Teissier, Axel Baldacchino)	150

Introduction

L'objectif de ce projet est de réaliser un module de communication Zigbee. Le Zigbee est un protocole de télécommunication courte-distance très utilisé en domotique ou bien encore dans le domaine des objets connectés. Le but de ce projet sera donc de respecter les spécifications suivantes :

- Puissance d'émission maximale : 14 dBm
- Sensibilité en réception maximale : -92 dBm
- Portée : 44m

Comme il s'agit d'un module de télécommunication, il y aura par définition une partie émission et une partie réception qui peuvent se représenter de la manière suivante :

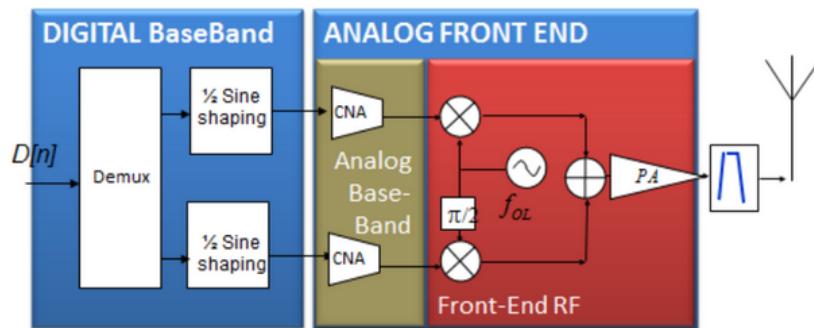


Figure 1 : Architecture en émission

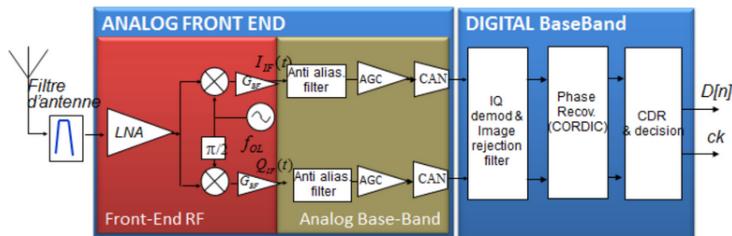


Figure 2 : Architecture en réception

Enfin, pour mener ce projet à bien, la classe sera divisée en 2 avec une partie analogique et une partie numérique dont l'organisation est représentée ci-dessous :

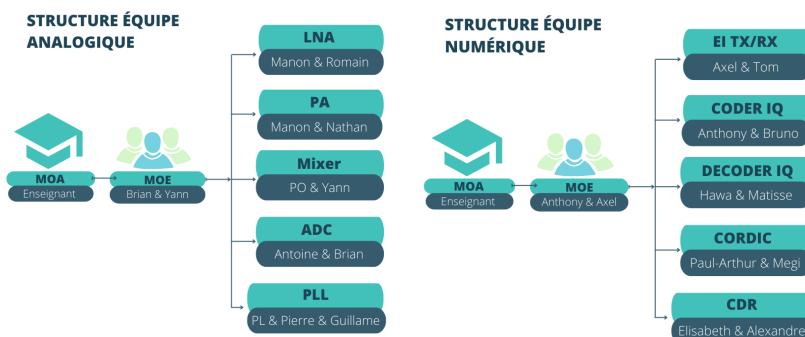


Figure 3 : Organisation du projet

Chacun travaille en binôme (sauf le groupe de la PLL en trinôme car nous sommes une classe de 21) et a un bloc à réaliser comme vous pouvez le voir sur la figure ci-dessus.

ETUDE SYSTEME

FONCTIONS NUMÉRIQUES

L'intérêt de l'étalement de spectre DSSS (Direct Sequence Spread Spectre) est de toujours avoir une puissance non nul en réception, notamment s'il y a un pic de puissance. Il permet également de maximiser le rapport signal sur bruit.

L'expression de la modulation MSK (Minimum Shift Keying) s'exprime par la relation suivante :

$$S_{MSK}(t) = A \cos(2\pi f_0 t + \varphi(t))$$

Or nous voulons déterminer la fréquence instantanée du signal $S_{MSK}(t)$ qui n'est autre que la dérivée de la phase par rapport au temps divisée par 2π .

La fréquence instantanée vaut alors :

$$f = (2\pi f_0 t + \varphi(t))' / 2\pi$$

avec

$$\varphi(t) = \frac{\pi}{4} + \frac{\pi}{2} \left(b(i) \frac{t-iT_b}{T_b} + \sum_{n=0}^{i-1} b(n) \right)$$

Finalement nous avons :

$$f = 2\pi f_0 + \frac{\pi}{2} b(i) \frac{1}{T_b} / 2\pi$$

$$f = f_0 + \frac{b(i)}{4Tb}$$

$b(i)$ vaut +1 ou -1, selon sa valeur nous obtenons au final :

$$f_+ = f_0 + \frac{b(i)}{4Tb} \quad \text{et} \quad f_- = f_0 - \frac{b(i)}{4Tb}$$

La figure ci-dessous correspond à la séquence binaire envoyée en entrée du générateur MSK. Il s'agira dans notre projet des bits de données à transmettre qui proviendront de la FIFO.

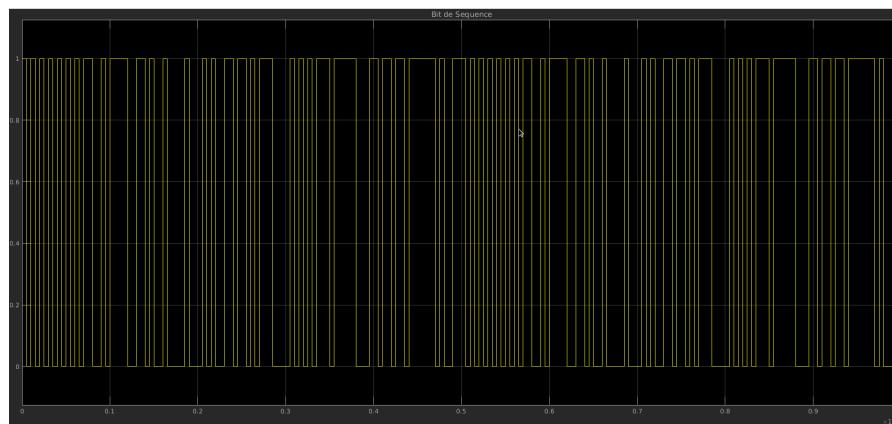


Figure 1 : Séquence binaire envoyée en entrée du générateur MSK

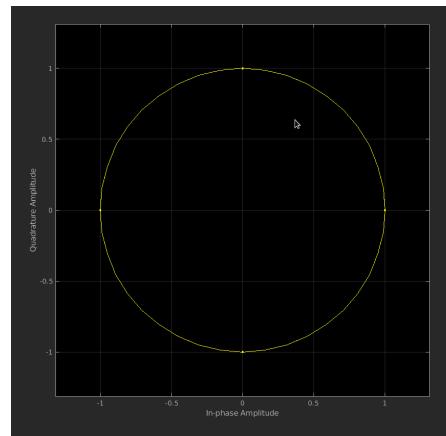


Figure 2 : Diagramme de constellation

Après observation du diagramme de constellation plus haut en figure 2, nous constatons que l'amplitude du signal est constante pour tout temps et la phase est continue. Nous avons donc une enveloppe constante.

Nous allons parcourir le cercle de phase en suivant les voies I et Q

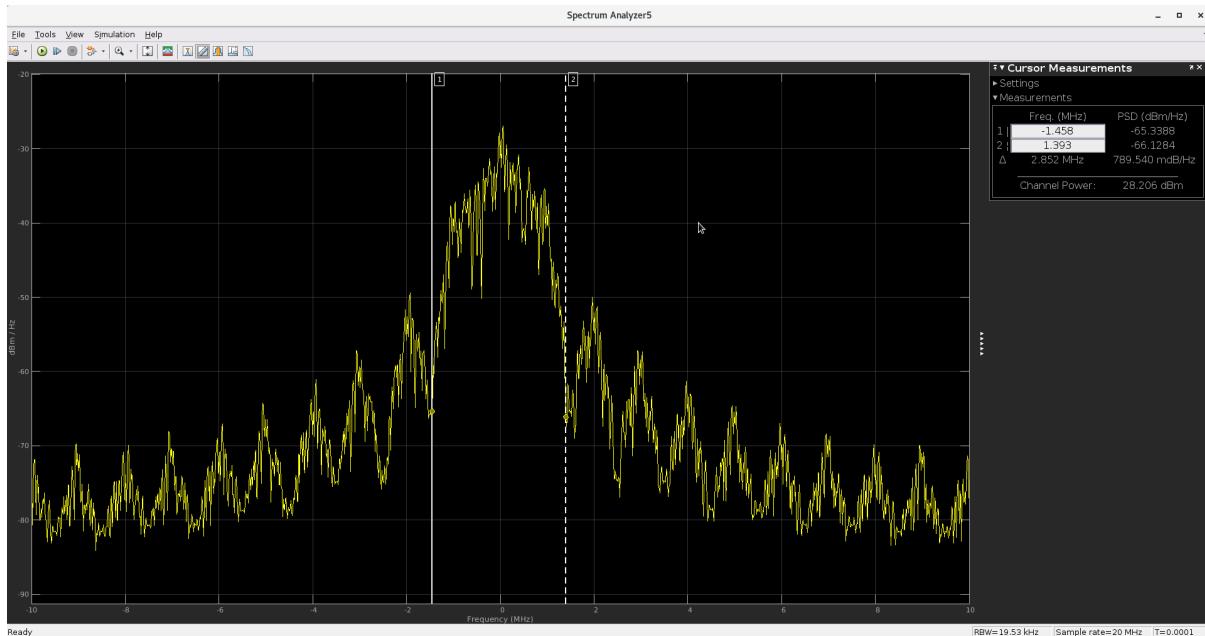


Figure 3 : Spectre de la modulation MSK

En observant le spectre ci-dessus, nous pouvons en déduire que la bande passante est de 2.85 MHz trouvée

La figure ci-dessous illustre les signaux en bande de base sur les voies I et Q.

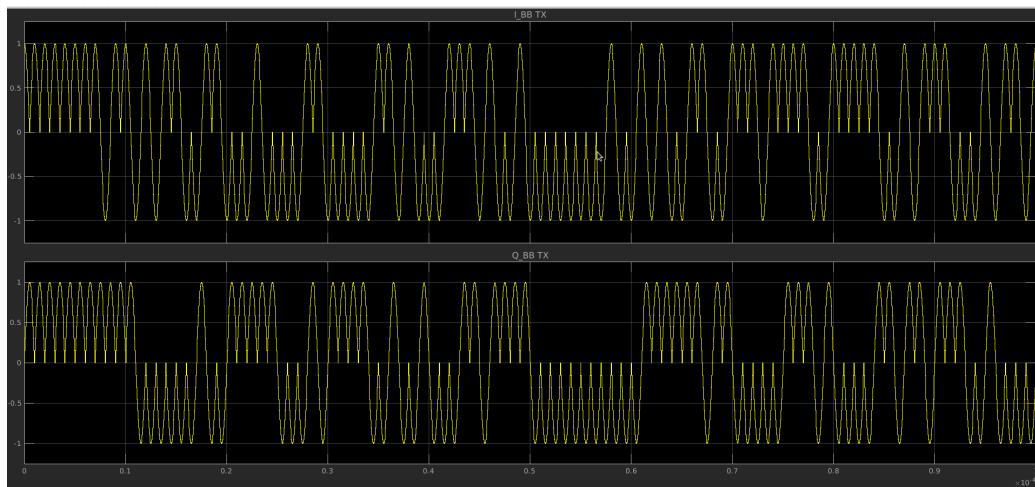


Figure 4 : Signaux S_I et S_Q en bande de base de la modulation IQ

Nous pouvons voir que la voie I du haut est bien déphasée de $\frac{\pi}{2}$ par rapport à la voie Q en dessous.

Nous n'allons pas utiliser les récepteurs de types homodyne car photo + fuite de l'oscillateur local

Nous allons utiliser les récepteurs de type LOW IF
 la fréquence de l'oscillateur locale est $f = f_{rf} - f_{if}$

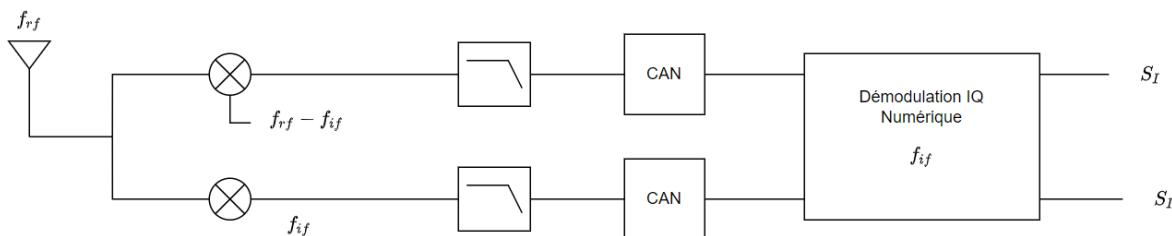


Figure 5 : Représentation de la chaîne de réception d'un module MSK

La bande passante est centrée autour de la fréquence f_{if} et de largeur 3MHz. Tandis que la génération DC est à 0 et ne va pas perturber la bande. Nous pouvons simplifier cela avec le graphique ci-dessous :

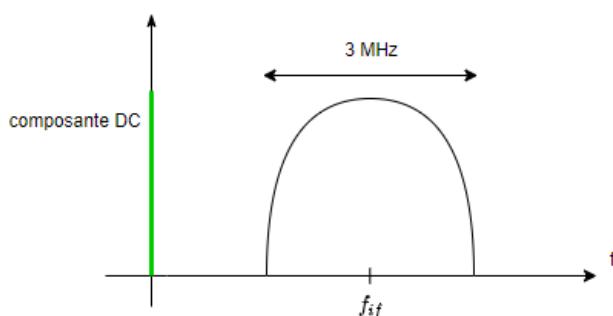


Figure 6 : Bande passante d'un canal Zigbee

Le spectre est centré autour de la fréquence Fif par un convertisseur de fréquence.

Afin de revenir en bande de base, nous allons utiliser un démodulateur IQ numérique à la fréquence Fif pour récupérer des voies I et Q.

La fréquence Fif est faible (2,5MHz) donc nous pouvons en conclure que la fréquence d'échantillonnage est atteignable (quelques dizaines de méga). La démodulation numérique se fait à la fréquence fif pour récupérer Si et Sq.

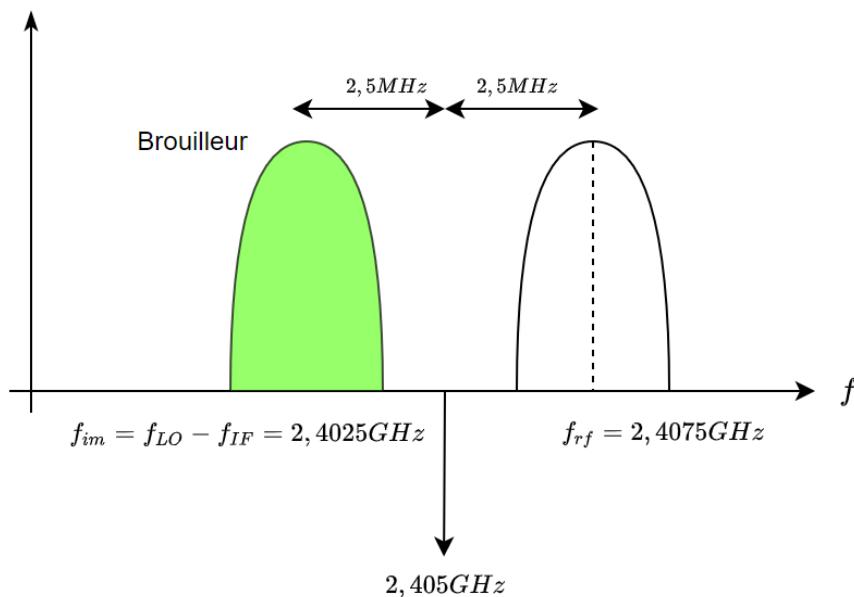


Figure 7 : Représentation de la fréquence image du canal

Le récepteur Low IF si $f_{im} = (f_{lo} - f_{if})$ est un brouilleur autour de 2.4025GHz alors on a aussi 2.5 MHz en sortie du Mixeur (au signe près) ce qui engendre du repliement sur la freq. Nous appelons cela "fréquence image", c'est-à-dire que la fréquence Fim est aussi convertie à Fif, nous pourrions filtrer mais cela nécessiterait un filtre d'ordre très élevé ce qui serait compliqué à réaliser.

La démodulation numérique permet d'éliminer la fréquence image, cela n'est pas particulièrement compliqué à réaliser relativement au domaine RF.

Rejeter l'impact de la fréquence image du récepteur c'est s'occuper du canal ZigBee adjacent. Chaque canal est séparé de 5 MHz. C'est pour cela que nous avons choisi Fif fixe et égal à la moitié de 5 MHz soit 2,5 MHz.

Comment adresser les différents canaux?

- à la réception le convertisseur IQ possède un oscillateur avec $Frf - Flo = Fif$

Nous avons intérêt à figer la Fif car elle va conditionner énormément de blocs numériques

Nous avons $Frf = 2,4075 \text{ GHz}$ et $Flo = 2,405 \text{ GHz}$

Donc Frf-Fol= 2,5 MHz

Nous nous attendons à voir sur le spectre de la voie I, centrée à la Fif, une bande passante égale à 3 MHz.

Pour le CAN on visera un $F_s = 10$ MHz pour bien respecter la fréquence d'échantillonnage et le théorème de Shannon ($F_s > 2.f_{\max}$)

Observation des spectres des signaux I_{IF} et Q_{IF} après conversion AN.

Deux signaux type MSK centrées à 2,5 MHz.

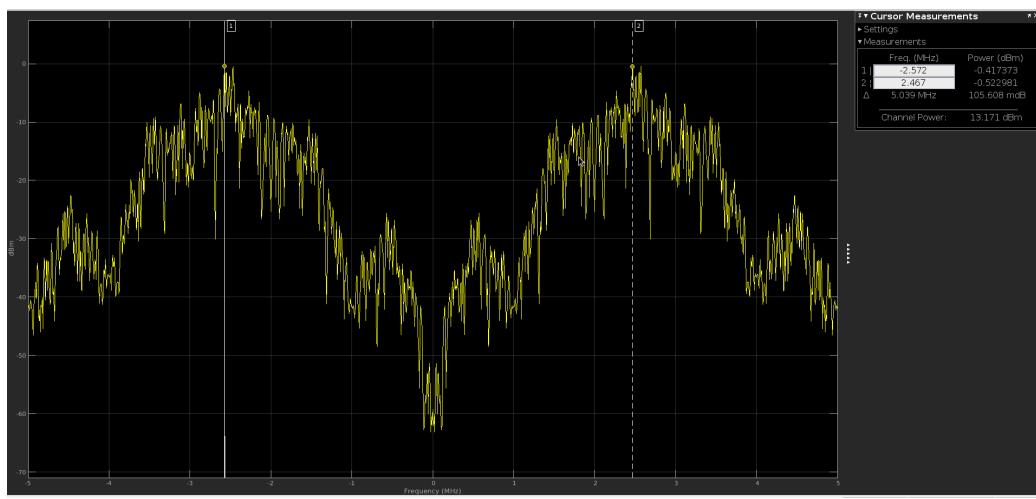


Figure 8 : graphique de I_{IF}

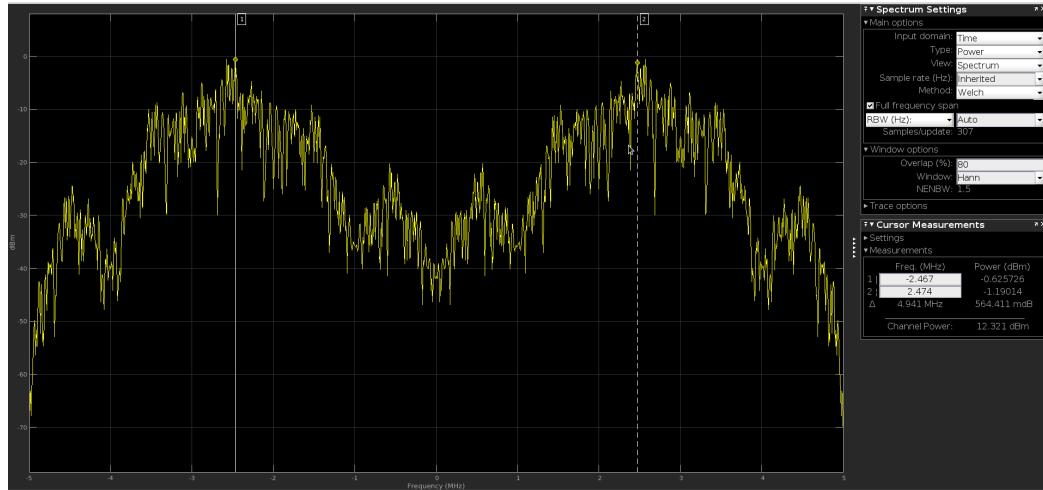


Figure 9 : graphique de Q_{IF}

Nous réalisons le démodulateur IQ numérique à réjection d'image.

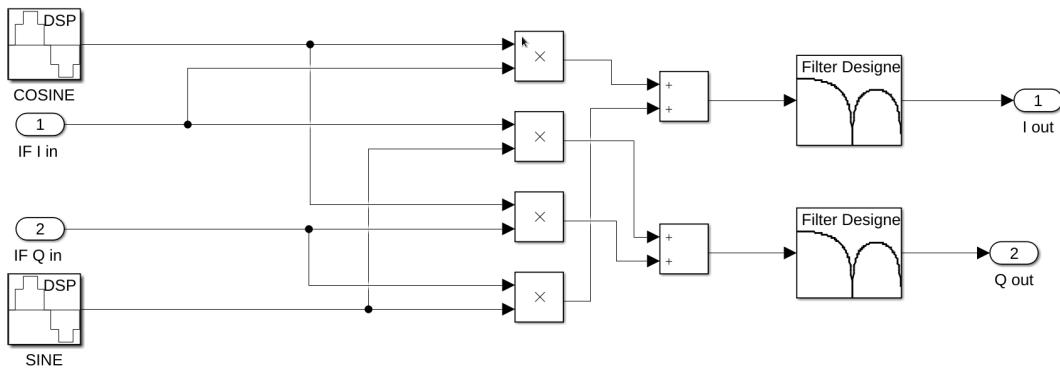


Figure 10 : Démodulateur IQ numérique à réjection d'image

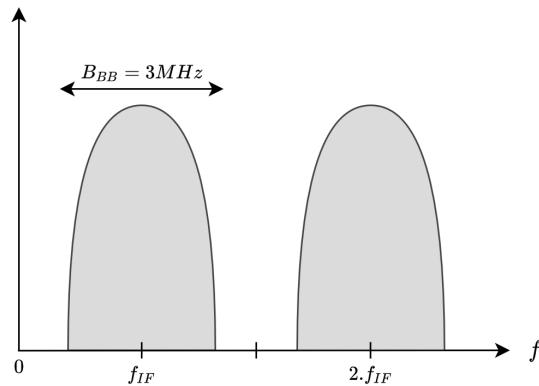


Figure 11 : Effet du filtre sur la périodisation du signal MSK

Avant de filtrer, nous devons ramener la bande de fréquence autour de 0 de façon à extraire nos signaux Ibb et Qbb. On ne filtre pas directement autour de Fif car passe bande trop difficile à réaliser.

Nous ramenons donc Fif à 0, la bande ira alors de -1.5 MHz à +1.5 MHz.

Le filtre doit nous permettre de supprimer le lobe centré en 2Fif généré par les mixeurs du démodulateur IQ. Tout ce qui concerne -Fif est rejeté à 2Fif.

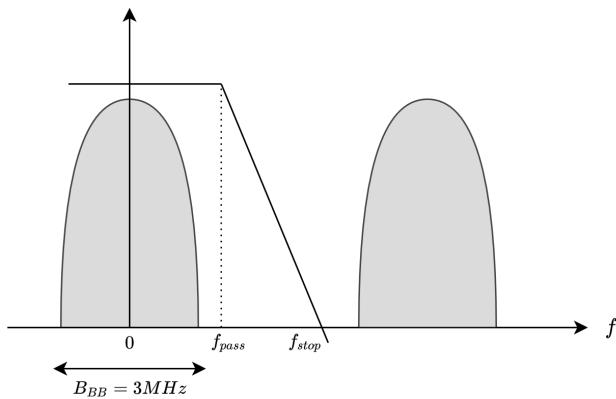


Figure 12 : Filtrage de la réjection d'image

Nous filtrons donc à Fpass = 1.5MHz et Fstop = 4.5 MHz

- 1) Comment sont mis en forme les niveaux sur I et Q. Si le démodulateur est capable de retrouver parfaitement le Ibb et le Qbb du transmetteur on est capable de retrouver la phase du transmetteur, on est capable de démontrer le rôle du démodulateur → comparer Ibb Tx avec Ibb Rx
- 2) Réjection de la fréquence image

L'architecture proposée va discriminer les deux signaux suivant s'il est supérieur à f_{ol} ou inférieur à f_{ol} . Cela va permettre de garder le lobe centré en f_{rf} et supprimer celui centré en f_{im} .

$$f_{rf} = f_{ol} + f_{if}$$

$$f_{im} = f_{ol} - f_{if} \text{ avec } f_{if} = 2,5 \text{ MHz}$$

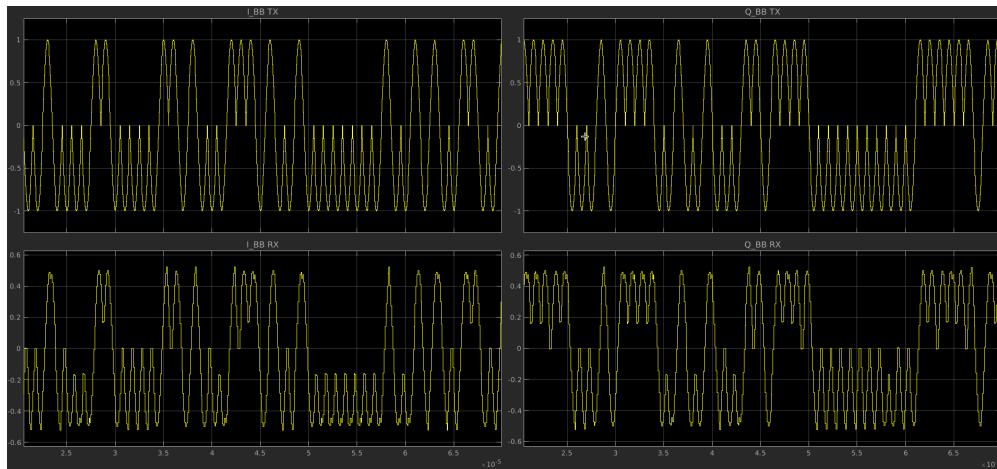


Figure 13 : Signal de sortie en bande de base sans le filtre

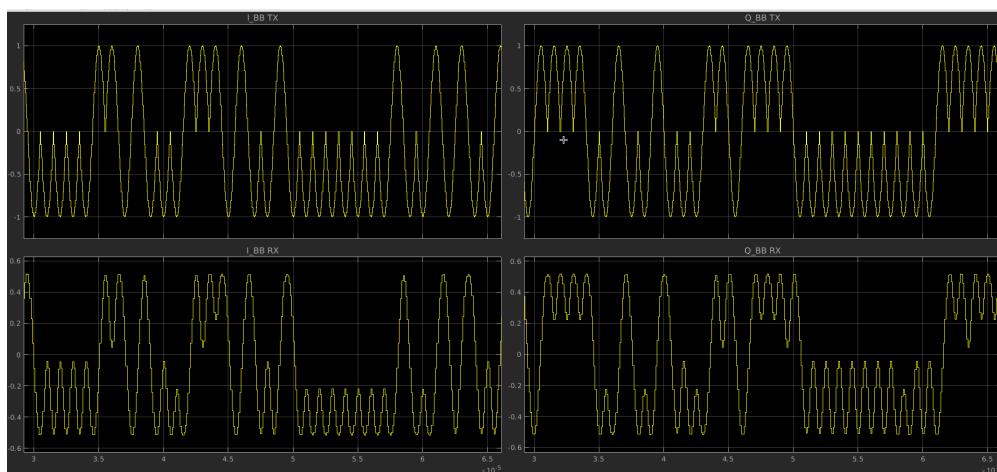


Figure 14 : Signal de sortie en bande de base avec le filtre

Nous pouvons remarquer que le filtre n'a aucun impact sur le signal de sortie. S'il a une utilité, ça ne peut être le cas que sur la fréquence image.

Si nous retirons le filtre passe-bas cela ne va rien changer.

C'est le filtre passe bas qui va nous permettre de rejeter la fréquence image.

Pour tester cela, on va présenter à l'entrée du transmetteur un signal à la fréquence image $f_{IM} = f_{RF} - f_{OL} = 2,025\text{GHz}$. Attention à ne pas envoyer les mêmes données, pour cela on change le seed du générateur de Bernoulli. Puis, nous sommes les deux signaux RF TX pour observer à la réception.

Nous pouvons alors observer les blocs du module zigbee sur le logiciel Simulink de Matlab.

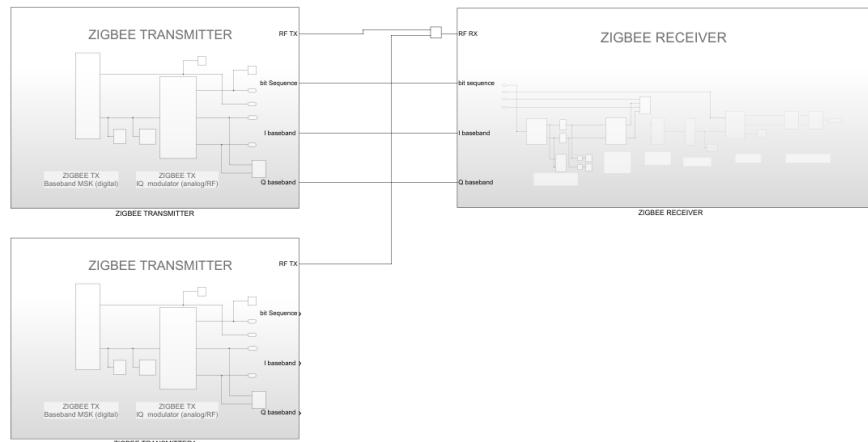


Figure 15 : Représentation Matlab des blocs de réception et d'émission du module Zigbee

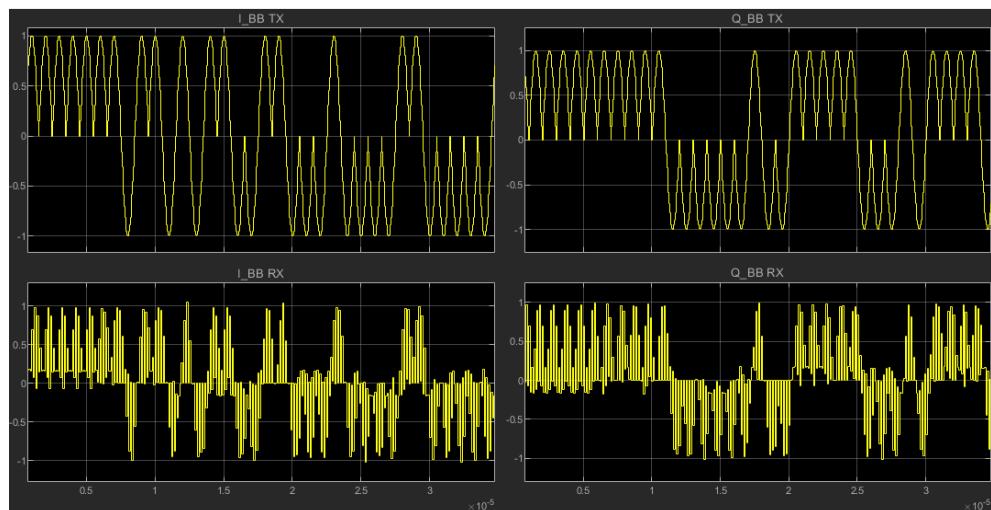


Figure 15 : Comparaison entre le signal I et Q en réception et en émission
 à la fréquence Fif sans filtre numérique

Nous observons du bruit. Il sera plus difficile de récupérer la fréquence de la porteuse.

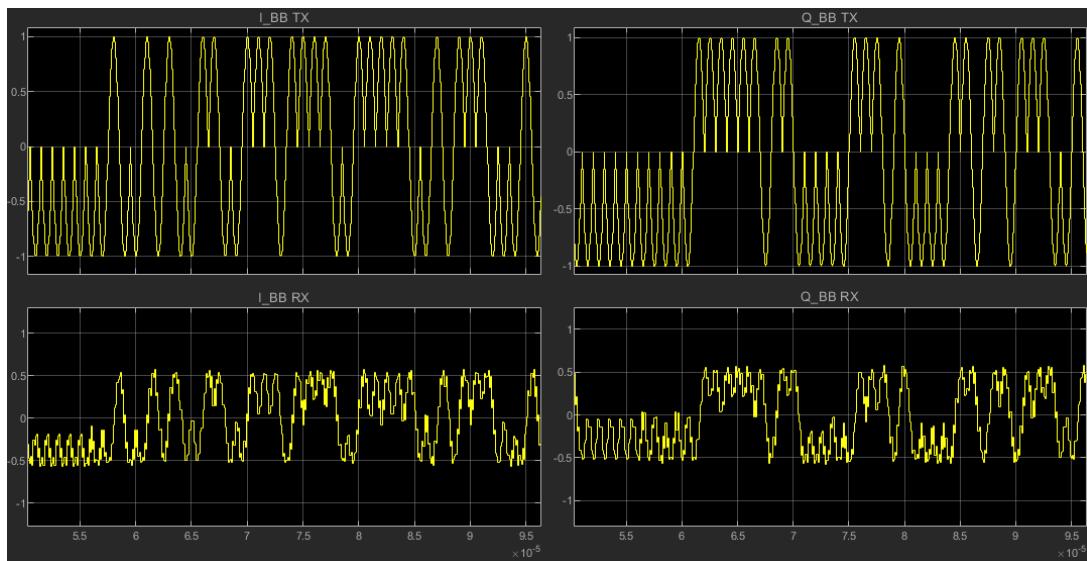


Figure 16 : Comparaison entre les signaux I et Q en réception et en émission
 à la fréquence Fif avec le filtre numérique

CORDIC

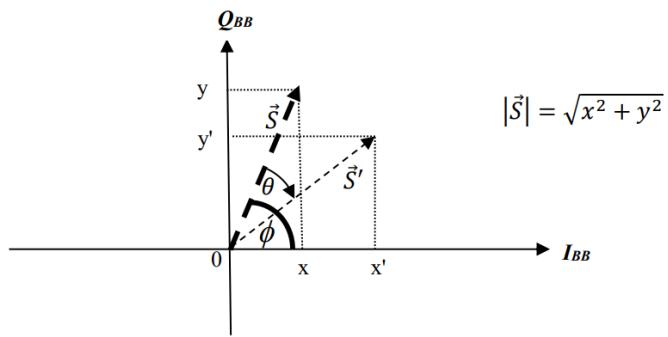


Figure 17 : $Q_{BB} = F[I_{BB}]$

Le CORDIC vient juste après le bloc de démodulation numérique. Il nous permet de connaître la variation de phase $\phi(t)$.

En connaissant $\phi(t)$ en fonction du temps, nous nous attendons à une croissance linéaire de 0 à π entre 0 et 2Tb puis décroissance de π à 0 de 2Tb à 4Tb

Lorsque nous dérivons la phase on peut retrouver la valeur des bits. En effet lorsque la pente est positive = bit à 1, quand la pente est négative = bit à 0.

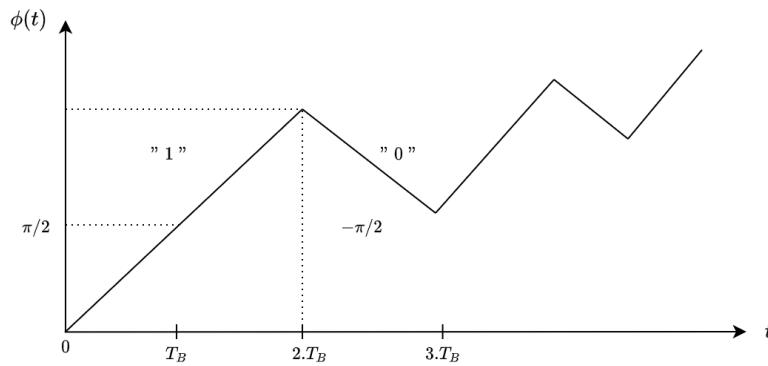


Figure 18 : Estimation de la phase ϕ en fonction du temps

Estimer ϕ avec un algorithme simple, ergonomique et économique en termes de temps de calcul.

Il suffit d'utiliser $\arctan(QBB/IBB) = \phi \rightarrow$ problème faire l'arctan en num simplement \rightarrow impossible, mais TOUT est possible sur MatLab.

Une autre manière de le faire serait d'estimer la phase quelle que soit t en fonction de IBB et QBB par rotations successives par des angles connus (par ex $-\pi/2 + \pi/4 + \dots$).

Quand le vecteur est ramené sur l'axe des abscisses on a estimé ϕ ("dichotomie angulaire")

si Y positif on ajoute -angle, si Y est négatif on ajoute +angle

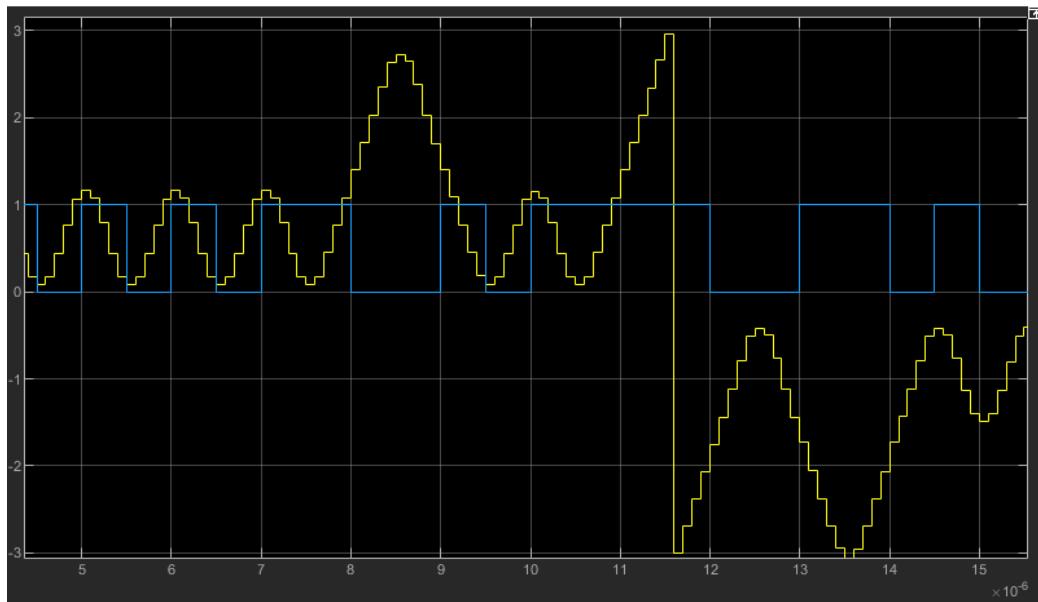


Figure 19 : Comparaison entre les bits d'entrée et la phase

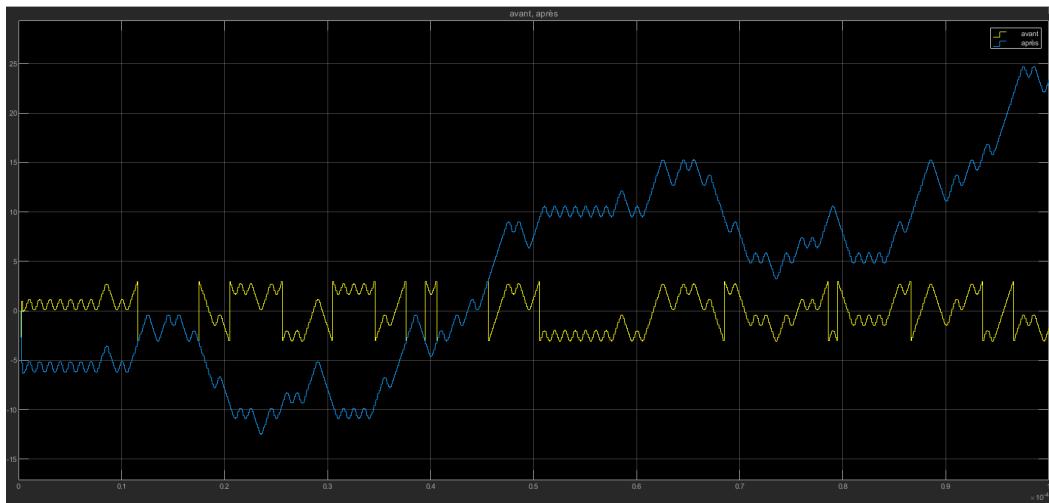


Figure 20 : Comparaison entre la phase sans bloc UNWRAP et avec le bloc UNWRAP

Pour finir, le facteur de décimation indiqué dans le bloc décision est de 5, nous éliminons donc 4 échantillons à chaque 5-ième échantillon. Cela réduit donc la bande utile par 5.

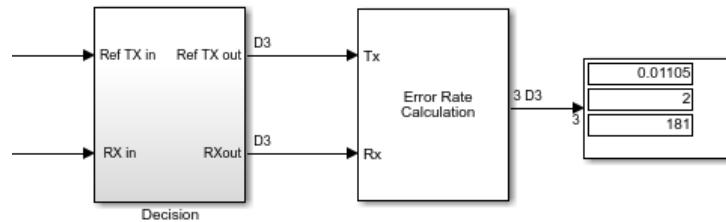


Figure 21 : Calcul du taux d'erreur

FRONT END RF

I) Chaîne d'émission

La chaîne d'émission est composée de deux mixeurs (voies I et Q), d'un oscillateur local, d'un additionneur et d'un amplificateur de puissance.

1) Spécification de l'amplificateur de puissance

Pour obtenir un rendement efficace, il faut travailler proche du point de compression de l'amplificateur. Au-delà du point de compression, le gain de l'amplificateur n'est plus linéaire.

Cependant nous utilisons une modulation à enveloppe constante. Cela signifie que la puissance instantanée du signal modulé est constante. Cela nous permet de travailler dans la zone non-linéaire de l'amplificateur sans dégrader le signal.

Nous cherchons donc à travailler au point de compression de l'amplificateur.

L'objectif est d'obtenir une puissance d'émission de 14 dBm. On estime la puissance du signal en sortie du modulateur à -1 dBm.

On veut : $P_{tx} = OCP_{pa} = 14 \text{ dBm} = P_{in} + G_{pa} - 1 \text{ dB}$

On déduit : $G_{pa} = 14 \text{ dBm} + 1 \text{ dB} - P_{in} = \mathbf{16 \text{ dB}}$.

Le gain de l'amplificateur doit donc être de 16 dB pour obtenir une puissance d'émission de 14 dBm.

II) Chaîne de réception

La chaîne de réception est composée d'un amplificateur faible bruit différentiel, de deux mixeurs et d'un oscillateur local.

1) Calcul du facteur de bruit de la chaîne de réception

On propose les spécifications suivantes :

- LNA : $G_{lna} = 20 \text{ dB}$ et $NF_{lna} = 3,5 \text{ dB}$.
- Mixeur : $G_{mix} = -4 \text{ dB}$ et $NF_{mix} = 10 \text{ dB}$.

On considère le bruit apporté par l'oscillateur local comme négligeable. On suppose les étages adaptés.

En appliquant la formule de Friss (avec les valeurs linéaires), on déduit le facteur de bruit de la chaîne globale : $\mathbf{NF_{rx} = 3,6 \text{ dB}}$.

On observe que le bruit de la chaîne globale dépend principalement du facteur de bruit du premier étage. Le fort gain du LNA permet de minimiser l'impact des étages suivants.

2) Calcul de la sensibilité du récepteur

Le rapport signal sur bruit minimum (SNR_{min}) pour garantir un taux d'erreur minimum de 10^{-6} en démodulation est : $\mathbf{SNR_{out,in} = 14 \text{ dB}}$.

Le facteur de bruit de la chaîne de réception avec les spécifications proposées est $\mathbf{NF_{rx} = 3,6 \text{ dB}}$.

On suppose que la puissance de bruit en entrée ($P_{noise,in}$) de la chaîne de réception est égale au bruit thermique à 20°C (température ambiante supposée) dans une bande de 2,8 MHz (largeur de canal) : $\mathbf{P_{noise,in} = -109 \text{ dBm}}$.

La sensibilité ($P_{in,min}$) est la puissance minimale en entrée de la chaîne permettant d'obtenir $SNR_{out,in}$. On déduit à partir de la formule du facteur de bruit (en décibels) :

$$P_{in,min} (\text{dBm}) = NF_{rx} (\text{dB}) + P_{noise,in} (\text{dBm}) + SNR_{out,in} (\text{dB}).$$

$$P_{in_min} = -92 \text{ dBm.}$$

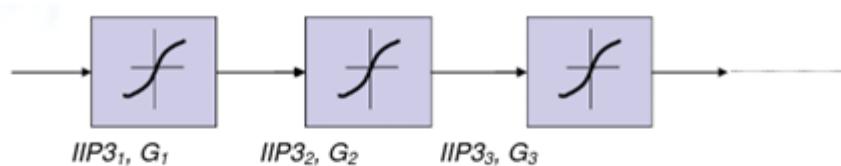
Conclusion sur le dimensionnement de la chaîne Rx : En appliquant la formule de Friss, on cherche un compromis sur les performances des sous-fonctions de la chaîne afin d'obtenir la sensibilité souhaitée. Les performances du premier étage sont déterminantes, ce qui explique l'utilisation d'un amplificateur faible bruit.

3) Calcul de l'IIP3 du récepteur

On propose :

- $IIP3_{lna} = 13 \text{ dBm}$ soit $OIP3_{lna} = IIP3_{lna} + G_{lna} = 33 \text{ dBm}$.
- $IIP3_{mix} = 26 \text{ dBm}$ soit $OIP3_{mix} = IIP3_{mix} + G_{mix} = 22 \text{ dBm}$.

On calcule l' $IIP3_{rx}$ de la chaîne de réception à partir de la formule suivante :



$$\frac{1}{IIP3(mW)} = \frac{1}{IIP3_1(mW)} + \frac{G_1}{IIP3_2(mW)} + \frac{G_1 G_2}{IIP3_3(mW)} + \dots \quad (5)$$

On trouve : **IIP3_{rx} = 5 dBm.**

On déduit $OIP3_{rx} (\text{dBm}) = IIP3_{rx} (\text{dBm}) + G_{rx} (\text{dB}) = 21 \text{ dBm}$.

4) Calcul du SFDR

L'apparition des IM3 à pour effet de désensibiliser la chaîne de réception.

D'après le cours, le SFDR est défini comme le rapport entre la puissance maximale (en sortie de chaîne pour laquelle les IM3 sortent du bruit) et la puissance de bruit en sortie de la chaîne de réception : $SFDR (\text{dB}) = P_{out_max} (\text{dBm}) - P_{noise_out} (\text{dBm})$.

$$\text{On a : } P_{out_max} (\text{dBm}) = P_{in_max} (\text{dBm}) + G_{rx} (\text{dB}) = -17 \text{ dBm} \quad (\text{avec } P_{st_{max}} (\text{dBm}) = \frac{2P_{IIP3} (\text{dBm}) + P_{NF} (\text{dBm})}{3})$$

$$\text{et : } P_{noise_out} (\text{dBm}) = P_{noise_in} (\text{dBm}) + G_{rx} (\text{dB}) + NF_{rx} (\text{dB}) = -90 \text{ dBm.}$$

On déduit **SFDR = 73 dB.**

Critique du SFDR : le SFDR est défini à partir du plancher de bruit. Or, pour notre application nous souhaitons garantir un taux d'erreur minimum en réception. Il ne suffit pas d'obtenir un SNR_{out} supérieur à 0 dB.

Il est plus intéressant de calculer la **dynamique** du récepteur, pour laquelle le taux d'erreur minimum est garanti. La dynamique est définie comme le rapport entre la puissance maximale en entrée et la sensibilité du récepteur : $Dyn\ (dB) = P_{in_max}\ (dBm) - P_{in_min}\ (dBm) = 59\ dB$.

5) Bilan de liaison : calcul de la portée de transmission

On note D_{max} la distance maximale de transmission pour laquelle le taux d'erreur en réception est garanti. Il s'agit de la distance pour laquelle le récepteur reçoit un signal de puissance égale à sa sensibilité.

a) Calcul de la portée de transmission en espace libre

On suppose que l'on utilise des antennes omnidirectionnelles. Le modèle de perte en espace libre est :

$$PL(d, f_0)\ (dB) = 20 \log\left(\frac{4\pi f_0}{c}\right) + 10n \log(d)$$

avec $n = 2$.

Il s'agit d'une situation idéale où les seules pertes sont dues à la dispersion de l'énergie dans toutes les directions (antenne omnidirectionnelle).

Le bilan de liaison est : $P_{tx}\ (dBm) = PL(D, f_0)\ (dB) + P_{rx}\ (dBm) = PL(D_{max}, f_0) + P_{in_min}$ (sensibilité)

On a :

- $P_{tx} = 14\ dBm$
- $P_{in_min} = -92\ dBm$
- $f_0 = 2.45\ GHz$

En résolvant l'équation on trouve **$D_{max} = 1930\ mètres$** .

b) Calcul de la portée avec modèle de pertes "one slope"

On utilise le même modèle de perte que précédemment mais avec $n = 3$.

On obtient **$D_{max} = 155\ mètres$** .

III) Spécification générale

La spécification générale retenue est :

Puissance d'émission : 14 dBm.

Sensibilité en réception : -92 dBm.

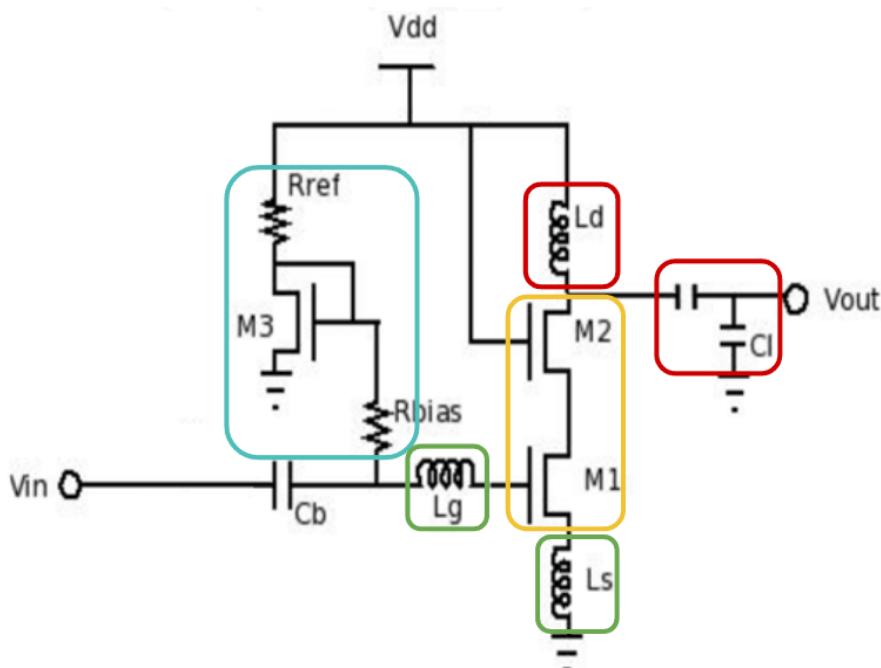
1.1 LNA

Le LNA est un amplificateur à faible bruit, il est très souvent utilisé dans les télécommunications hautes fréquences dans la chaîne de réception. C'est donc dans ce but que nous l'utilisons dans ce projet. Il est caractérisé par deux paramètres, le bruit qu'il va ajouter à au signal d'entrée et le gain à sa fréquence fonctionnement. Le LNA a pour but d'amplifier le signal reçu par l'antenne en ajoutant le moins de bruit possible. Si du bruit s'ajoute au signal alors le rapport signal sur bruit sera diminué et le traitement des données sera compliqué pour la suite de la chaîne de réception. Notre amplificateur devra donc avoir ses meilleures performances pour une fréquence de 2.45GHz.

Dans ce projet, le LNA de la chaîne de réception sera positionné en entrée des mixers des voies I et Q. Lors de la phase de test, nous mesurerons toute la chaîne de réception et il sera donc impossible de valider les performances des mixers seuls. Pour pallier cela nous allons réaliser un LNA single qui sera fabriqué sur une autre puce. Le LNA single et le LNA de la chaîne de réception doivent avoir les mêmes performances afin qu'on puisse isoler le mixer dans la chaîne de réception.

1.1.1 LNA Single (Romain HUC)

L'une des caractéristiques principales du LNA single c'est qu'il sera donc tout seul dans un boîtier, donc en relation avec aucun autre bloc. Il doit également être adapté à 50 Ohms en entrée et en sortie.



Éléments constituant notre LNA single :

- Réseau d'adaptation entrée
- Réseau d'adaptation sortie
- Miroir de courant
- Amplificateur

Nous retrouvons également une capacité C_b , qui va nous permettre de réaliser des mesures avec n'importe quel appareil. Si elle n'était pas intégrée à notre circuit, une capacité aurait dû être branchée entre notre boîtier et l'appareil de mesure.

Tout le défi avec le LNA single est d'arriver à avoir un gain élevé tout en restant adapté en entrée ainsi qu'en sortie. Comme on peut le voir sur la figure ci-dessus, notre réseau d'adaptation en entrée est formé de L_s et L_s , tandis qu'en sortie, il est composé d'une inductance L_d et d'un diviseur capacitif C_i . Étant donné qu'il est adapté en sortie, le gain dans le cahier des charges ne peut pas être comparable à notre gain du LNA single car le diviseur capacitif en sortie va casser notre gain. Pour pouvoir faire une comparaison, il faudra évaluer le ratio de notre diviseur capacitif pour en déduire notre gain avant ce dernier.

Rappel des spécifications attendues du LNA :

	NF (dB)	Gain (dB)	Pc1dB (dBm)	Conso (mA)	IIP3 (dBm)
Cdc	3.5	17	-2	5	13

Calculs théoriques pour les deux transistors :

Etant donné que nous connaissons le facteur de bruit et notre facteur de qualité, nous pouvons en déduire le gm :

$$F = 1 + \frac{\gamma}{50 * gm} * \frac{1}{Qe^2}$$

$$gm = \frac{\gamma}{50 * (F - 1)} 50 * \frac{1}{Qe^2} = 24,24 mS$$

Connaissant le gm , nous pouvons calculer le W et le V_{gs} :

$$W = \frac{L_{min} * gm^2}{4 * Kn * Id} = 128 \mu m$$

$$V_{gs} = \frac{2 * Id}{gm} + V_{tn} = 0.98V$$

Les calculs des autres composants ont aussi été fait :

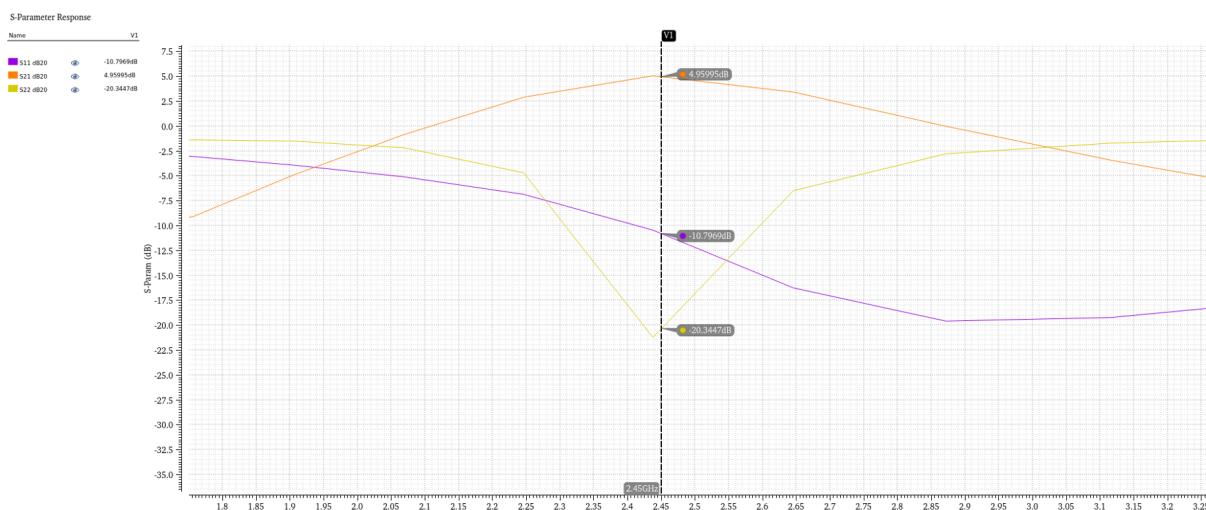
gm	Vgs-Vtn	Vgs	W	W/L	Cgs	wt	L1	L2	C
24.24m S	0.413V	0.98V	128 μm	367	182fF	3.7E10 rad/s	5.15nH	1.34nH	1.41pF

Nous avons donc rentré nos résultats théoriques dans les paramètres de nos composants. Le gain était moyen et nous n'étions pas vraiment adapté, que ce soit en entrée ou en sortie, mais cela nous a donné une base sur laquelle nous avons fait évoluer notre circuit en essayant d'avoir le meilleur gain, tout en respectant toutes les contraintes imposées.

Adaptation continue au réseau 50 Ohms :

Il faut savoir que lorsque la valeur d'un composant augmente ou diminue, le comportement en entrée et en sortie de notre LNA va varier. Par exemple, si je suis adapté en sortie mais que je fais une modification sur mon adaptation en entrée, alors il est fortement possible que je ne sois plus adapté en sortie. C'est donc pour cela, que nous allons devoir procéder par étape, c'est-à-dire commencer avec des composants idéaux, puis ajouter petit à petit des parasites comme le passage d'une capacité idéale vers un modèle réel, tout en ré adaptant continuellement en entrée et en sortie dès que des parasites s'ajoutent. En plus de cette contrainte, il faut arriver à faire les bons compromis, en essayant de mettre les plus petites inductances possible car, sur notre layout, ce sont elles qui vont prendre le plus de place et en microélectronique, les layouts doivent avoir une faible surface car le coût d'un circuit est calculé en fonction de ce dernier paramètre.

Voici donc les premiers résultats obtenus avec des composants idéaux :

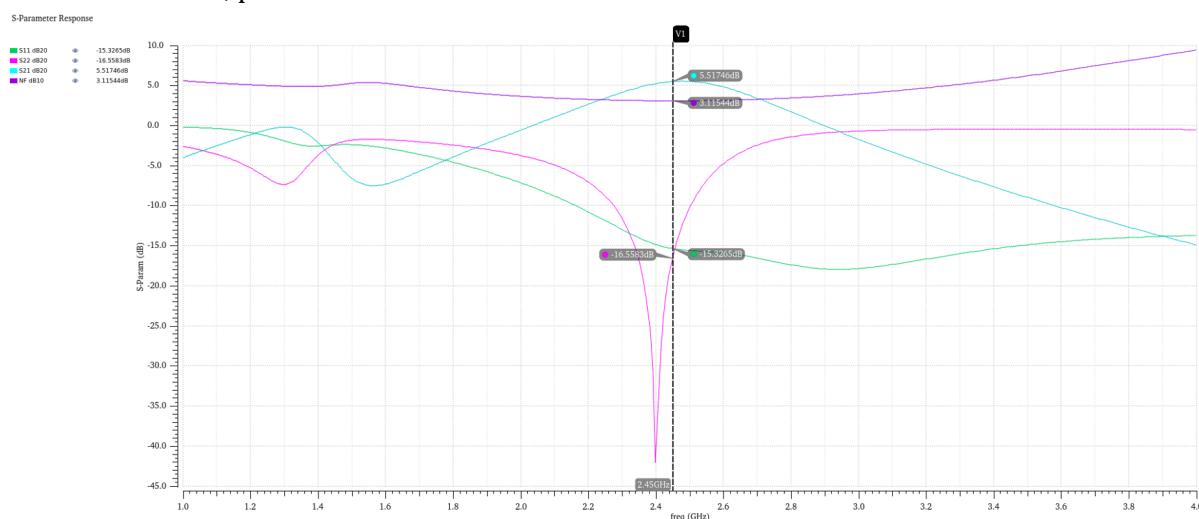


	NF (dB)	Gain (dB)	Pc1dB (dBm)	Conso (mA)	IIP3 (dBm)
Résultat	3.1	4.95 = ~15.63 (avant le diviseur capacatif)	-2.69	8	14.42

Nous avons donc remplacé petit à petit nos composants idéaux par des composants avec des parasites, puis nous avons ajouté des parasites, correspondant aux pads (capacité) et au wire bonding (inductance) qui feront la liaison entre notre die et le package. Des capacités de découplage seront également ajoutées sur les deux alimentations. Les performances du LNA se

sont écroulés suite à l'ajout de tous les derniers éléments, notamment à cause des inductances rajoutées sur GND. Pour diminuer l'impact de ce dernier, nous avons rajouté 2 pads GND sur lesquels viendront se poser d'autres fils, ce qui permettra de mettre en parallèle les inductances et de diminuer sa valeur équivalente. L'ajout de ces parasites ont donc bien évidemment changé notre adaptation en entrée ainsi qu'en sortie, la valeur de nos inductances et capacités d'adaptation ont dû être modifiées.

A la suite de ces changements, le gain obtenu était assez bon, 6.3dB à 2.45GHz, mais au détriment de la stabilité, car en basse fréquence, notre facteur de stabilité descendait en dessous de 0.6. Pour gagner en stabilité, des petites résistances de 4 Ohms ont été ajoutées au VDD et au GND, ce qui a permis de retrouver un bon facteur de stabilité, mais nous a fait perdre en gain et en consommation, passant à 5.5dB et 11mA.



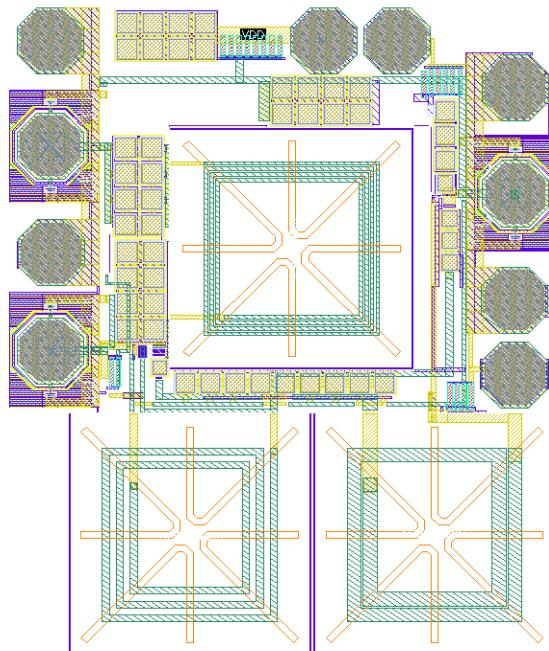
Layout

Comme il a été dit précédemment, les inductances et les pads sont les composants les plus imposants sur notre circuit. Nous devons donc arriver à trouver la disposition la plus optimum pour prendre le moins de place possible, tout en garantissant que le chemin RF sera également le plus court que l'on puisse faire pour éviter de trop grandes pertes. Tous les autres composants, à savoir résistances, capacités et transistors se placeront aux endroits disponibles. Le chemin RF étant prioritaire, il faudra qu'il soit le plus court, le moins résistif, donc éviter trop de changements de métallisation et passage par des vias ainsi que des pistes trop fines et essayer de router au niveau le plus haut. Les effets capacitifs sont également à éviter, il est donc important d'être éloigné du plan de masse au M1 ainsi que d'avoir des lignes RF qui longent une métallisation de VDD par exemple.

Une fois le layout terminé, nous avons dû le faire passer des tests pour vérifier que nous respectons toutes les règles de layout imposées par le fondeur (DRC) et qu'il n'y a pas de différence avec notre schematic au niveau électrique (LVS). Une fois ces deux tests passés avec succès, nous devons passer aux simulations post layout, qui permettent d'extraire les parasites de notre layout. A ce moment-là, nous avons eu d'énormes pertes, passant de 5.5dB de gain sans extraction à -0.6dB avec extraction. Nous avons donc amélioré le layout pour réduire les pertes au maximum, prendre en compte le couplage des inductances, tourner une inductance de 90°

pour réduire le chemin RF avant l'amplification, faire deux plan de masse au M1, un pour GND et un autre pour la masse avant les résistances ajoutées pour la stabilité, élargissement de certaines pistes, etc... Toutes ces modifications ont permis de remonter jusqu'à 2.8dB mais ce n'était toujours pas assez. La stratégie qui a donc été adoptée, est de se servir des parasites induits par le layout, pour diminuer les valeurs de nos composants (inductance en entrée et diviseur capacitif en sortie). Cette stratégie nous a donc permis de diminuer la taille de notre inductance la plus grosse et donc de diminuer la taille de notre circuit, tout en ayant une meilleure adaptation et donc un meilleur gain. Ce dernier est de 4.73dB, ce qui équivaut à ~ 15 dB avant le diviseur capacitif.

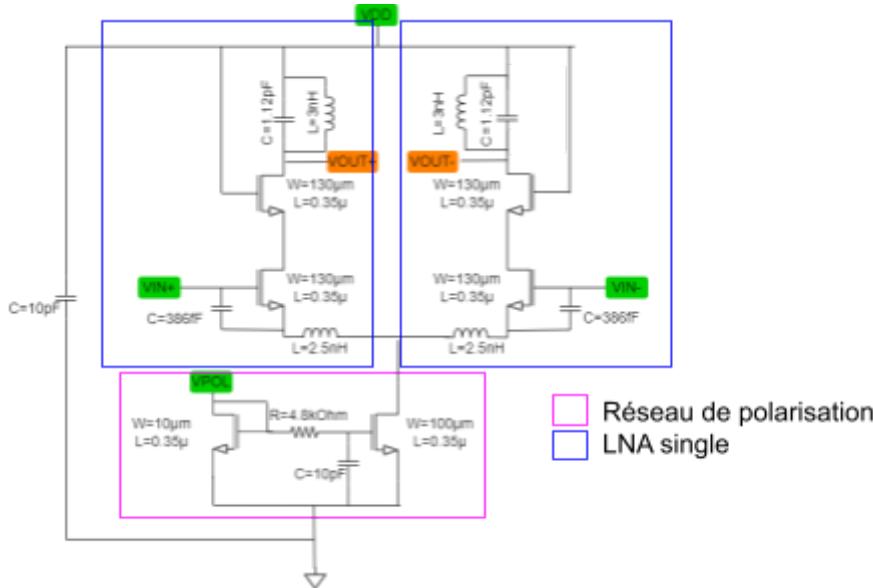
	NF (dB)	Gain (dB)	Pc1dB (dBm)	Conso (mA)	IIP3 (dBm)
Résultat	3.7	4.73 (~ 15)	-5.95	13.25	12.93



1.1.2 LNA Diff (Manon TABARDEL)

Fonctionnement et dimensionnement

La structure choisie du LNA différentiel est la suivante:



Structure du LNA différentiel

On peut noter que cette structure est très similaire à la structure single. En effet, elle contient 2 LNA single ainsi qu'un réseau de polarisation.

Pour dimensionner les deux LNA single permettant la partie différentielle je me suis appuyée sur le LNA standalone. En effet, pour que la comparaison des mesures entre le LNA single et la chaîne de réception ait un sens il faut que les LNA aient la même structure et que les transistors aient le même gm . J'ai cependant adapté les valeurs des composants au fur et à mesure des simulations.

Le LNA étant différentiel, l'impédance d'entrée est de 100Ω contrairement à la partie précédente où elle était de 50Ω . De plus, l'entrée du LNA est différentiel, elle est fournie par un balun présent sur la carte de test il n'était donc pas à designer.

Le LNA différentiel ne possède pas de réseau d'adaptation en sortie. En effet, il est positionné en entrée de la chaîne de réception et sera suivi des mixers passifs. Pour l'étude du LNA différentiel j'ai réalisé mes simulations en considérant une charge de $1\text{k}\Omega$, cela correspond à l'impédance d'entrée des mixers passifs.

Dans l'étude théorique du LNA single, Romain a considéré que la consommation du LNA est de 5mA . Pour dimensionner le circuit de polarisation du LNA différentiel je me suis appuyée sur cette valeur. Pour que les 2 LNA mis en miroir soient traversés par 5mA il faut que le courant sortant du circuit de polarisation soit de 10mA . Cela implique un rapport de 10 entre la largeur des deux transistors du miroir de courant. De plus, la carte sur laquelle sera embarquée le récepteur Zigbee possède une résistance variable alimentée par une source de tension. Pour gagner en stabilité et s'affranchir des pertes de la résistance j'ai fait le choix d'alimenter le circuit de polarisation par l'alimentation de la carte et non par une résistance inclue dans le circuit.

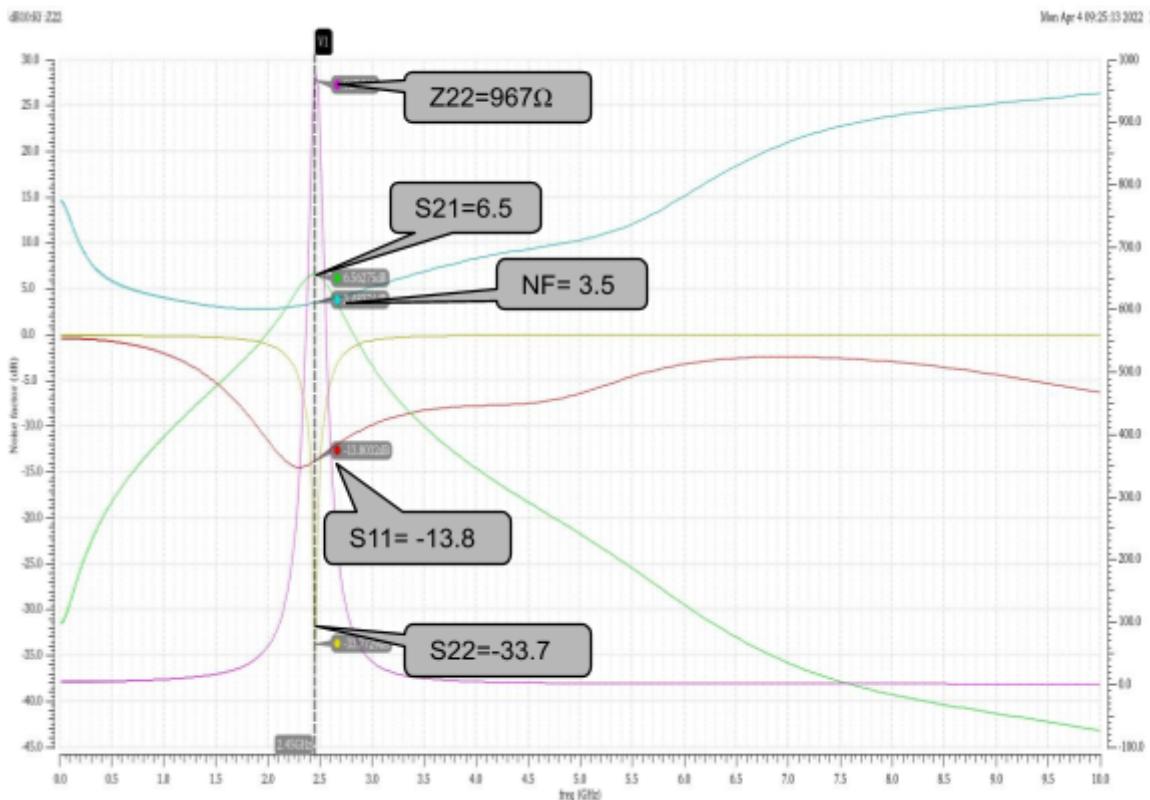
Pour réaliser le LNA différentiel j'ai réalisé l'étude et le layout des LNA single et du circuit de polarisation indépendamment puis je les ai assemblés. Pour faciliter l'étude, j'ai dans un premier temps réalisé un dimensionnement avec des composants idéaux puis avec des composants réels.

Voici ci-dessous le résultat des différentes simulations que j'ai réalisées.

	<i>Cahier des charges</i>	<i>Composants idéaux</i>	<i>Composants réels</i>
S11 (dB)		-19,5	-13,8
S21 (dB)		11,7	6,5
S22 (dB)		-5,9	-33,7
Gain (dB)	20	-22,6	20,9
ICP (dB)	-2	-6,3	-2,1
IIP3 (dBm)	13	20	14,1
NF (dB)	3,5	1,9	3,5
Kf	>1	21	46

Résultats de simulations dans différents cas

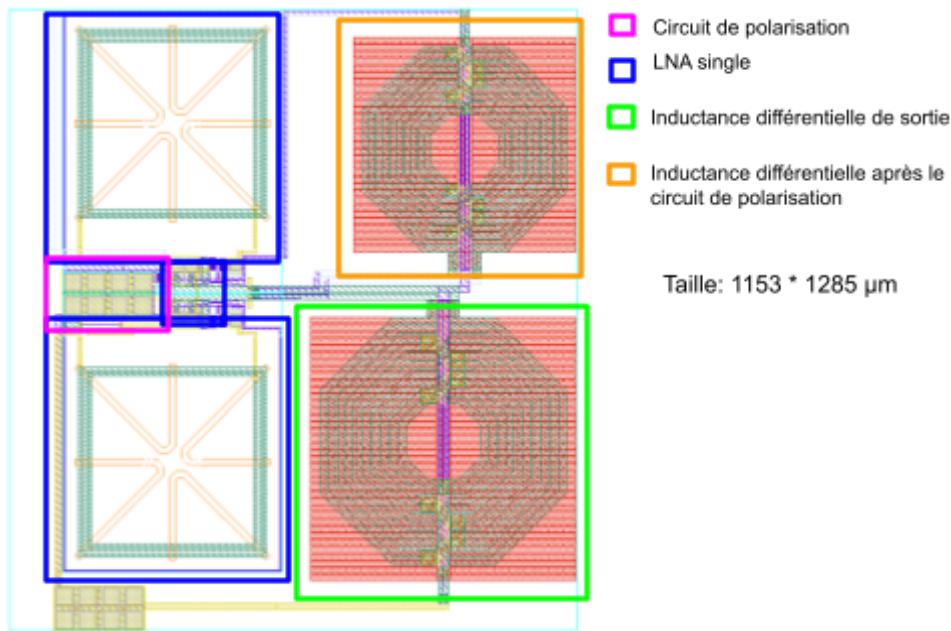
On remarque que les performances atteintes correspondent au cahier des charges. On voit cependant que le fait d'utiliser des composants réels influe sur les performances du LNA. Pour compenser certaines des pertes induites par les composants réels j'ai dû changer les valeurs de certains composants. J'étais cependant limitée par les valeurs des inductances différentielles, j'ai donc dû dimensionner le LNA selon les valeurs imposées pour les inductances différentielles.



Résultats de simulation SP avec des composants réels et bonding+capadec

Layout et simulations post layout

Afin de simplifier le dessin du LNA différentiel, j'ai réalisé le layout d'un LNA single et du circuit de polarisation, je les ai ensuite assemblés dans un layout global.



Layout du LNA différentiel

Le LNA doit être le plus compact possible, j'ai cependant volontairement laissé de la place entre l'inductance de sortie du circuit de polarisation et le LNA single afin qu'on puisse mettre les mixers et les buffers de la chaîne de réception.

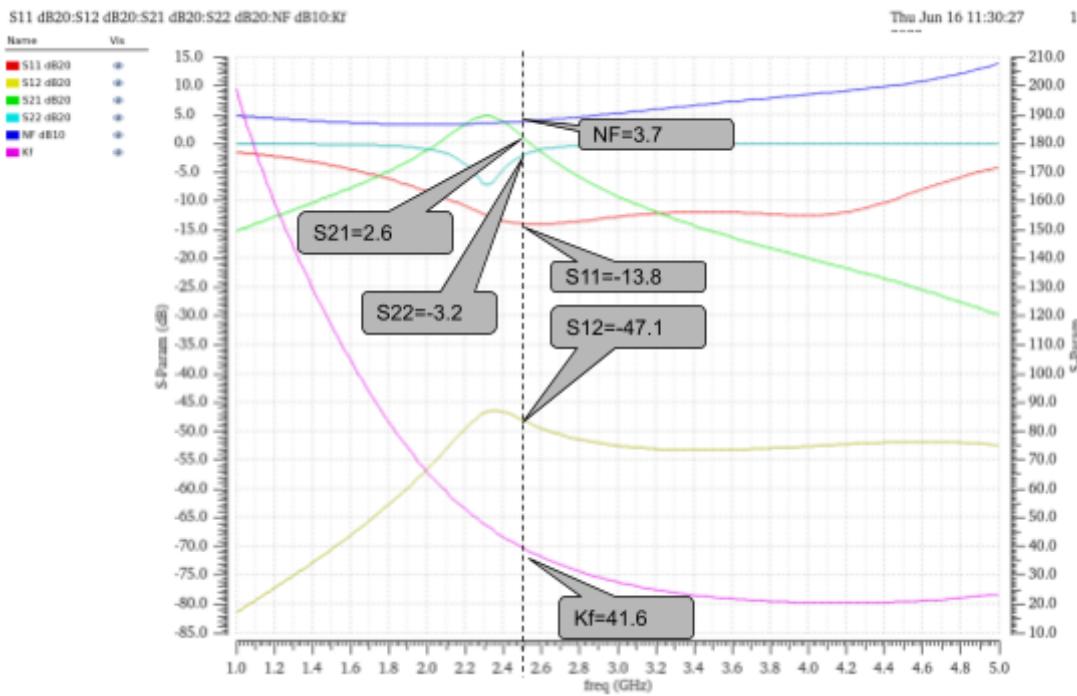
Lors de la réalisation de layout j'ai dû faire attention aux entrées et sorties du LNA. Il faut que les entrées soient sur les extrémités du layout afin d'être facilement reliées aux plots. Les sorties du LNA différentiel doivent être proche de l'endroit où seront positionnées les mixers et les buffers.

J'ai fait le choix de ne pas réaliser le plan de masse et le pad ring au niveau du LNA différentiel et de l'avoir réalisé lors de l'assemblage de la chaîne de réception.

Après avoir fait le layout du LNA différentiel j'ai fais des simulations afin de voir l'impact des parasites induits par mon layout. Les résultats sont les suivants:

	<i>Cahier des charges</i>	<i>Post layout</i>
S11 (dB)		-13,7
S21 (dB)		2,6
S22 (dB)		-3,2
Gain (dB)	20	17,9
ICP (dB)	-2	-4
IIP3 (dBm)	13	10,9
NF (dB)	3,5	3,7
Kf	>1	24

Résultats des simulations post-layout



Résultats de simulation SP après le layout

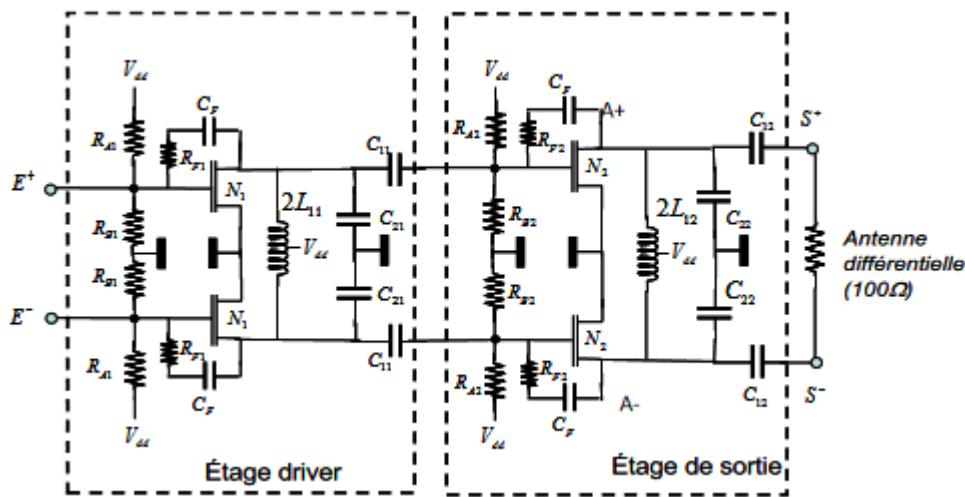
On remarque que les performances du LNA sont moins bonnes après le layout. J'ai volontairement arrêté l'étude du LNA différentiel seul après les simulations post layout et je l'ai assemblé aux mixers. En effet, les mixers ayant un impact sur le LNA avec son impédance d'entrée il était plus intéressant de les simuler ensemble assez rapidement pour avoir le temps d'adapter le layout global de la chaîne de réception.

1.2 PA

L'amplificateur de puissance (PA) fait partie de la chaîne d'émission de l'émetteur Zigbee. Dans notre projet, le PA permet d'amplifier un signal issu du mélangeur et de délivrer une puissance sur une impédance de 100Ω qui représente une antenne différentielle. Nous avons réalisé deux PA : un PA différentiel qui sera associé au mixer et un PA single qui servira au test sous pointe sur wafer.

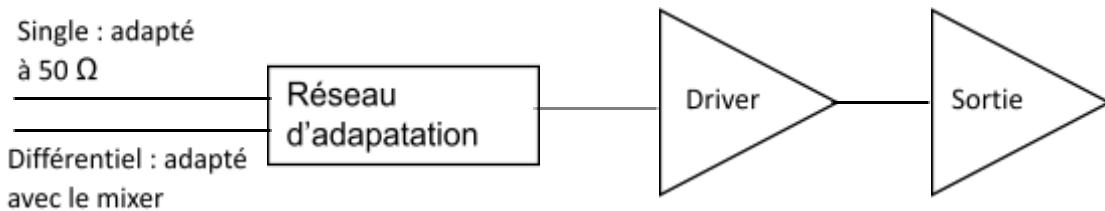
Nous allons travailler à une fréquence de 2.45GHz. Le cahier des charges que nous devons respecter à cette fréquence est le suivant : nous devons avoir un gain minimum de 15 dB, un OCP1dB (Output Compression Point) de 13 dBm et une puissance maximum d'au moins 14 dBm.

L'architecture de notre PA différentiel est la suivante :



Au niveau schéma bloc, notre PA peut se résumer de la manière suivante.

Un bloc d'adaptation qui permet d'adapter notre PA single à 50Ω (impédance de l'appareil de mesure), et d'adapter le PA différentiel au mixer. Un bloc pour l'étage driver et un autre bloc pour l'étage de sortie.



Notre PA est conçu au niveau des transistors et avec des éléments réels.

Le cahier des charges pour le PA est le suivant :

	Gain (dB)	OCP1dB (dBm)	Psat (dBm)
PA single	15	10	11

PA différentiel	15	13	14
-----------------	----	----	----

Nous avons commencé par faire nos calculs avec des composants idéaux pour avoir un ordre de grandeur des valeurs de nos composants.

Nous avons commencé par calculer les valeurs des composants du bloc de sortie à savoir C12 C22 et L12. Pour cela nous avons commencé par calculer la valeur de la partie réelle Rp de l'impédance différentielle ramenée entre les drains des MOS N2 pour avoir une puissance développée de 17 dBm avec une charge de 100Ω : nous prenons une marge de 3 dB pour prendre en compte les futures pertes des composants idéaux. Nous avons donc choisi d'atteindre une puissance de 14 dBm pour le PA single et 17 dBm pour le PA différentiel.

$$Rp_{single} = \frac{Vdd^2}{P} = \frac{3^2}{10^{14/10} * 10^{-3}} = 179 \Omega$$

$$Rp_{diff} = \frac{Vdd^2}{P} = \frac{3^2}{10^{17/10} * 10^{-3}} = 359 \Omega$$

Nous calculons le facteur de qualité Q qui nous permettra de calculer ensuite les valeurs de nos composants.

$$Q = \sqrt{\frac{Rp}{Rload}} = \sqrt{\frac{179}{50}} = 1.6$$

$$C12 = \frac{1}{2\pi f * Q * 50} = \frac{1}{2\pi * 2.45 * 10^9 * 1.6 * 50} = 808 fF$$

Pour calculer la capacité C22, il faut connaître la capacité totale ainsi que la capacité équivalente C12' grâce au facteur de qualité qui correspondent à des capacités fictives qui nous permettent de calculer la capacité réelle. L12 étant connu et vaut 3 nH.

$$C12' = \frac{C12 * Q^2}{1 + Q^2} = \frac{808 * 1.6^2}{1 + 1.6^2} = 583 fF$$

$$Ctot = \frac{1}{(2\pi f)^2 * L12} = \frac{1}{(2\pi * 2.45 * 10^9)^2 * 3 * 10^{-9}} = 1.41 pF$$

$$C22 = Ctot - C12' = 1410 - 583 = 824 fF$$

Pour ce qui est du courant fourni par le transistor on a :

$$Idc = \frac{Vdd}{Rp} = \frac{3}{179} = 16.7 mA$$

$$Irf = \frac{Vrf}{Rp} = \frac{3}{179} = 16.7 \text{ mA}$$

$$Imax = Idc + Irf = 33.5 \text{ mA}$$

Nous calculons maintenant la tension à appliquer sur la grille du transistor ainsi que la largeur de sa grille :

$$Vgso = \frac{Idc}{Imax} * 4 * Vdsmin + Vtno = \frac{0.0167}{0.0335} * 4 * 0.3 + 0.57 = 1.17 \text{ V}$$

$$W = \frac{Idc * L}{Kn * (Vgso - Vtno)^2} = \frac{0.0167 * 35 * 10^{-7}}{5.5 * 10^{-5} * (1.17 - 0.57)^2} = 296 \mu\text{m}$$

Pour les résistances de polarisation RA et RB on choisit respectivement 6kΩ et 4kΩ pour une tension d'alimentation de 3V.

La taille maximale des transistors de la technologie utilisée étant de 150μm, nous avons dû mettre 2 transistors de W= 200μm en parallèle pour satisfaire le cahier des charges.

Les résistances Rrf1 et Rrf2 permettent d'améliorer la stabilité de notre système. En choisissant ces valeurs de Rrf nous avons Kf (facteur de qualité) >1 quelque soit la bande fréquence.

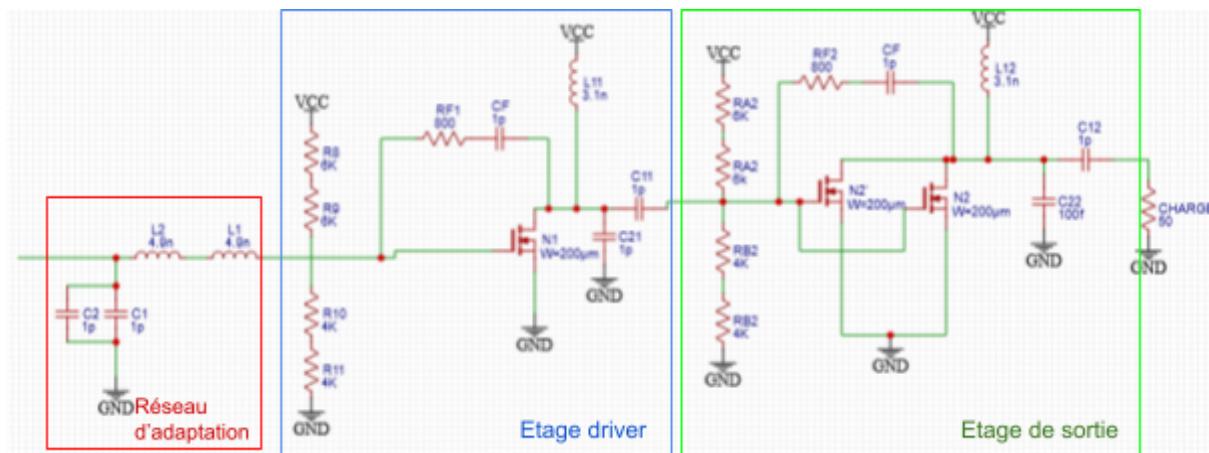
Après simulation du schéma avec les valeurs théoriques, nous constatons que nous sommes loin du compte. Toutes les valeurs sont donc optimisées grâce à des simulations paramétriques.

Les calculs sont effectués ici pour l'étage de sortie. Un raisonnement identique est utilisé ensuite pour l'étage driver. A la différence que l'impédance de charge initial correspond cette fois à l'impédance d'entrée de l'étage de sortie.

1.2.1 PA single (Manon CREVEL)

Le PA single sera testé seul sur wafer, donc en relation avec aucun autre bloc. Il doit être adapté à 50 Ohms en entrée et en sortie pour permettre les mesures avec un VNA.

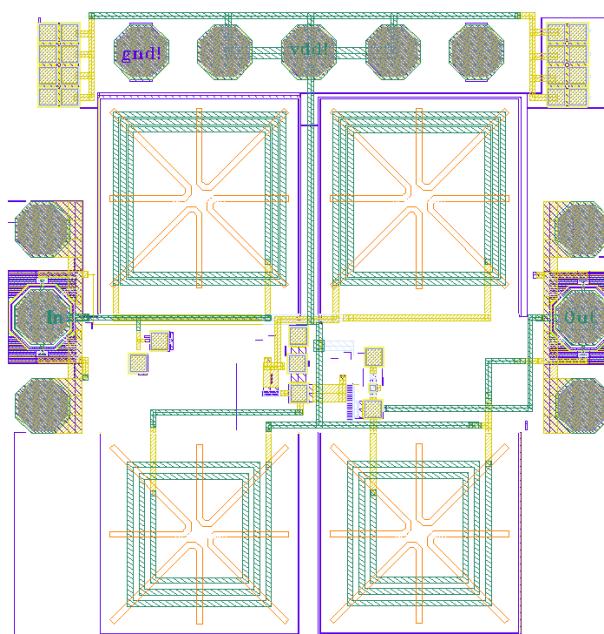
Voici les 3 blocs du PA single avec les valeurs optimisées :



La principale contrainte du PA single est d'être adapté en entrée et en sortie tout en respectant le cahier des charges.

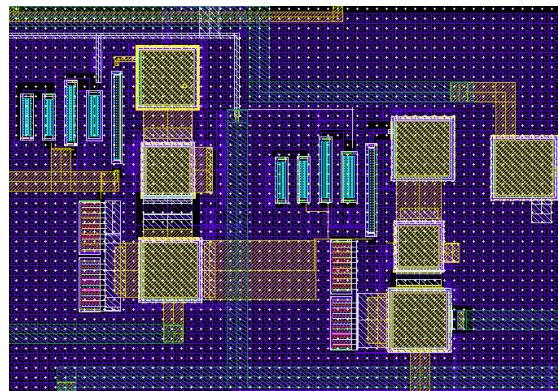
Le PA single sera testé sous pointe, il ne sera pas mis en boîtier. Il n'y a donc pas besoin d'ajouter des wirebonds cependant il faut prendre en compte les contraintes liées aux pointes qui ont un pitch de 150 µm. Ainsi dans le layout, il faut avoir des pads DC qui soient séparés de 50 µm (3 pads VDD entouré de pads GND).

Le layout du PA s'est fait de la façon suivante : on commence par agencer les composants intelligemment pour utiliser le moins de place possible et faire des pistes de métal les moins longues possibles. Ce sont les inductances, les sondes RF et les pads qui prennent le plus de place dans notre layout. On a donc placé ces composants en 1er de façon à délimiter la surface du layout et de façon qu'il n'y ait pas de couplage entre les inductances. Ensuite, on passe au routage des blocs du PA, on veille à ce que le chemin RF soit le plus court avec des pistes larges et le moins de changement de niveau de métallisation pour éviter les pertes. On place ensuite les autres composants dans les endroits disponibles. On passe régulièrement des DRC et LVS pour vérifier que nous respectons toutes les règles de layout imposées par le fondeur et qu'il n'y a pas de différence entre notre layout et le schematic. Pour finir, on passe un DRC avec les règles d'antenne qui vérifient que les grilles de transistors ne sont pas connectées à des métaux de trop grande dimension et qui pourraient produire de décharges électrostatiques. On réalise un plan de masse pour réduire les boucles de courant et assurer une bonne masse en métal 1



Layout du PA single

Le surface du PA single

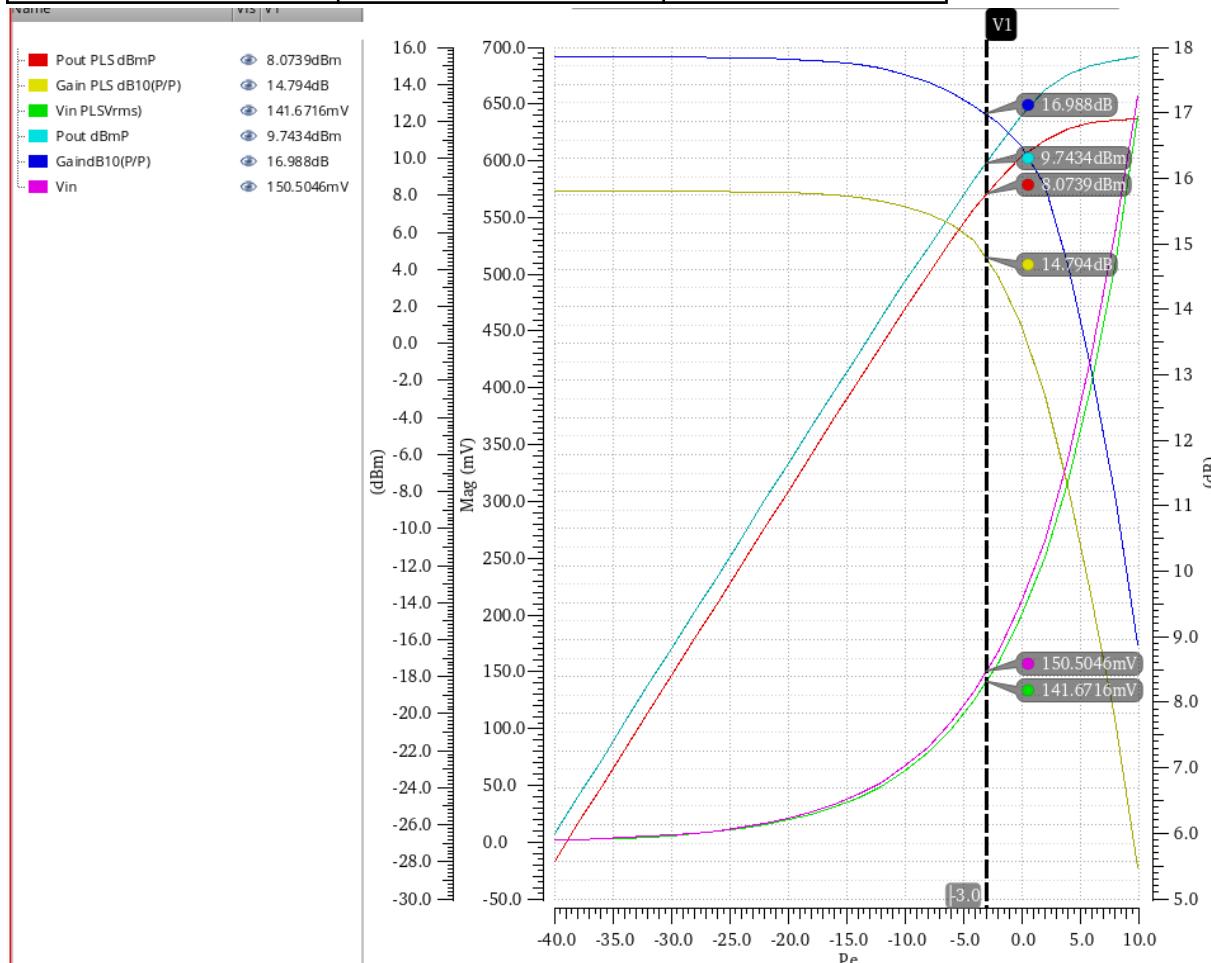


Zoom entre les inductances

Quand le DRC et le LVS sont corrects pour le layout, on peut passer à l'extraction du layout et réaliser des simulations post layout.

Le tableau ci-dessous répertorie les performances post design et post layout :

PA single	Simulations post design	PLS
Consommation [mW]	201.0 (67mA/3V)	209.7 (70mA/3V)
OCP1dB(dBm)	11.05	8.14
Gain (dB)	16.77	14.78
Psat (dBm)	15.50	12.00
S11 (dB)	-2.80	-5.55
S12 (dB)	-38.33	-41.31
S21 (dB)	-6.41	-8.00
S22 (dB)	-4.74	-4.71
Zin [Ω]	$50 + j24$	$46 + j38$



On remarque qu'on a perdu environ 2 dB sur les performances de notre PA. Ceci est dû aux parasites induits par le layout (voir explication 2.2.2).

Nous sommes en dessous du cahier des charges pour le point de compression en sortie mais le gain et la puissance max du PA est satisfaisante. Nous sommes raisonnablement désadaptés en entrée du PA.

1.2.2 PA Différentiel (Nathan LOISY)

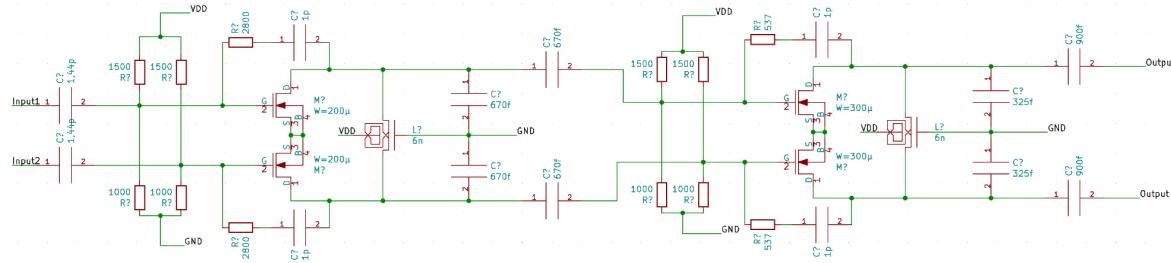


Schéma finale PA dif

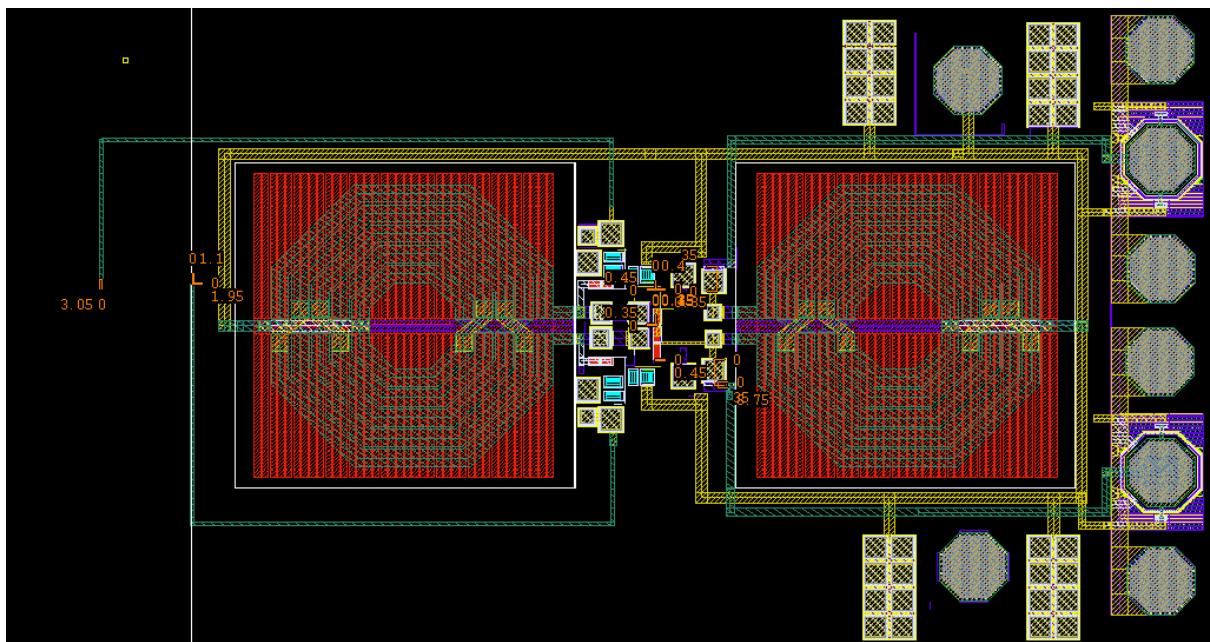
Le schéma ci-dessus montre les valeurs des différents composants implantés dans le layout.

Le PA différentiel doit s'interfacer en sortie avec une antenne différentiel d'impédance 100Ω en sortie et avec le MIXER actif en entrée. L'impédance d'interface avec le MIXER a été fixée à 800Ω pour la partie réelle et 300Ω de partie imaginaire.

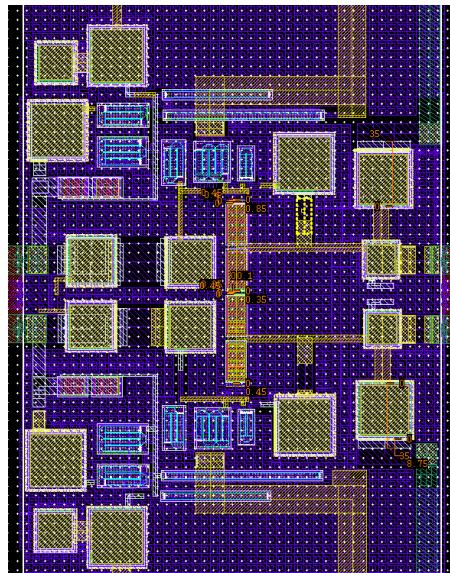
En ce qui concerne le layout, le routage a été limité principalement par la taille des inductances et des plots.

On retrouve entre les inductances, en haut et en bas, les deux parties + et -, complètement symétriques.

On note également la place gardée à gauche, en haut et en bas pour le mixer actif et ses plots.



Layout PA dif seul

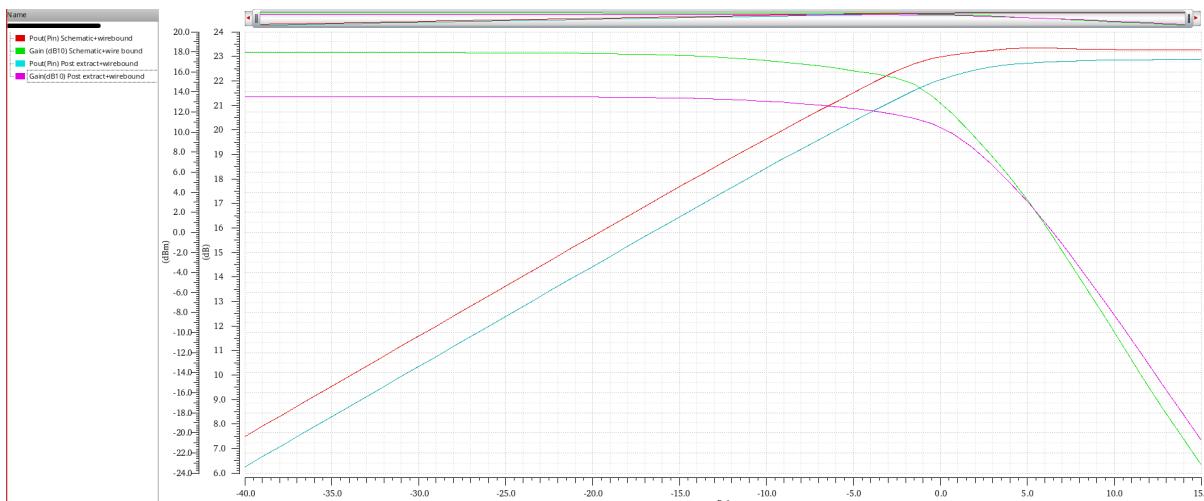


Zoom entre les inductances layout PA dif

En ce qui concerne les performances on obtient donc :

PA dif	Schematic + wirebound	PLS + wirebound
Conso (mW)	284,5 (94,85mA/3V)	256 (85,45mA/3V)
OCP1 (dBm)	15,8	14,25
Gain	22,17	20,47

Psat	18,25	17,25
------	-------	-------



Puissance et gain en sortie en fonction de la puissance d'entrée

On note d'abord que les simulations ont été réalisées avec un modèle de wire bonding sur les sorties qui permet de simuler les pertes causées par le wire bonding entre le silicium et le boîtier.

Ensuite on remarque que toutes les valeurs ont baissé puisque le layout introduit un certain nombre de parasites. Principalement les parasites capacitifs dû aux différentes pistes, et qui viennent modifier les impédances et donc compromettre les adaptations. On retrouve également des pertes résistives du aux Via et aux pistes, ces pertes sont d'autant plus importantes que nous transportons une puissance électrique considérable.

On peut évaluer ces pertes à environ 2dB en moyenne sur les performances ce qui est un chiffre prévisible et satisfaisant. Le PA répond donc parfaitement au cahier des charges (voire le surpasse largement).

1.3 Mixer

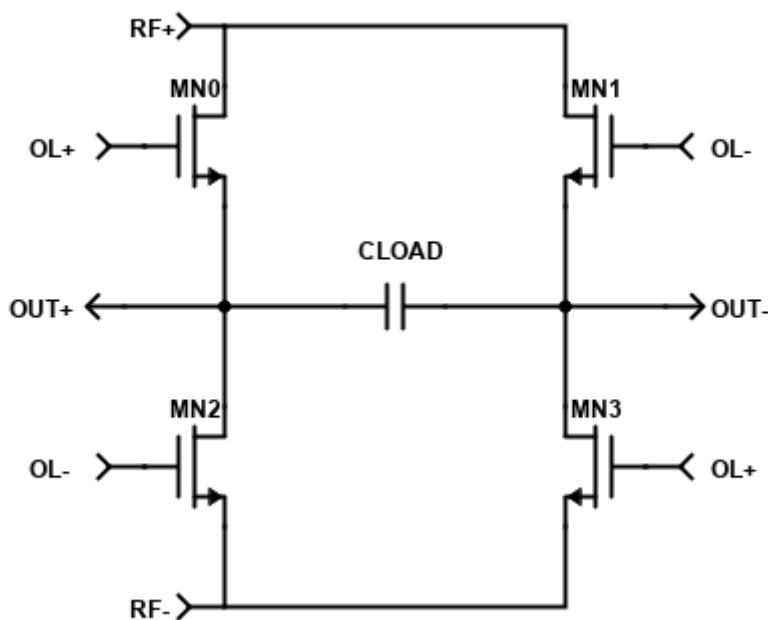
Le mixer est la partie du système qui permet de réaliser la transposition du signal en fréquence. Il est donc présent à deux endroits de la chaîne, en émission avant le Power Amplifier et en réception à la suite du Low Noise Amplifier. Pour des raisons d'architecture, deux types de mixers seront utilisés.

En émission, le mixer sera actif, car cette architecture permet de sortir un signal transposé de puissance constante tout en tenant de bonnes performances, ce qui convient très bien à l'application visée. En effet, on souhaite faire travailler le Power Amplifier à son point de compression, il est donc nécessaire pour le mixer d'avoir une puissance de sortie constante. En réception, la puissance RF en entrée du mixer est très variable, et peut atteindre des seuils élevés. Il est donc dans ce cas nécessaire d'utiliser une architecture de mixer passif, qui peut donc supporter cette grande variation d'amplitude en entrée. Nous allons donc détailler dans les

deux parties suivantes les étapes de design, simulation et layout nécessaire à la conception de ces circuits.

1.3.1 Mixer Passif (Yann Zyzelewicz)

Le mixer présent dans la chaîne de réception sera donc réalisé par une structure passive, qui est simplement un pont en H, chargé par une capacité et piloté par les signaux d'OL. Il a donc deux entrées RF, qui sont les sorties différentielles du LNA, deux entrées OL déphasées de 180°, et deux sorties basse fréquence différentielles. Voici le design du circuit réalisé :



Le circuit est très simple, les 4 MOS présents sont identiques et fonctionnent en switch, permettant ainsi de réaliser la fonction de mélangeur. Les transistors présents dans l'architecture agissant en switchs, on peut ainsi retrouver aisément l'équation régissant la tension différentielle de sortie :

$$V_{OUT}(t) = \operatorname{sgn}(V_{OL}(t)) \cdot V_{RF}(t)$$

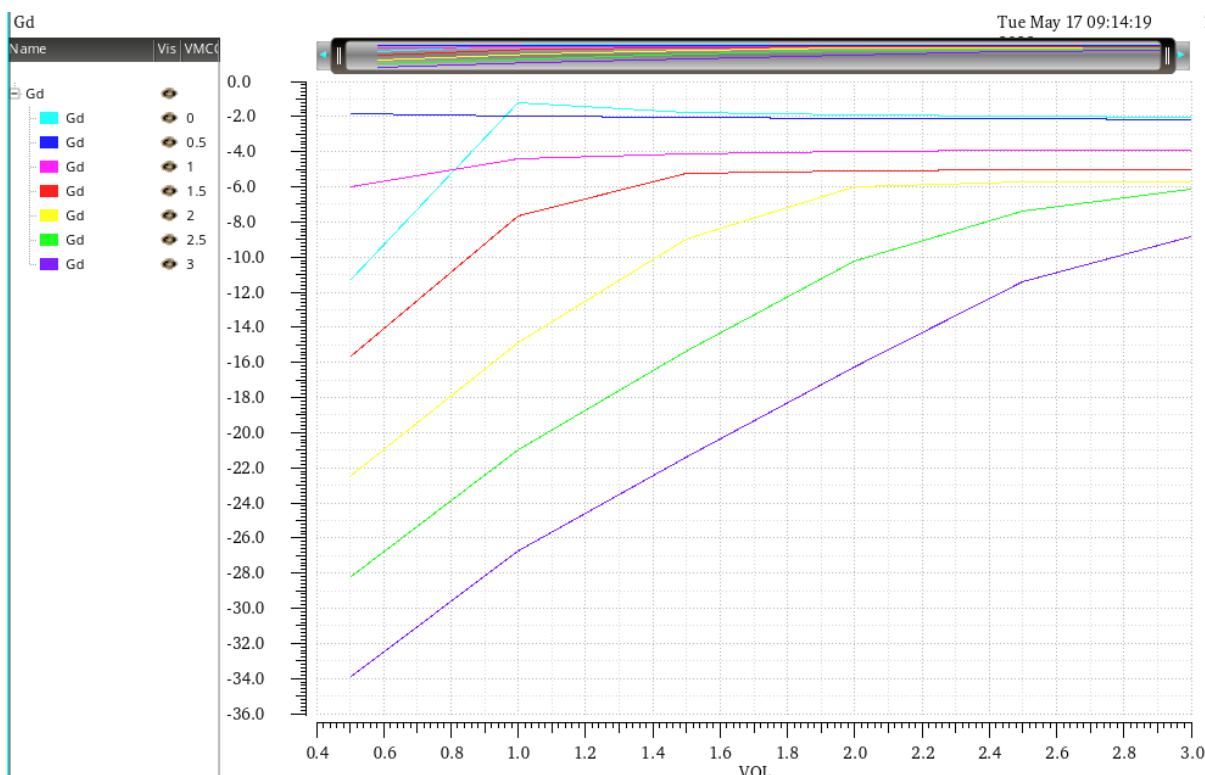
Avec cette structure, le signal RF n'étant pas amplifié, le gain de conversion est inférieur à 1, ce qui aura un impact négatif sur la chaîne de bruit du récepteur. L'axe principal de design du mixer sera donc d'avoir un gain le plus élevé possible, afin de minimiser son impact sur le bruit du récepteur.

Plusieurs éléments jouent sur le gain du mixer. Le premier est le rapport W/L des transistors utilisés. Dans l'hypothèse où l'on admet que l'impédance de sortie du LNA est bien inférieure à l'impédance d'entrée du mixer, un fort rapport W/L permettrait d'avoir un gain élevé. Cette hypothèse n'est cependant pas réaliste, car les impédances de ces deux éléments sont en réalité du même ordre de grandeur. Pour réaliser un bon transfert de tension, l'impédance d'entrée du mixer doit être la plus élevée possible, ou grande devant l'impédance de sortie du LNA, ce qui est réalisable avec un petit rapport W/L des transistors. On remarque que

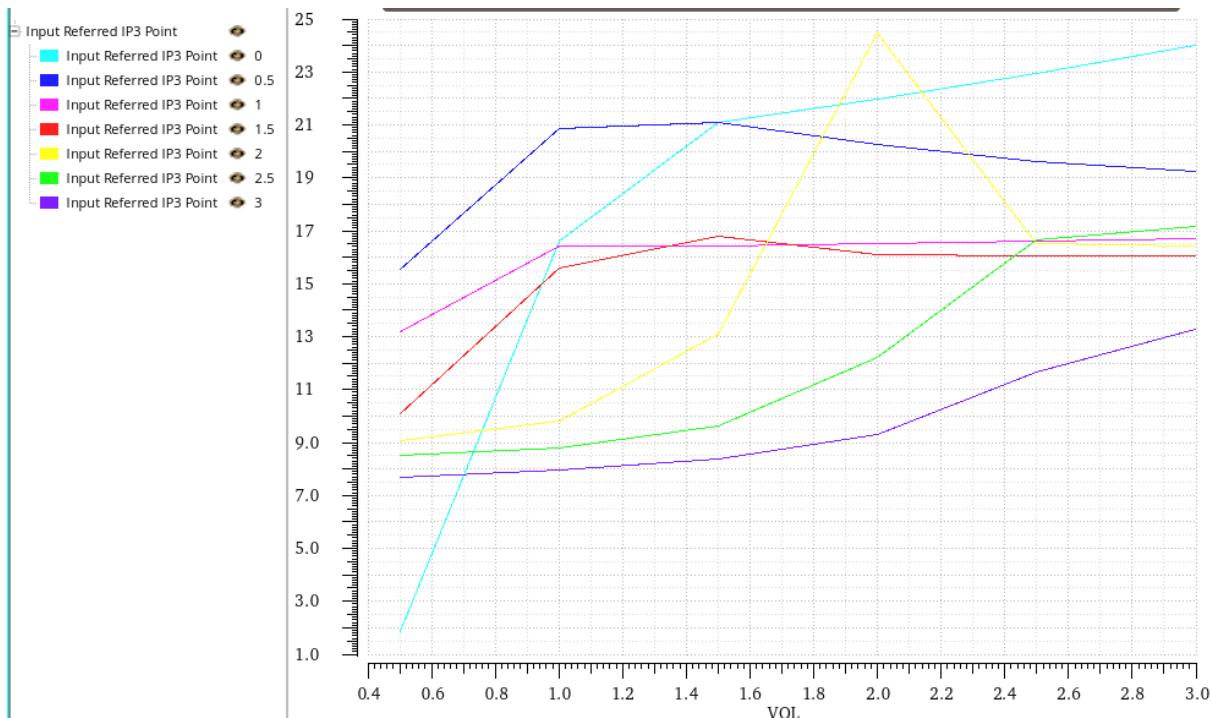
pour réaliser le même but, qui est d'avoir un gain élevé, deux directions contraires sont à prendre sur le W/L des transistors. Il est donc nécessaire de réaliser un compromis sur leur taille.

Une étude paramétrique a donc été réalisée pour observer l'impact du rapport W/L d'une part sur le gain du mixer, mais aussi sur l'IIP3, qui est un élément déterminant sur la qualité de la transmission. La longueur du canal étant fixée à $0.35\mu\text{m}$ sur les modèles RF des mos utilisés, cette étude a en réalité été réalisée en faisant varier leur W. On remarque que plus le rapport W/L est grand, meilleur le gain est. Cependant, l'IIP3 baisse avec l'augmentation du W/L. Les spécifications exigées sur le mixer sont un gain de -4 dB, et un IIP3 de 26 dBm. La largeur du canal nous permettant d'avoir le meilleur compromis IIP3/Gain est $W = 5 \mu\text{m}$, en atteignant un gain de 0dB, et un IIP3 de 10 dBm. Vis à vis du gain, augmenter davantage la largeur W le fait converger vers un seuil de 2 dB, augmenter le W pour ne gagner que peu de gain n'était donc pas envisageable en vue de la perte d'IIP3. Enfin, pour maximiser ce dernier, le W_{MIN} a été pris, afin d'obtenir le meilleur IIP3 possible.

Un autre élément déterminant du design du mixer passif est le choix des signaux d'OL utilisés. Nous avons la main sur deux paramètres, l'amplitude du signal OL et son mode commun. Dans un premier temps, il faut considérer le signal reçu du LNA. Le mode commun de ce dernier est de 0V. Des simulations paramétriques ont été réalisées afin de déterminer le mode commun et l'amplitude du signal OL. Cette étude a montré qu'un mode commun d'OL égal au mode commun du signal RF permettait le meilleur gain et IIP3 pour le mixer. Voici les courbes de l'étude réalisée, avec en abscisse l'amplitude du signal OL, et une variation du mode commun d'OL pour chaque courbe.



Gain en fonction de l'amplitude VOL, pour VMCOL allant de 0 à 3V



IIP3 en fonction de l'amplitude VOL, pour VMCOL allant de 0 à 3V

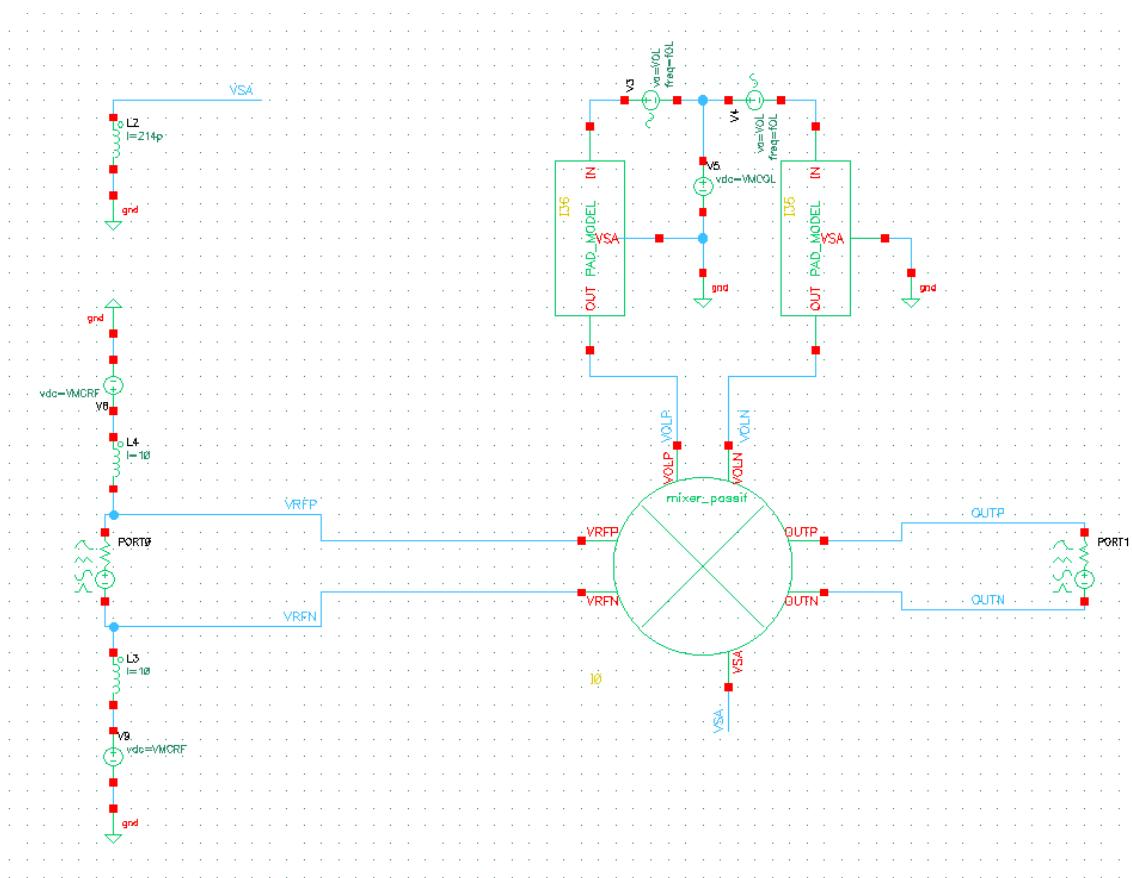
Ces courbes montrent que les meilleurs Gain et IIP3 sont atteints pour un mode commun d'OL à 0V (pour un mode commun RF de 0V) et une amplitude d'OL maximale, ici 3V. Ce sont donc les paramètres qui ont été choisis pour l'OL.

Enfin, la capacité de sortie du mixer permet de maîtriser la bande passante, que l'on souhaite être légèrement au-dessus de notre signal de fréquence intermédiaire afin de limiter le signal RF. Elle est ici d'une capacité de 1pF, ce qui nous donne une fréquence de coupure de quelques dizaines de MHz, suffisant pour atténuer le signal RF d'une 40aine de dB.

Simulations

Afin de réaliser les simulations pour extraire les performances du mixer, nous avons dû mettre en place un testbench approprié. En effet, le mixer réalisant une transposition en fréquence, les méthodes de simulation Harmonic Balance ont été utilisées. Ces méthodes de simulation permettent de simuler la transposition en fréquence avec une grande précision très rapidement comparé à ce que pourrait réaliser une simulation transitoire par exemple.

Dans la figure ci-dessous est montré le bench utilisé. Premièrement, le signal OL est modélisé par des sources de tension émettant des sinusoïdes déphasées de 180° à une fréquence de 2.45 GHz. Ensuite, le LNA est modélisé par un Port d'impédance 1k Ohms (donnée par le groupe du LNA). Les signaux de ce port ont été fixés respectivement à des fréquences de 2.451 GHz et 2.453 GHz afin de pouvoir réaliser les mesures d'IIP3. Enfin, le mixer est chargé par un port de très forte impédance, qui simule les grilles du buffer auquel il sera relié.



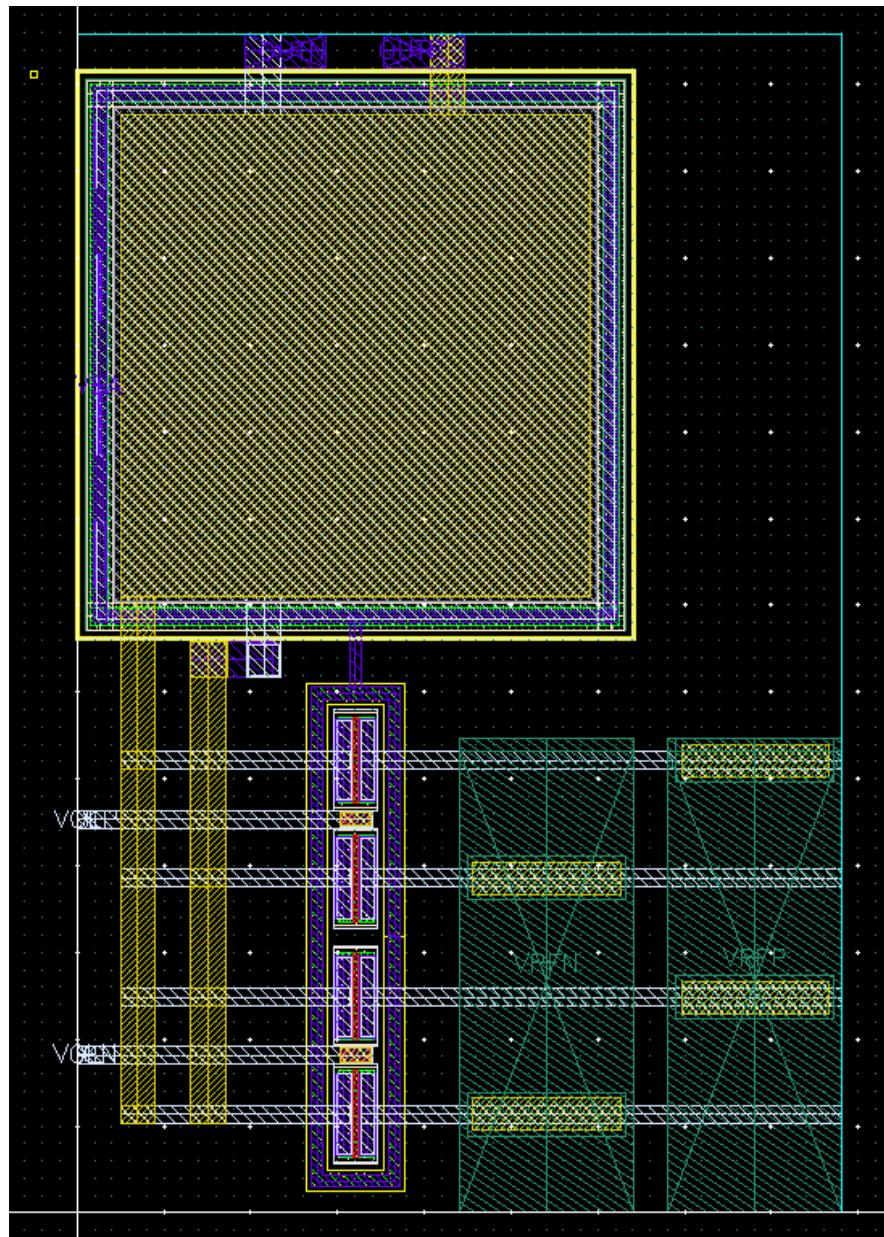
Testbench du mixer seul

Avec les paramètres déterminés dans l'étude qui précède, on obtient les spécifications suivantes pour le mixer passif seul :

Gain	IIP3	NF
0 dB	10 dBm	6

Performances schématiques du mixer

Layout



Layout du mixer passif

Le layout de ce mixer est très simple, et a été orienté afin de simplifier l'intégration avec le récepteur. Sa petite taille ne rend pas les simulations PEX très pertinentes, mais elles ont tout de même été passées, et une variation minime de ces dernières a été observée :

Gain	IIP3	NF
-0.6 dB	10 dBm	6.3 dB

Performances PEX du mixer

Le mixer passif est désormais prêt à être intégré avec le récepteur.

2.3.2 Mixer Actif (Pierre-Olivier Guessard)

Pour la chaîne TX, nous avons conçu un Mixer Actif qui sera composé de deux Balun Actif suivi de deux cellules de Gilbert, chacune pour les voies I et Q. Il se situe avant le PA et après l'ADC.

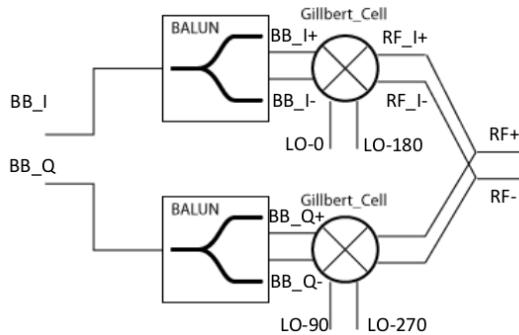


Schéma Bloc du Mixer Actif

En entrée, les deux signaux IQ auront une amplitude de 100 mV. Les signaux OL sont issues du VCO de la PLL à 2.45 GHz déphasées de 180°.

A) Balun Actif (Yann Zyzelewicz)

Dimensionnement et simulation

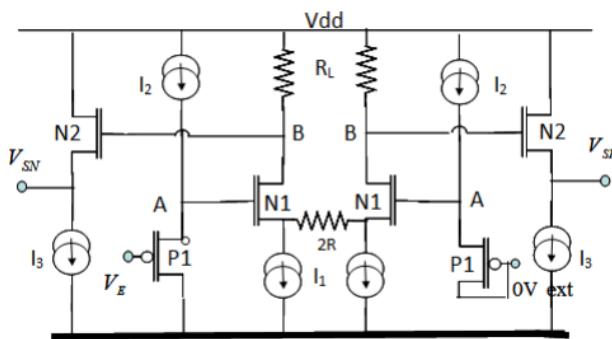
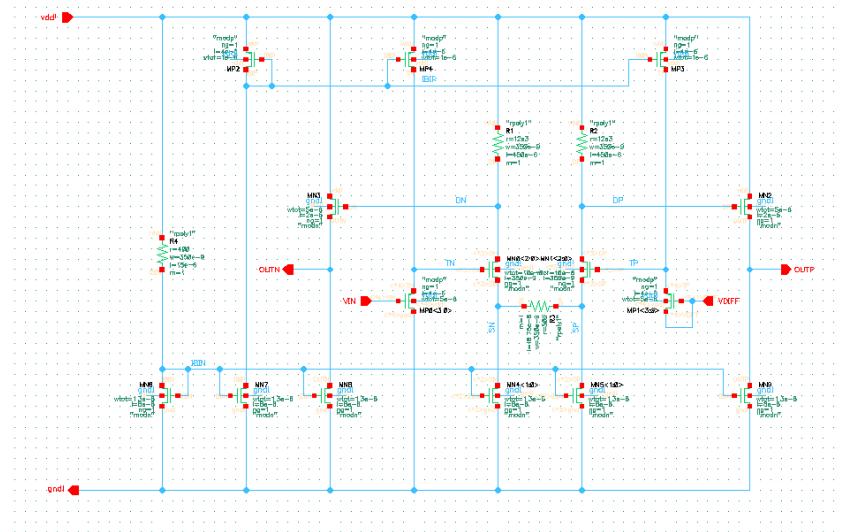


Schéma Bloc du Mixer Actif

Le balun Actif comporte un translateur PMOS en entrée, un convertisseur de mode commun vers le différentiel et un translateur NMOS en sortie. Ces différents blocs permettent d'obtenir la meilleure optimisation possible entre la translation des modes communs désirées et le gain.

Pour le balun actif, après avoir suivi le dimensionnement donnée dans le document du projet, nous obtenons le schematic suivant :

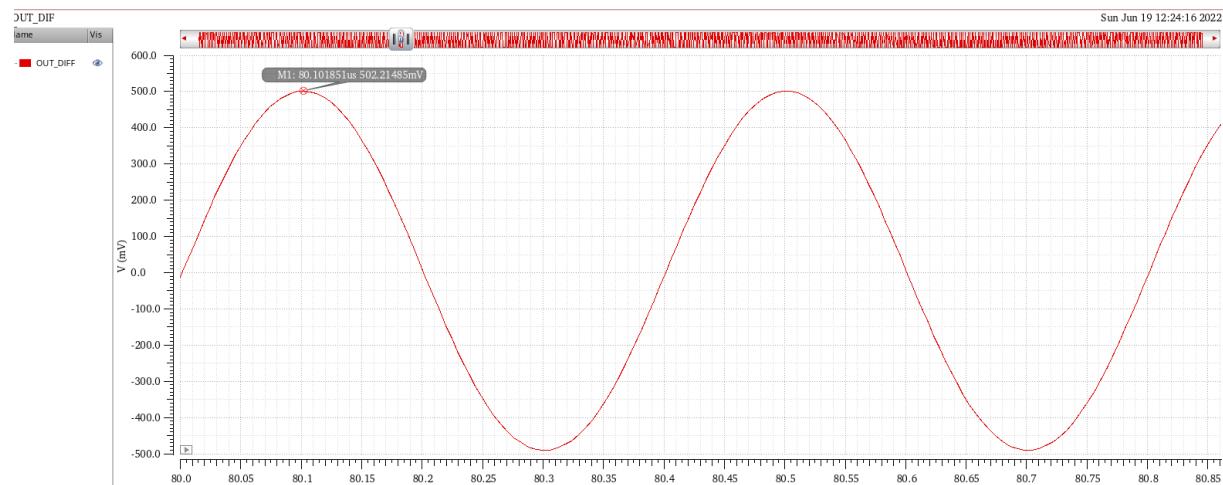


	N1	N2	P1
W(μm)	10	5	5
L(μm)	0,35	2	4

RL(ohms)	12K
2R(ohms)	1K

Schematic Balun Actif et dimensionnement

Nous envoyons donc 100 mV d'amplitude en entrée du balun actif, puis nous observons sa sortie différentiel.



Youtdiff Balun Actif

Nous observons un signal d'amplitude de 500 mV, ce qui est au-dessus de ce que nous souhaitions (400 mV).

B) Cellule de Gilbert (Pierre-Olivier Guessard)

Dimensionnement et Simulation Schematic

Pour la cellule de Gilbert, nous aurons en entrée le signal différentiel du balun à une fréquence de 2.5MHz et d'amplitude environ 100mV.

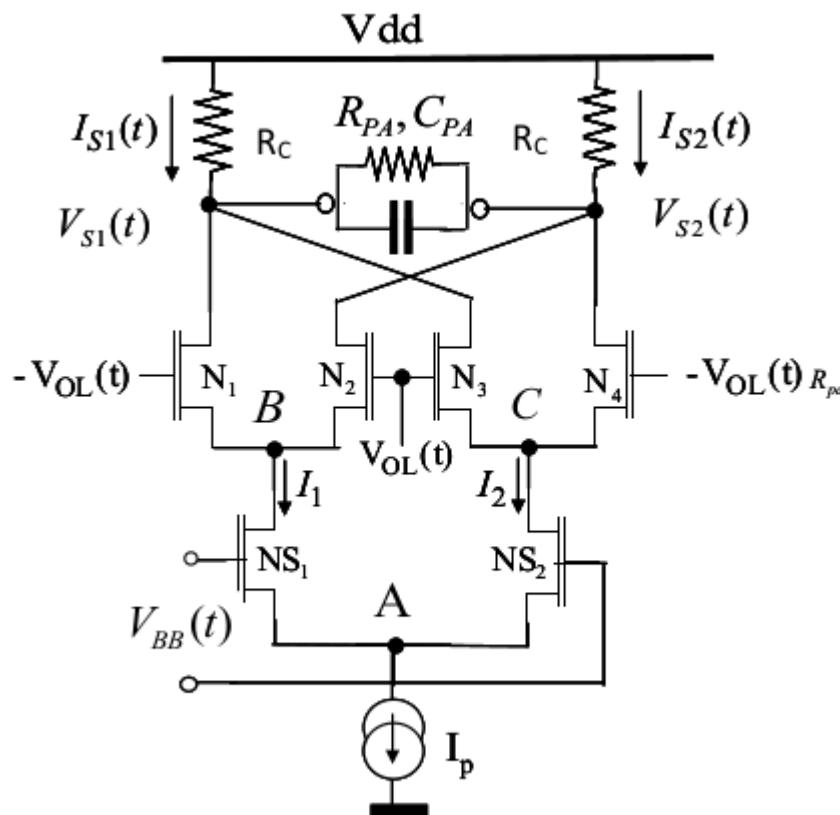


Schéma Cellule de Gilbert

La cellule de Gilbert se compose d'une paire différentiel (NS1 et NS2) recevant le signal du balun. La cellule fonctionne à l'aide des commutations des deux paires différentielles au-dessus, au rythme des signaux fournis par la PLL : quand N1 et N4 conduisent, N2 et N3 sont bloqués et vice-versa.

1e étape : Calcul du courant I_p en fonction de la charge en sortie et l'amplitude V_a du signal différentiel $V_s(t)$

Si I_p est alternativement commuté entièrement sur une des sorties, nous avons :

$$I_p = \alpha * \frac{4V_a}{R_{pa}} \text{ avec } \alpha = \sqrt{1 + \left(\frac{R_{pa} * C_{pa}}{2} * 2\pi f_{ol}\right)^2} = 5.55 \text{ mA}$$

2e étape : Détermination des tensions de mode commun à l'équilibre

Nous souhaitons déterminer les tensions de mode commun à l'équilibre du signal du balun V_{BBMC} et de l'OL V_{OLMC} .

Afin que NS1 et NS2 soient en région active, on fixe $V_{BBMC} = \frac{VDD}{2.5} = 1.33V$ et $V_{OLMC} = \frac{2}{3} * VDD = 2.2V$.

3e étape : Pré-calcul du W de $NS_{1,2}$

Nous déterminons le W de $NS_{1,2}$ en se basant sur les commutations qu'il va y avoir entre ces deux MOS, et donc que l'un des deux soit complètement bloqué et l'autre saturé.

Nous avons l'équation suivante : $(\frac{W}{L})_{NS1,2} = \frac{I_p}{KnV_{bb}^2} = 39.8$ avec $Kn = 80 \mu A/V^2$

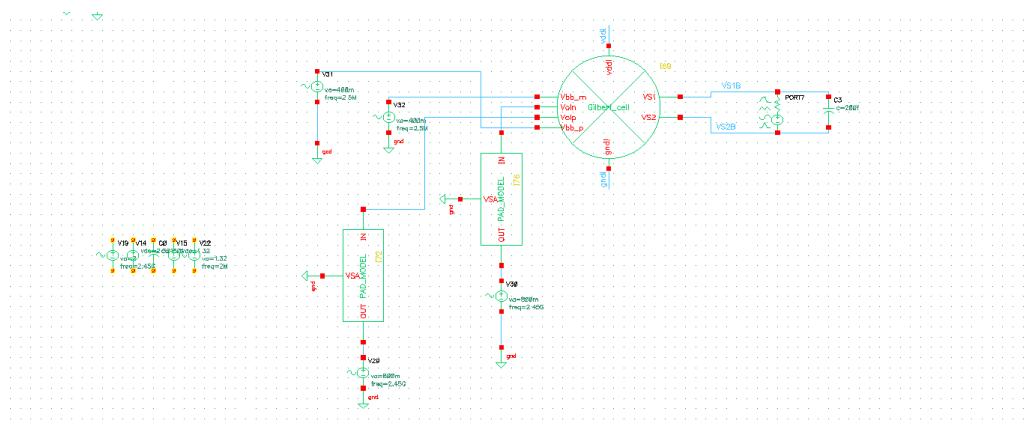
Nous savons que $L=0.35 \mu m$, donc $W_{NS1,2} = 80 \mu m$.

4e étape : Détermination du W de $N_{1,2,3,4}$

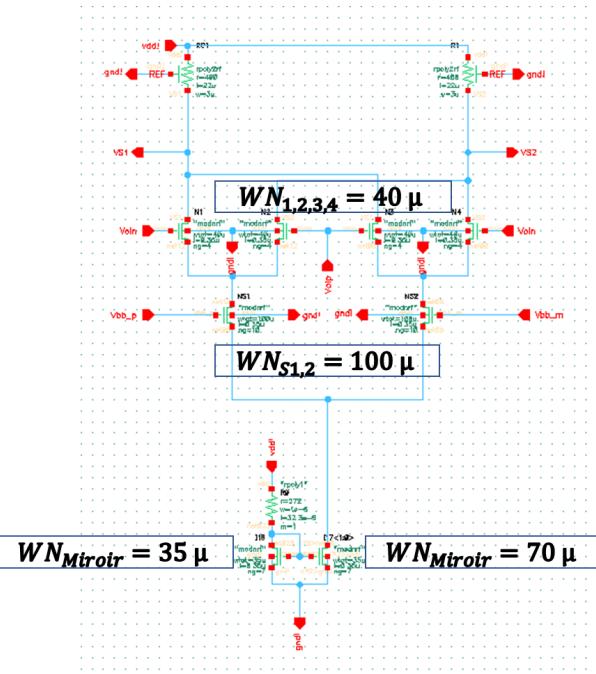
Nous calculons ce W en se servant de la condition de limite de Zone Active des MOS $NS_{1,2}$ qui est $Vds_{min} > (Vgs_{max} - V_T)$.

Nous obtenons la relation suivante en utilisant cette condition et les valeurs de mode-commun définies à l'étape 2 : $(\frac{W}{L})_{N1234} = \frac{I_p}{Kn(0.26*VDD + \frac{V_{OL}}{2} - \frac{V_{BB}}{2} - 0.7)^2} = 60 \mu m$.

Après ce "pré-dimensionnement" théorique, nous avons créé le schematic de la cellule de Gilbert sous cadence. Suite à cela, nous créons un bench afin de tester les différentes caractéristiques de la cellule de gilbert et vérifier sa tension de sortie.



Bench Cellule de Gilbert

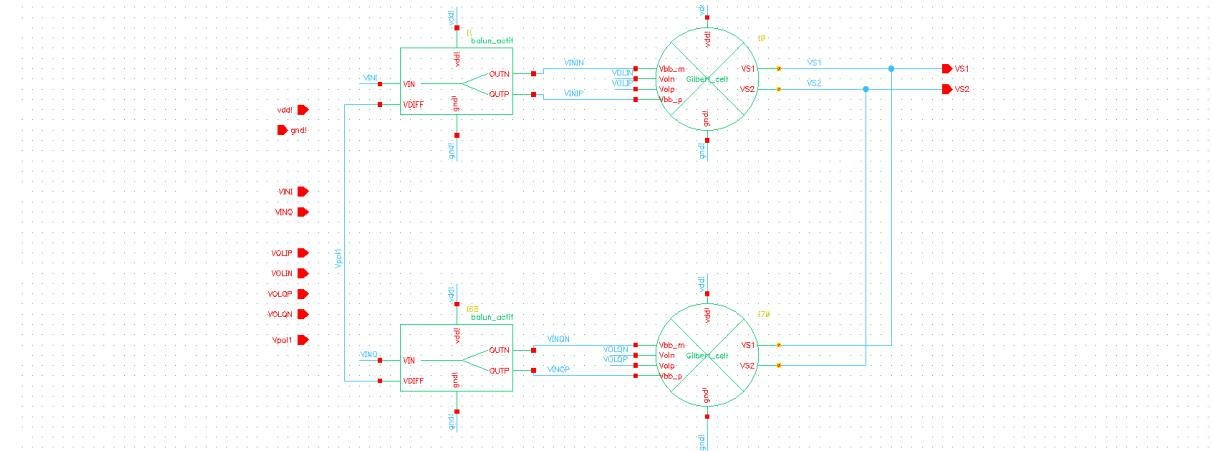


Dimensionnement finale Cellule de Gilbert

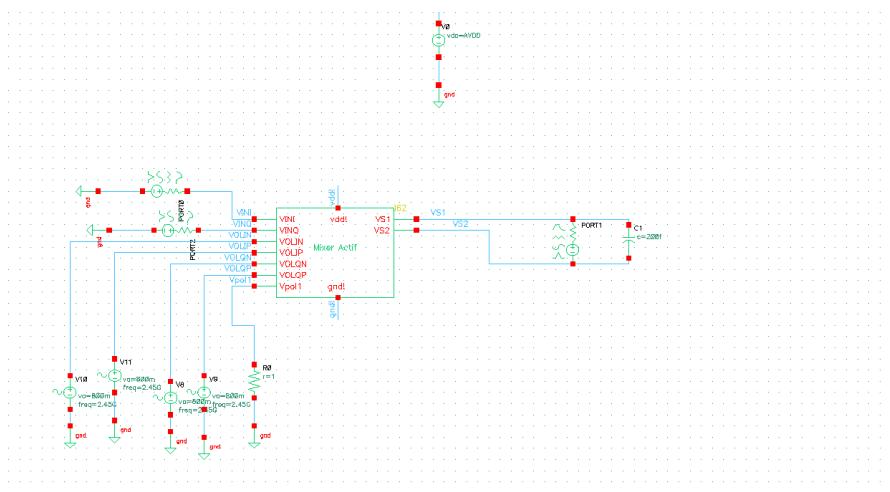
Lors des différentes simulations effectuées, nous avons remarqué que la cellule de Gilbert ne fournissait pas une amplitude différentielle assez élevée en tension de sortie, nous avons donc augmenté la taille des transistors de notre paire différentielle qui reçoivent les signaux en sortie du Balun et diminué celle des deux étages au-dessus.

Après plusieurs tests et simulation, nous avons mis en commun le balun actif avec la cellule de Gilbert afin de créer le Mixer Actif. Nous avons donc obtenu la cellule suivante puis nous l'avons testé en simulation afin d'obtenir la puissance émise.

C) Mixer Actif

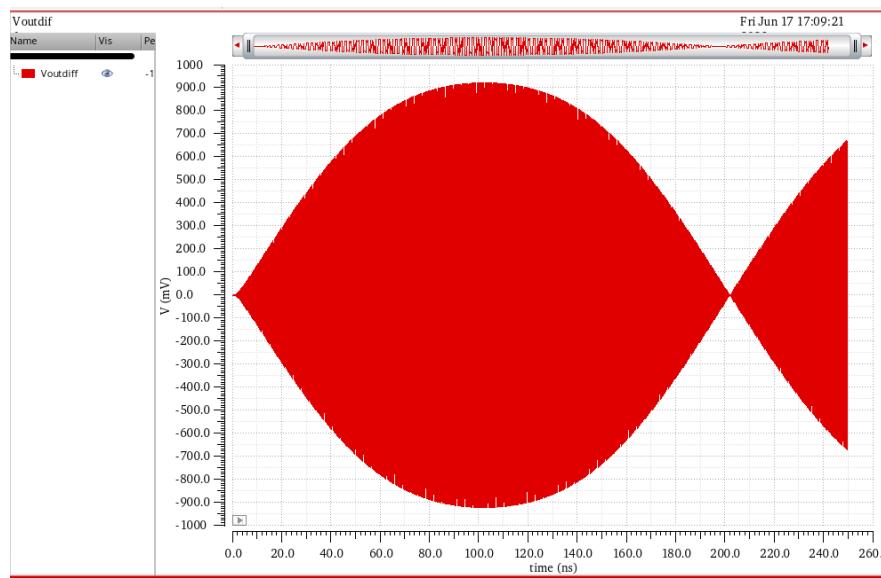


Schematic Mixer Actif (Balun + Celulle de Gilbert)



Bench Mixer Actif

Nous avons pris comme impédance d'entrée du PA 800 ohm pour la partie réel et 200 fF pour la partie imaginaire. Nous obtenons la tension de sortie suivante avec en entrée des baluns une tension de 100 mV à 2MHz pour les signaux déphasées I et Q :

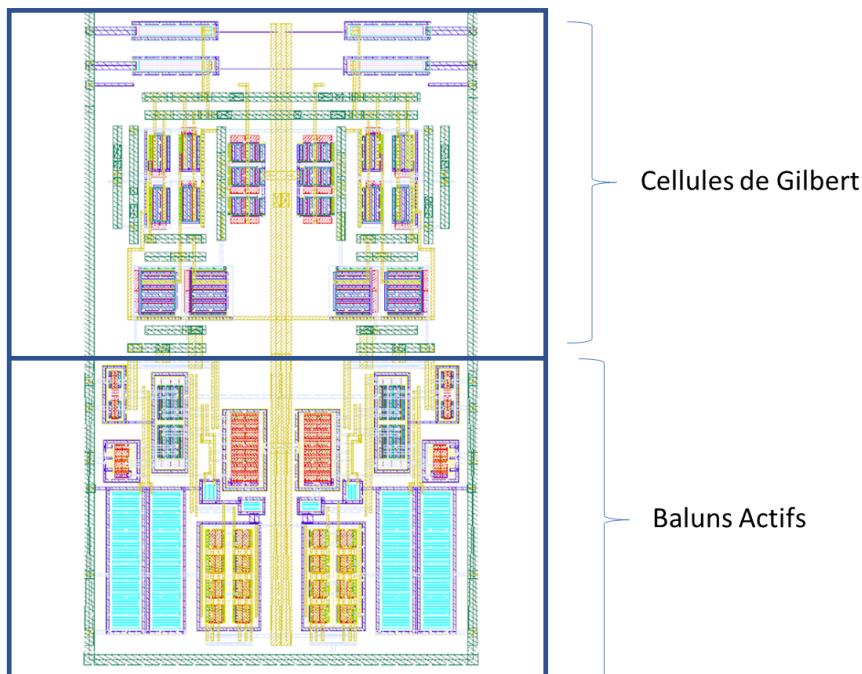


Voutdiff Mixer en fonction du temps

Nous obtenons donc un signal différentiel d'amplitude 915 mV différentiel en sortie du mixer, ce qui respecte le cahier des charges (nous visions 700 mV en sortie).

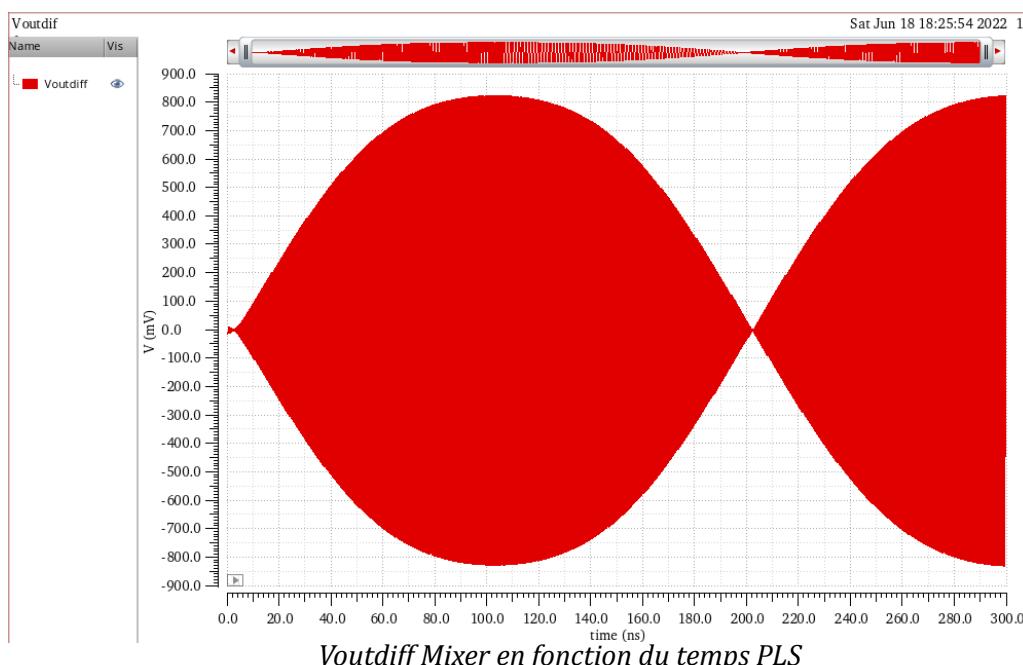
Layout :

Pour le layout du Mixer, vu qu'il est composé des deux voies I et Q, nous avons essayé de l'assemblée de la façon la plus symétrique et compacte possible, afin d'avoir le moins de pertes sur les tensions de sorties.



Layout Mixer Actif

En simulation Post Layout, nous obtenons les résultats suivants :

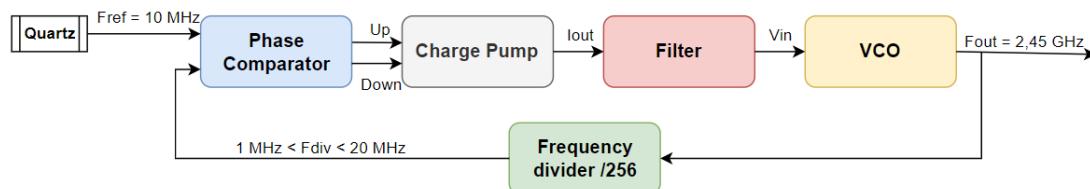


Mixer Actif	Schematic	PLS
Vout	915 mV	810 mV

Nous remarquons qu'en simulation Post Layout, nous perdons 100 mV sur l'amplitude de la tension de sortie par rapport aux résultats sans extraction. L'effet résistif dû aux routages y joue. Cela reste cependant acceptable.

1.4 PLL (Guillaume Monti - Pierre-Louis Hellier - Pierre Bordère)

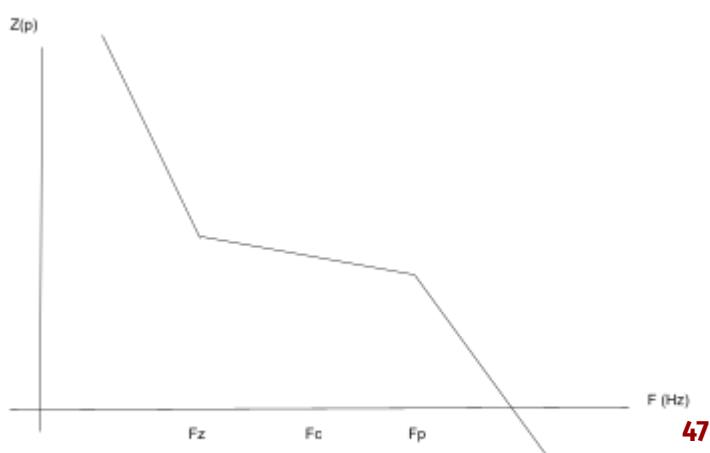
La PLL, ou Phase-Locked Loop, est le module qui sert d'oscillateur local : elle doit être capable de générer une fréquence porteuse de 2.45GHz pour les mixeurs en réception et en émission. Le module est composé d'un ensemble de sous-blocs, parmi lesquels se trouvent un Phase-Frequency Detector (PFD), une Charge Pump (CP), un filtre de boucle, un Voltage Controlled Oscillator (VCO) et un Diviseur de Fréquence placé sur le loopback. Le schéma ci-dessous représente la PLL sous forme d'architecture bloc.



- **Phase-Frequency Comparator** : en comparant la fréquence de référence Fref, issue du quartz, à la fréquence du loopback issue du VCO et divisée par le facteur N du diviseur de fréquence, deux signaux up et down sont générés pour être utilisés par la CP en aval (fait partie du PFD)
- **CP** : permet de produire un courant positif ou négatif en fonction de l'état de up et down. Le courant donc généré en sortie possède une composante continue directement proportionnelle à $V_{PD_b} = K_{PD} * (\Phi_{REF} - \Phi_{DIV})$ (fait partie du PFD)
- **Filtre de boucle** : son rôle est d'extraire la composante continue de la sortie de CP
- **VCO** : permet de générer, à partir du signal de CP filtré, la fréquence de la porteuse de 2.45GHz qui servira au mixer et qui sera utilisée en loopback par le diviseur de fréquence
- **Diviseur de fréquence** : ramener la fréquence de 2.45GHz à une fréquence d'environ 10MHz pour le PFD et avoir le même ordre de grandeur que FREF

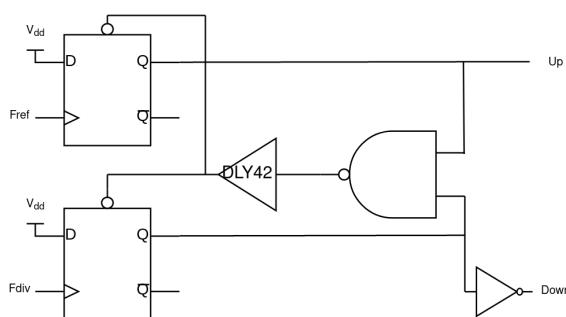
Principe de fonctionnement de la PLL		
Cas où Fref < Fdiv	Cas où Fref = Fdiv (PLL locked)	Cas où Fref > Fdiv
$(d\Phi_{REF} - \Phi_{DIV})/dt < 0$ et $V_{PD_b} = K_{PD} * (\Phi_{REF} - \Phi_{DIV})$ diminue	$(d\Phi_{REF} - \Phi_{DIV})/dt = 0$ et $V_{PD_b} = K_{PD} * (\Phi_{REF} - \Phi_{DIV})$ constant	$(d\Phi_{REF} - \Phi_{DIV})/dt > 0$ et $V_{PD_b} = K_{PD} * (\Phi_{REF} - \Phi_{DIV})$ augmente
Vvco décroît (gain négatif) donc FOUT augmente	Vvco reste stable donc FOUT reste stable => $FOUT = N.FREF$	Vvco augmente (gain négatif) donc FOUT diminue

Fo (Hz)	2,45E+09
Fref (Hz)	1,00E+07
Iref (A)	1,00E-05
PhiM(Deg)	4,50E+01
PhiM(rds)	7,85E-01
Vdd (V)	3,30E+00
Kvco (rds-1)	7,50E+08
Kd	1,59E-06
N	255
wc (rad/s)	3,14E+06
wp (rad/s)	7,58E+06
wz (rad/s)	1,30E+06
C1 (F)	1,96E-13
C2 (F)	9,49E-13
R2 (Ohm)	810 129,32



En utilisant le tableur de la page précédente, nous avons d'abord choisi un courant de fonctionnement de la pompe de charge de $10\mu A$ pour le NMOS (et $-10\mu A$ pour le PMOS), de sorte à trouver un bon compromis entre faible consommation, valeurs et taille des composants du filtre, de manière à ce qu'ils soient intégrables. Concernant la fréquence de coupure F_c du filtre, sa valeur a été sélectionnée de telle sorte que $F_c = F_{ref}/20$, soit 500 kHz, afin d'avoir une bonne stabilité du système grâce à une marge de phase d'environ 90° et un gain du filtre d'environ -1.82 dB à la fréquence de fonctionnement attendue de 10 MHz. Les valeurs exactes de gain et marge de phase se trouvent plus tard dans le rapport.

Phase-Frequency Comparator



Tout d'abord, le PFD est composé de deux sous-blocs : le phase-frequency comparator (PFC) et la pompe de charge (les layouts de tous les blocs se trouvent à la fin du chapitre PLL).

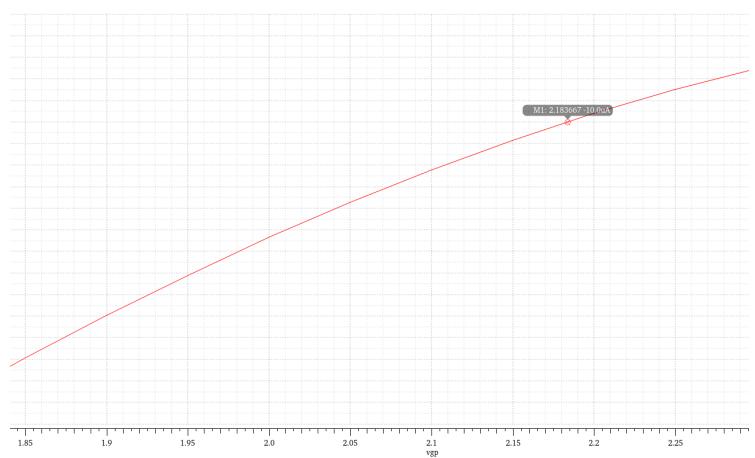
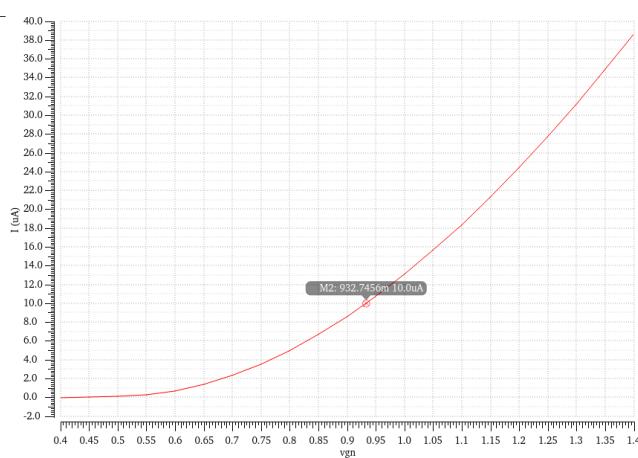
Pour ce qui est du premier bloc, le PFC, nous avons choisi une architecture de ce type pour 2 raisons principales : premièrement, on peut réaliser une détection de phase et de fréquence, ce qui permet un verrouillage de la PLL même si F_{REF} et F_{DIV} sont initialement éloignés. Ensuite, après verrouillage, la PLL est stabilisée en fréquence et donc les entrées du comparateur ont la même phase et fréquence. L'utilisation d'une cellule de retard de type DLY42 s'explique par le fait qu'il est nécessaire de reset les bascules de façon décalée et ainsi leur permettre une commutation "normale".

Pour ce qui est des bascules, elles sont extraites de la librairie digitale (CORELIB), étant donné que la fréquence de fonctionnement de ce bloc est de l'ordre de grandeur de la dizaine de MHz.

Cas où $F_{ref} < F_{div}$	Cas où $F_{ref} = F_{div}$ (PLL locked)	Cas où $F_{ref} > F_{div}$
<p>États logiques $UP = 0$ & $DOWN = 1$</p>	<p>États logiques $UP = 0$ & $DOWN = 0$</p>	<p>États logiques $UP = 1$ & $DOWN = 0$</p>

Charge Pump

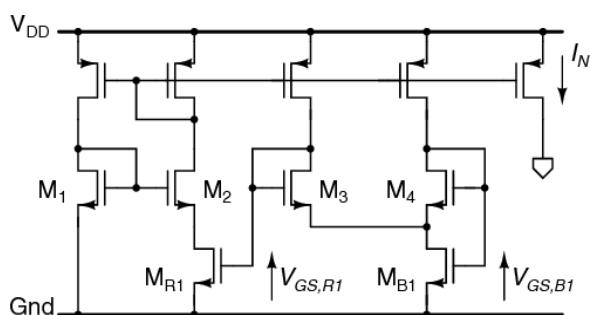
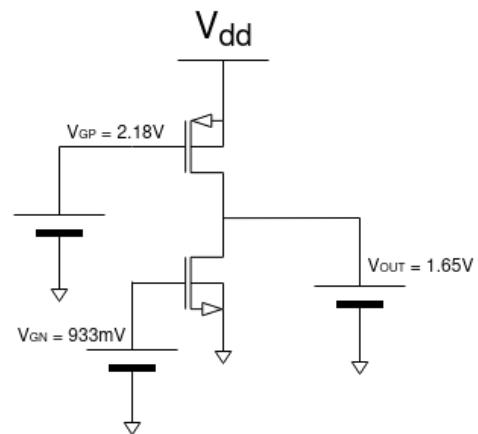
Passons maintenant à la pompe de charge, deuxième bloc du PFD. Afin de déterminer les paramètres W et L des MOSFET MN0 et MP0 faisant office de sources de courant, nous avons réalisé des simulations paramétriques.. Pour ces deux transistors, en ayant une tension V_{DS} fixée à 1.65V (pour être au milieu de dynamique d'entrée du VCO), on a fait varier la tension de grille de sorte à trouver le courant attendu de 10µA (et -10µA pour le PMOS) comme décrit précédemment. Sur les graphiques ci-dessous, pour le NMOS (à gauche), on trouve V_G = 933mV et pour le PMOS (à droite) un V_G égal à 2.18V.

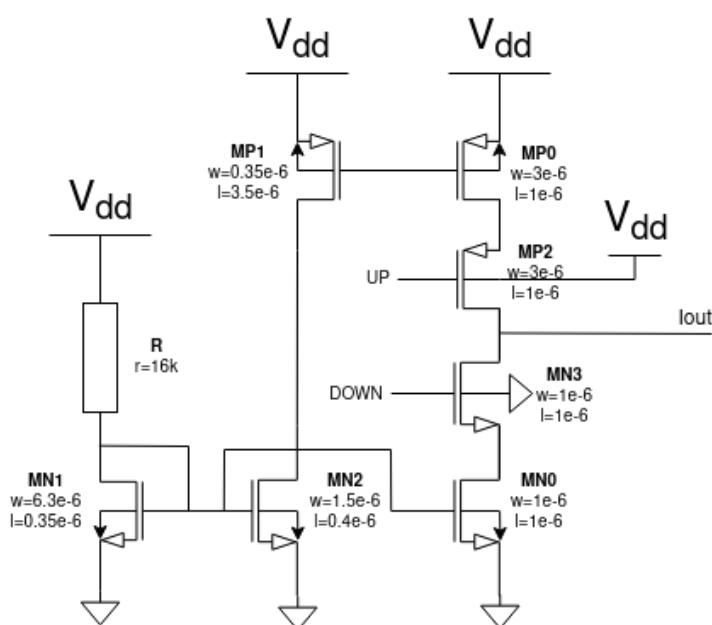


En connectant les deux MOS comme dans le schéma ci-contre, on obtient un courant nul en sortie. Pour fonctionner avec le PFC et former le PFD, on a donc ajouté des transistors faisant office d'interrupteurs pour que :

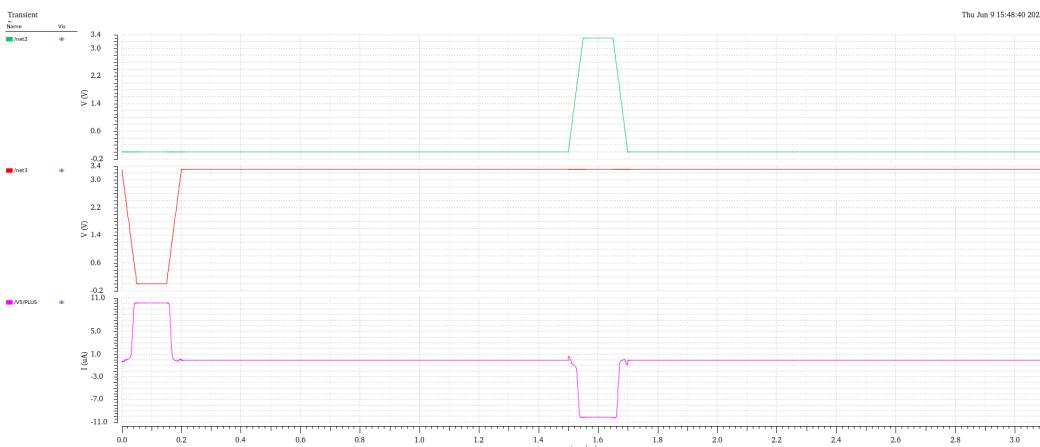
- quand UP=1 et DOWN=1 (logique inversée dû à l'inverseur sur DOWN dans le bloc PFC), on obtient un courant de -10µA
- quand UP=0 et DOWN=0, on obtient un courant de 10µA
- dans les états restants, on obtient un courant nul

Afin de générer les tensions de polarisation des deux sources de courants, et avoir un courant identique dans les branches NMOS et PMOS de $\pm 10\mu\text{A}$, on a utilisé deux miroirs de courants "interdépendants". Utiliser des miroirs de courants présente en effet ces avantages, néanmoins une résistance est nécessaire pour générer le courant du miroir NMOS : cela prend donc beaucoup de place en layout, génère des pertes d'énergie et dépend fortement de la température. Par manque de temps, nous avons donc utilisé le circuit avec résistance, mais on aurait pu utiliser un "current source reference design circuit" qui généralement est compensé en température (PTAT) et sans résistance (c.f. ci-dessous - Source : Semantic Scholar).





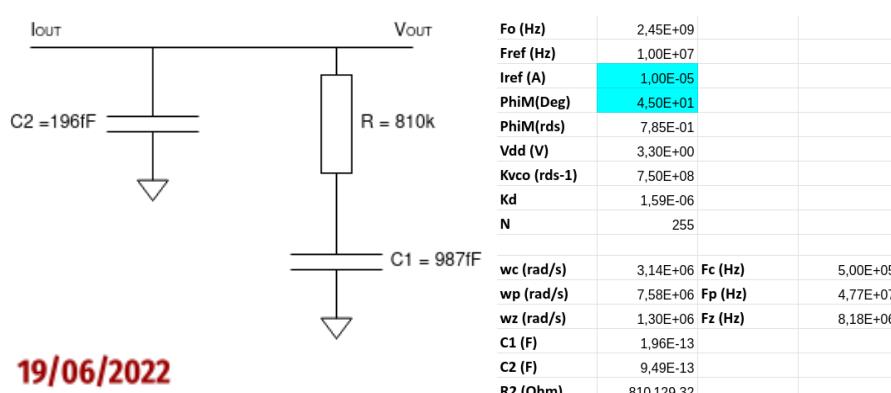
Voici donc notre schéma final de la pompe de charge. Pour générer les $\pm 10\mu\text{A}$ on a d'abord dimensionné MN1, à l'aide de simulations paramétriques et des annotations de tension sur le schematic, de sorte à avoir une tension de 937mV arrivant sur la grille de MN0. De même pour polariser MP0, en jouant sur la taille de MN2 et MP1, on a obtenu la tension désirée de 2.18V. Toujours en fixant une tension Vout de 1.65V (milieu de dynamique), on peut bien voir dans le graphique que quand UP=1 et DOWN=0, on obtient un courant de $-10\mu\text{A}$, et lorsque UP=0 et DOWN=1, on obtient un courant de $10\mu\text{A}$. Le fonctionnement du bloc est donc validé.



Chronogramme de I_{out} (violet) en fonction de UP (vert) & DOWN (rouge)

Filtre

Comme énoncé précédemment, l'ordre de grandeur du courant dans la charge pump influençant directement la taille des composants du filtre, on s'est donc orienté vers un courant de $10\mu\text{A}$ et donc à l'aide d'une feuille de calcul, on a pu déterminer les valeurs des composants du filtre et donc leur taille. On s'est orientés vers une architecture de type CRC, afin d'avoir une bonne stabilité du système mais aussi avoir 3 pôles permet de réaliser une bonne réjection de FREF à l'entrée du VCO et donc un meilleur filtrage. On a donc trouvé :

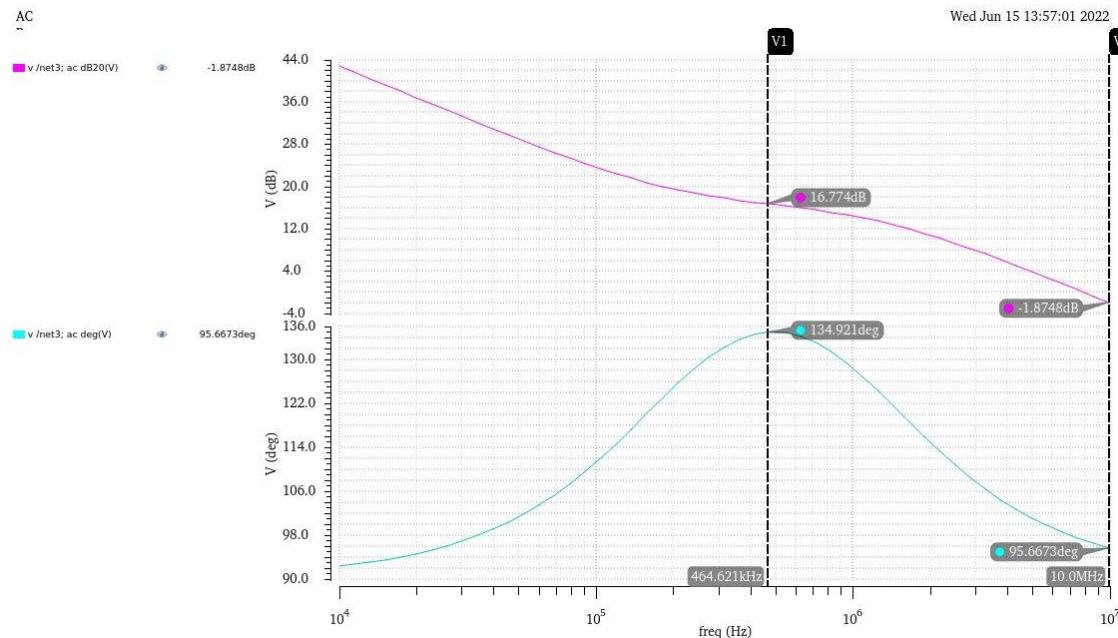


$$C_1 = \frac{wz}{wp} \cdot \frac{K_0}{wc^2 \cdot N} \cdot \frac{I_{REF}}{2\pi} \cdot \sqrt{\frac{1 + \frac{wc}{wz}}{1 + \frac{wc}{wp}}}$$

$$C_2 = C_1 \left(\frac{wp}{wz} - 1 \right)$$

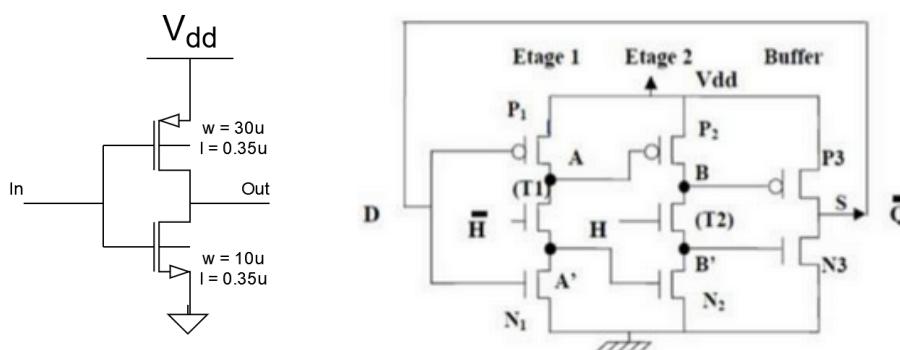
$$R_2 = \frac{1}{C_2 \cdot wz}$$

Par conséquent, afin de trouver un bon compromis entre réjection de FREF en entrée du VCO, taille des composants, marge de phase et gain du filtre, on s'est de nouveau servi du tableau précédent. Nous sommes partis sur une fréquence de coupure égale à $F_c = F_{REF}/20$. A 10MHz, on obtient une marge de phase de 85° et un gain de -1.8748dB (c.f. graphique ci-dessous), ce qui a suffit pour permettre au système de fonctionner. On aurait pu diminuer la fréquence de coupure à $F_c = F_{REF}/30$ ou $F_{REF}/40$ afin d'améliorer la réjection de FREF et ainsi réduire le jitter sur la sortie du VCO, mais la taille des composants aurait augmenté et la stabilité aurait diminué. Le graphique ci-dessous montre l'évolution de la phase et de la fréquence en fonction de la fréquence pour un circuit dimensionné avec $F_c \sim 500\text{kHz}$.

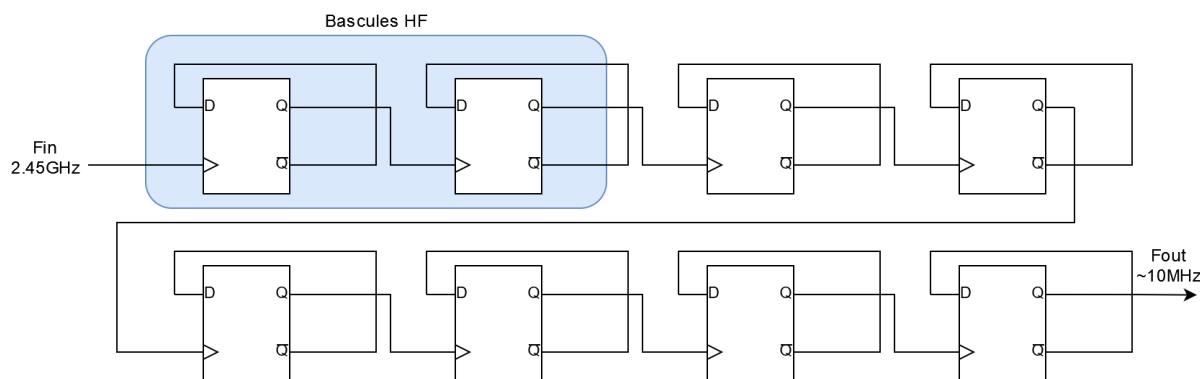


Frequency Divider /256

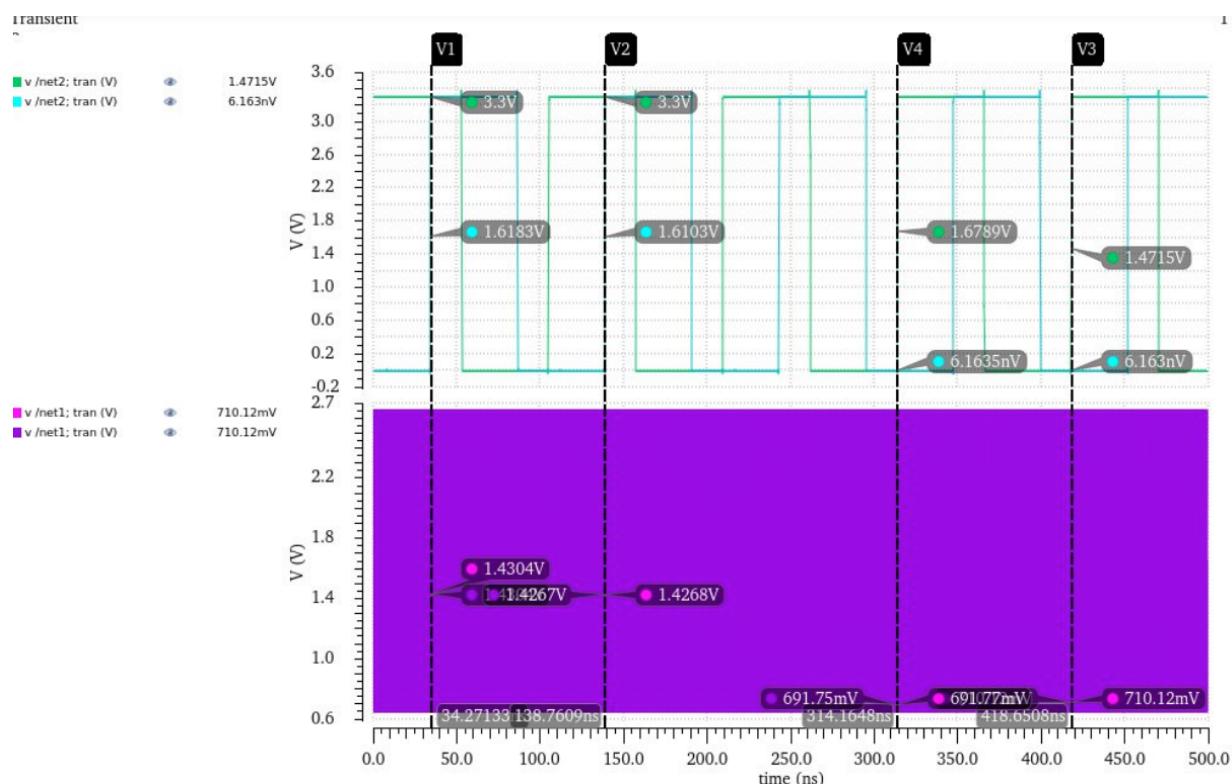
Le diviseur de fréquence placé sur la boucle de contre-réaction permettait de passer d'une fréquence d'environ 2.45GHz à une fréquence d'une dizaine de MHz, ce qui est dans le même ordre de grandeur que la fréquence de référence du quartz. Pour réaliser cette division, on s'est d'abord servi de bascules D, qui ont pour objectif de diviser la fréquence en entrée par /2. Or des bascules dites "numériques" ne peuvent fonctionner à des fréquences supérieures à 650MHz. On a donc dimensionné des bascules dynamiques /2 (schéma à droite) afin de palier ce problème. Mais pour les faire fonctionner, il fallait inverser le signal "d'horloge" entrant. Donc on a donc dimensionné un inverseur HF (schéma à gauche).



Le schematic est donc le suivant :



En cascasant deux bascules dynamiques et six autres bascules D numériques, on a pu arriver à diviser 2.45GHz vers \sim 10MHz. Dans ce graphique, on peut distinguer les différences entre simulation schématique et post extract. En violet, il s'agit du signal d'entrée à 2.45GHz et en vert le signal de sortie à 10MHz.



Voltage Controlled Oscillator

1) Dimensionnement du résonateur

Nous avons choisi une inductance différentielle DI60PT de 6nH. Le choix de cette valeur d'inductance a été déterminé à partir d'un compromis entre taille du condensateur et facteur de qualité de l'inductance.

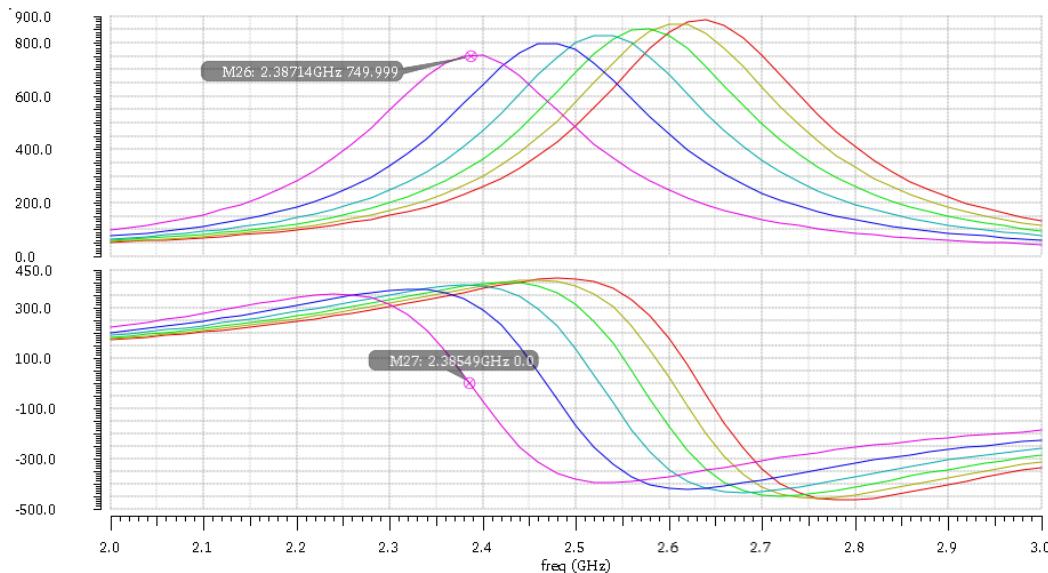
On a utilisé deux diodes varicap afin de pouvoir modifier la fréquence de résonance de la cellule LC. Les diodes varicap avaient une valeur de capacité allant de 200 fF à 400 fF en fonction de leur tension de polarisation. Pour osciller à 2,45 GHz avec une inductance de 6nH, on a calculé la valeur de la capacité avec la formule suivante : $C = 1/(w_0^2 * 6nH) = 700fF$

Il fallait donc ajouter une capacité C0 de valeur fixe en parallèle des diodes pour atteindre 700 fF. On a pris dans un premier temps $C_0 = 400 fF$ que l'on a ajusté afin d'osciller à 2,45GHz.

2) Dimensionnement de la paire croisée

La paire croisée est un amplificateur qui compense les pertes du résonateur afin d'entretenir l'oscillation. Nous avons d'abord simulé le résonateur complet pour mesurer l'impédance du résonateur en fonction de la fréquence.

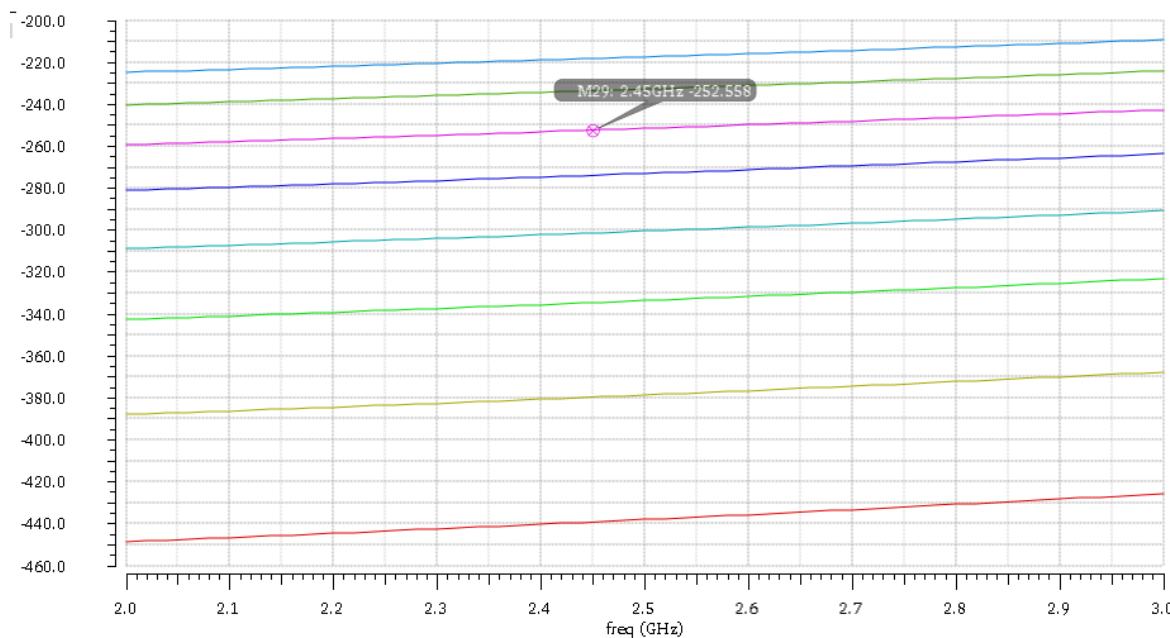
On a ensuite réalisé une simulation paramétrique sur la tension de polarisation des diodes :



Simulation impédance du résonateur

A la fréquence de résonance ($\text{imag}(Z_{11}) = 0$), il fallait dimensionner la paire croisée pour compenser la partie réelle. Celle-ci descendait jusqu'à 750Ω dans le pire cas (résistance parallèle). Nous avons pris un facteur 3 afin de compenser une résistance parallèle de $750/3 = 250\Omega$. Le gm correspondant est 8mS.

On a dans un premier utilisé une source de courant parfaite de 5mA, en faisant une simulation paramétrique, afin de trouver la largeur W de transistor nécessaire pour atteindre ce gm :

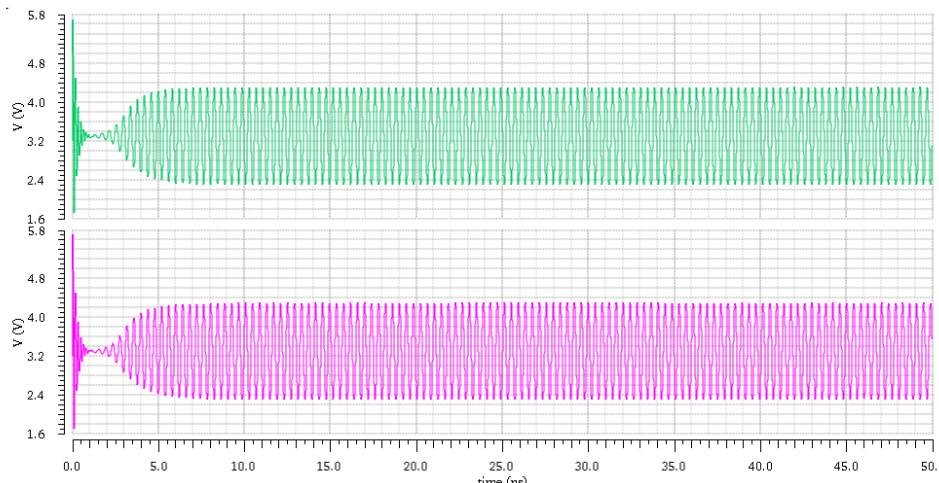


Simulation paramétrique de la résistance négative de la paire croisée

On a déterminé un $W = 50 \text{ um}$ pour avoir une résistance négative de -250Ω .

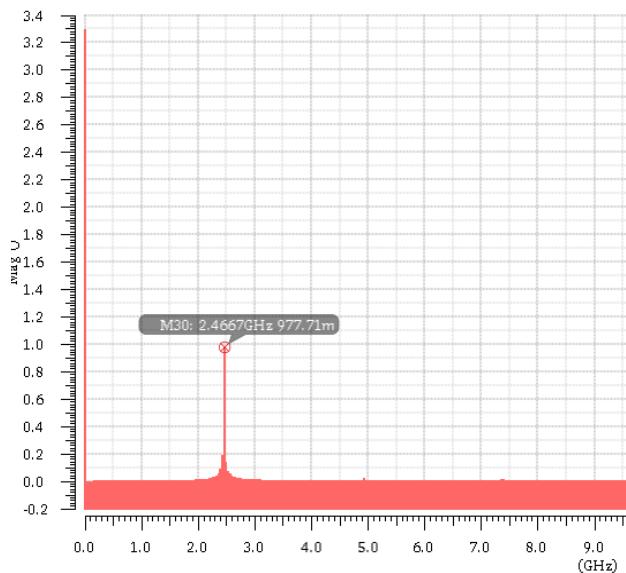
3) Oscillateur

On a assemblé le résonateur et la paire croisée. On a fixé une tension $V_{pol} = 1,65V$ (milieu de dynamique).



Simulation temporelle de l'oscillateur

On a paramétré une densité de bruit dans la bande de fréquence afin que l'oscillation puisse amorcer.

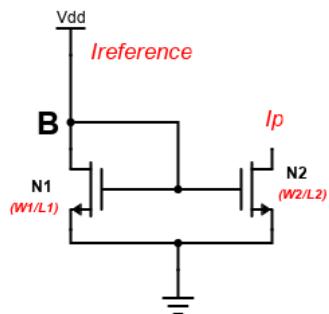


Analyse spectrale du signal

Le signal généré oscillait à 2,47 GHz : cette fréquence était satisfaisante.

4) Miroir de courant

Il fallait ensuite dimensionner la source de courant de la paire croisée.

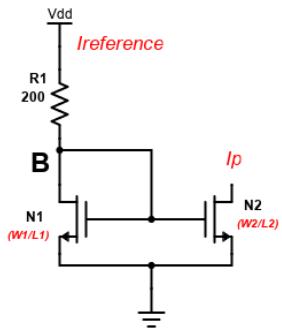


Nous souhaitons polariser notre VCO avec un courant de polarisation $I_p = 4\text{mA}$.

Ce courant va copier une référence $I_{ref} = K * \frac{w_1}{L_1} * (V_B - V_t)^2 = 4\text{mA}$.

par le calcul nous trouvons $W1=10\text{ }\mu\text{m}$ et $L1=0,39\mu\text{m}$ or pour un soucis de linéarité nous souhaitons un rds du transistor N1 le plus grand possible. rds et L1 étant liés il nous faut L1 le plus grand possible. Par simulation un rapport $W1/L1$ d'environ 25,3 est le plus optimal, avec un rapport de recopie correct ($\approx 2,5\% \text{ d'erreur}$).

Afin d'optimiser la consommation de notre miroir de courant nous allons passer d'un rapport 1:1 avec $I_p = I_{ref}$ à un rapport 1:20 ($I_p = 20 * I_{ref}$). Nous allons pour ce faire diviser W1 par 20. On obtient $W1/20 = 0,5\text{ }\mu\text{m}$.



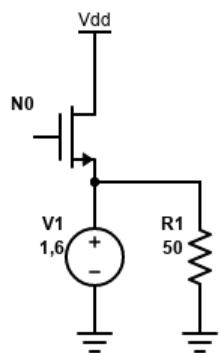
On ajoute ensuite une résistance à notre circuit, on utilise une RpolyBRF de 200Ω . On re-paramètre notre transistor $\rightarrow L1 = 0,56\mu\text{m}$.

5) Dimensionnement des buffer 50Ω

La PLL devait être désignée en stand alone, et la sortie connectée à un analyseur de spectre 50Ω en mode commun. Pour attaquer les 50Ω de l'analyseur, il était nécessaire d'implémenter un étage tampon (buffer).

Le rôle de cet étage était de recopier un signal tout en assurant une adaptation d'impédance en sortie. L'entrée du buffer présentait une haute impédance afin de ne pas perturber le montage amont.

Sur chaque branche, un buffer 50Ω a été placé afin de charger la paire croisée sans déséquilibrer le montage. Par conséquent, un des buffers était connecté sur une sortie de la puce, et l'autre était chargé par une résistance 50Ω .



Le montage de cet étage tampon sera une source suiveuse.

Afin de dimensionner le buffer on fixe $gm = 20\text{mS} = \frac{2Ids}{V_{Dsat}}$. De plus, nous savons que la tension V_g appliquée sur la grille sera de 3,3V. Avec ces conditions nous souhaitons obtenir un $V_{gs} = 1,6\text{V}$ pour être en milieu de dynamique. Il est donc nécessaire de calculer la valeur d' Ids dans ces conditions.

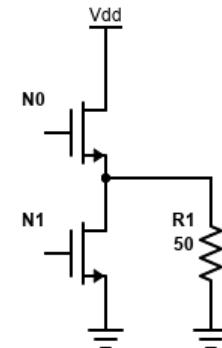
$$gm = 20\text{mS} = \frac{2Ids}{V_{gs}-V_{tn}} = \frac{20\text{mS} * (1,6 - 0,57)}{2} \rightarrow Ids \approx 11,3 \text{ mA}$$

Grâce à une simulation paramétrique nous allons faire varier le W de notre transistor en fixant $V_g=3,3\text{V}$ pour paramétriser notre Ids Réel.

On obtient qu'avec $W = 19$ (*variable que l'on fait varier*) * $5\mu\text{m}$ (*pas de largeur*) $\rightarrow gm = 20\text{mS}$ et $Ids = 11,5\text{mA}$ ce qui est cohérent avec nos calculs théoriques.

Il faut à présent implémenter la source de tension à l'aide d'un transistor MOS (*appelé N1*).

Il doit nous permettre de fixer la tension de $1,6\text{V}$ sur son drain tout en imposant $11,5\text{mA}$ de courant Ids .

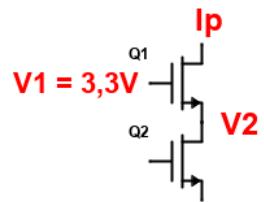


$$Ids_{N1} = Kn * \frac{W_{N1}}{L_{min}} * (1,6 - 0,57)^2 = 11,5mA \rightarrow W_{N1} = 69nm = 14 * 5 \mu\text{m}$$

6) Dimensionnement des translateurs de tension

La tension de mode commun sur la paire croisée est de $3,3\text{V}$. Pour attaquer le diviseur de fréquence (branche de contre-réaction de la PLL), il faut translater le signal RF autour d'une tension DC de $1,65\text{V}$.

Comme pour les buffer 50Ω , nous avons ensuite implémenté un translateur "dummy" afin d'équilibrer la paire croisée.



On souhaite obtenir un $V2=1,6\text{V}$ afin d'être en milieu de dynamique avec $V1=3,3\text{V}$ et $Ip \approx 11,3mA = K_n * W/L * (V1 - V2 - |Vt|)^2$ avec

$$Vt = Vt0 + 0,2V_{sb} = Vt0 + 0,2(1,6 - 0) = 0,89V$$

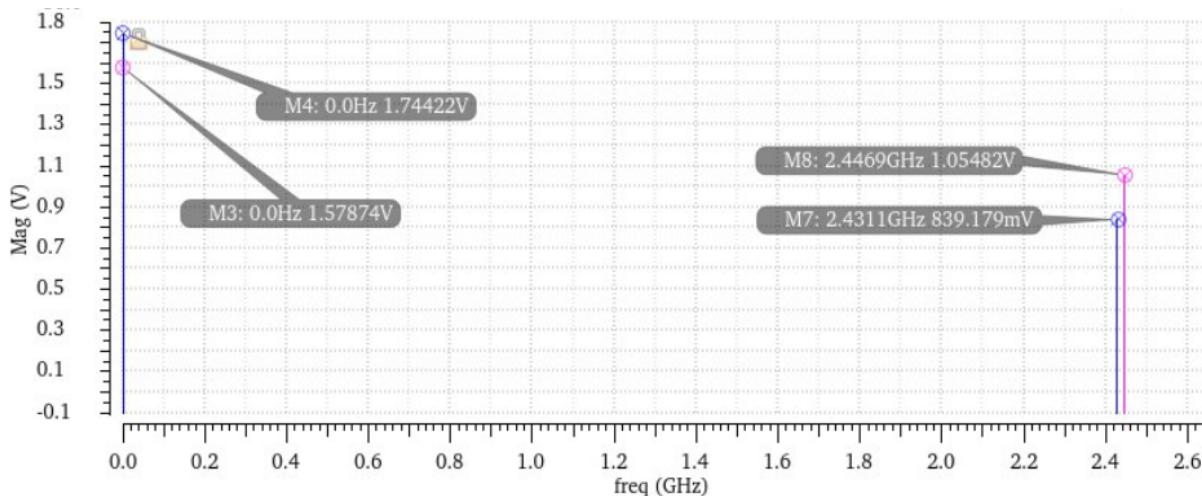
D'après l'équation d' Ip , le seul paramètre que l'on peut modifier est W/L , par le calcul on obtient:

$$\frac{W}{L} = \frac{11,3mA}{K_n * 0,66} = 311.$$

Il nous faut un rapport de $W/L=311$ afin d'obtenir le translateur de tension que nous voulons pour avoir un $V2$ en milieu de dynamique.

7) Caractéristique du VCO : Schématique VS Layout

Une fois le VCO complet, on peut réaliser une simulation “harmonic balance” du VCO. On fixe la tension d’entrée du VCO à 1,65V :



Simulation du VCO, sortie de contre-réaction

Légende :

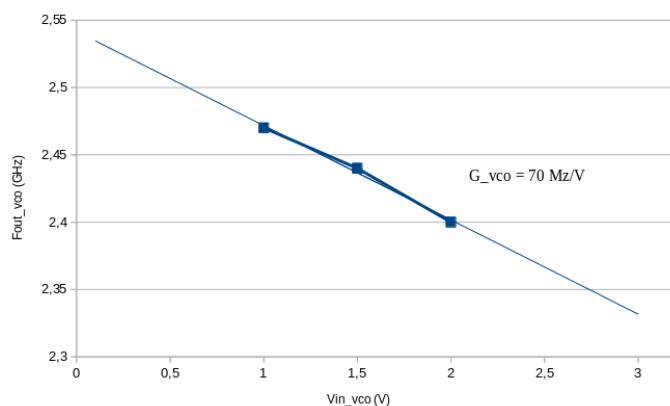
- Rose : schématique
- Bleu : post layout

On observe :

- La fréquence d’oscillation du VCO est plus basse en post-layout. Les capacités parasites augmentent la valeur de la capacité du résonateur et la fréquence de résonance du couple LC diminue en conséquence.
- L’amplitude du signal est plus faible en post layout. Les résistances parasites introduisent des pertes.

La tension de mode commun sur la boucle de feedback diminue également, mais reste proche de la tension de seuil du diviseur de fréquence (1,65V).

En faisant varier la tension Vpol sur la plage de tension [0V ; 3,3V], on peut tracer la caractéristique du VCO :



Caractéristique du VCO

Simulation PLL complète

Il est difficile de simuler la PLL complète car le signal de référence possède une fréquence très inférieure au signal de sortie. Cela demande des simulations avec beaucoup de points et par conséquent beaucoup de temps de calcul.

Pour évaluer efficacement les performances de la PLL (temps d'accroche et bruit de phase), il faudrait utiliser un outil de simulation plus performant comme le verilog.

1) Simulation boucle ouverte

On effectue une simulation temporelle afin d'observer le début du régime transitoire. On souhaite vérifier que la contre-réaction est bien négative.

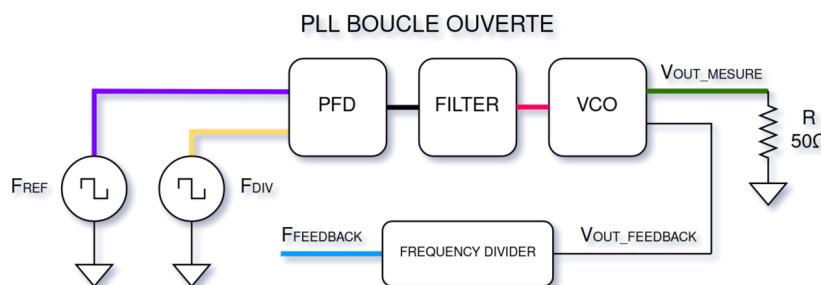
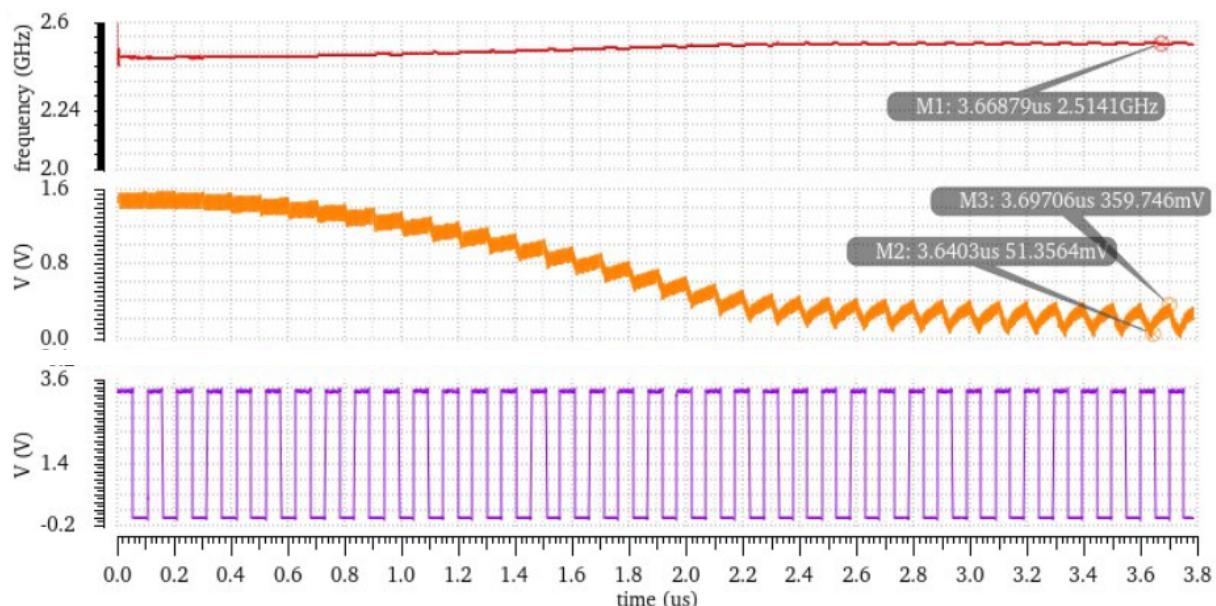


Schéma de simulation en boucle ouverte

On injecte en entrée du comparateur de phase deux fréquences, Fref et Fdiv avec : Fref > Fdiv.

Comme la fréquence de référence Fref est plus élevée que la fréquence Fdiv, le système doit faire en sorte d'augmenter la fréquence de sortie du VCO.

On observe la sortie du diviseur de fréquence :



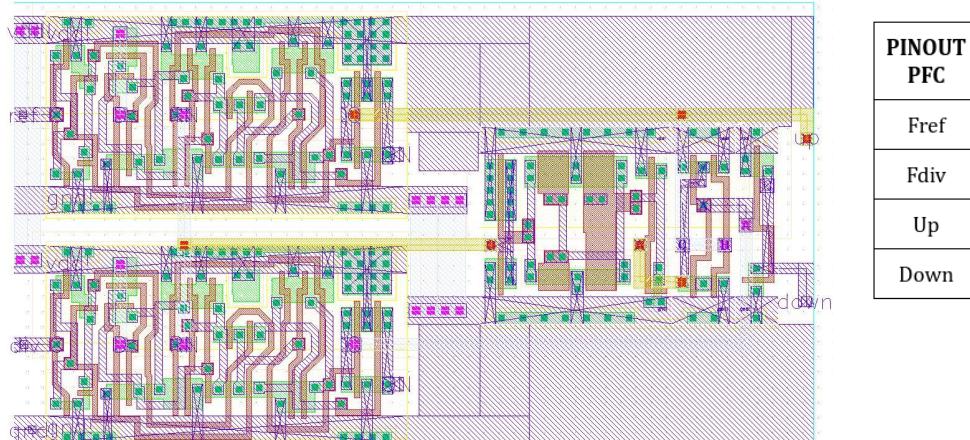
Simulation temporelle : PLL en boucle ouverte

Légende :

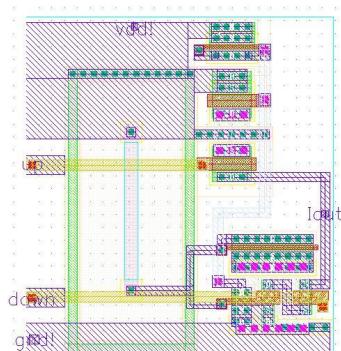
- **Rouge** : fréquence du signal de sortie du VCO
- **Orange** : signal de contrôle du VCO
- **Violet** : sortie du diviseur de fréquence

On observe que la fréquence de sortie augmente, ce qui est le comportement attendu.

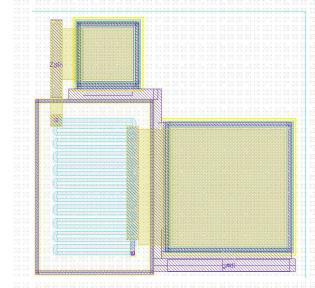
Layout du PFC:



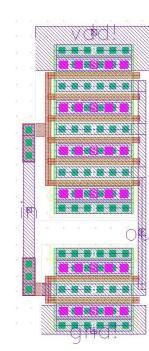
Layout de la Charge Pump:



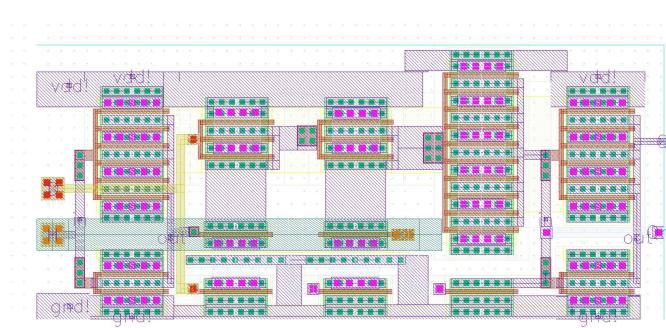
Layout du Filtre:



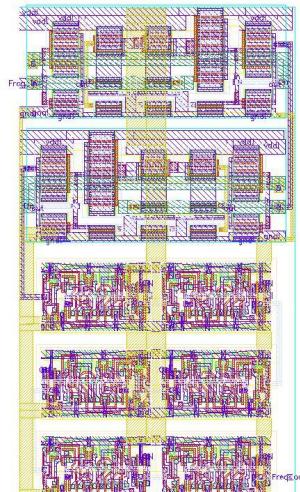
Layout de l'inverseur HF:



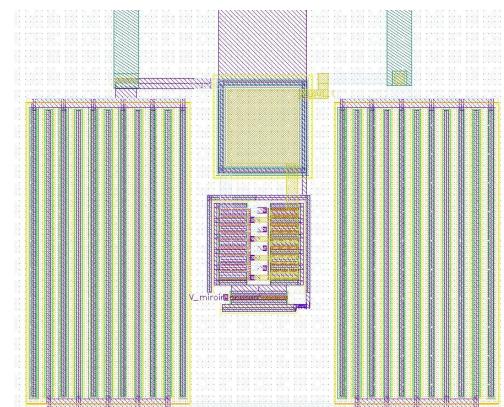
Layout d'une Latch Dynamique HF :



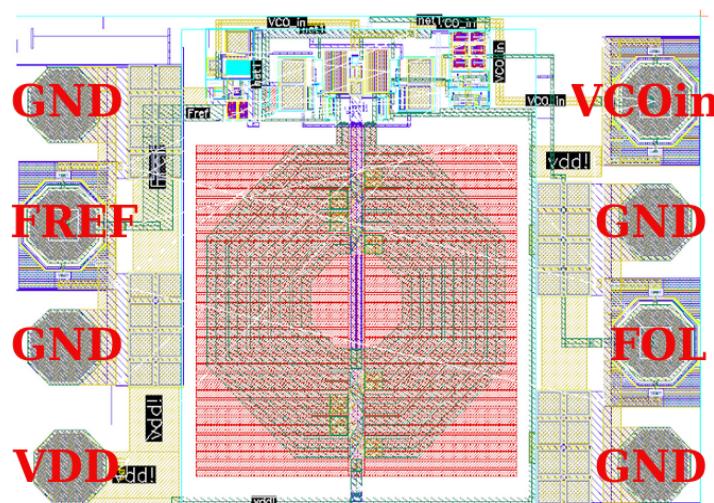
Layout du Diviseur de Fréquence /256 :



Layout du VCO (sans l'inductance):



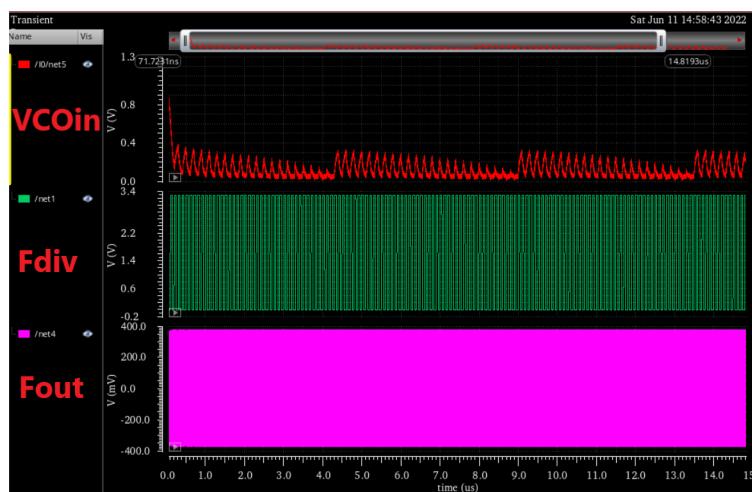
Au final, on a rassemblé tous les blocs pour former la PLL. En se basant sur la cahier des charges qui imposait une disposition des plots, nous avons obtenu le layout suivant :



Top-view du chip de la PLL complète - Taille : $900\mu\text{m} \times 615\mu\text{m}$

Le module PLL possède donc une surface de 0.55mm^2 ce qui en fait l'un des chips les plus petits. Toutefois, par manque de temps, nous n'avons pas optimisé le chip. On aurait par exemple pu répartir les capas de découplage en longueur, de sorte à ce qu'elles prennent moins de place en largeur.

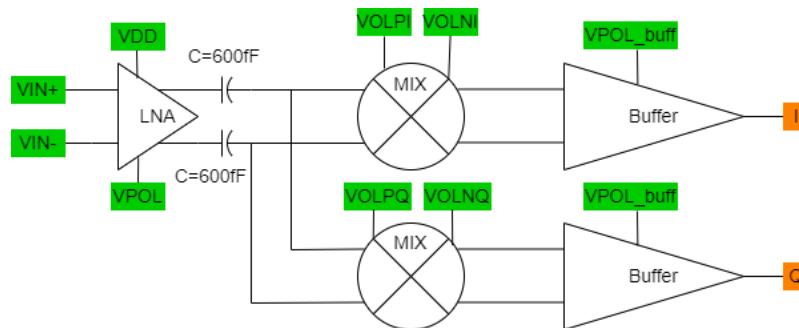
Pour ce qui est de la simulation post-extract du circuit, nous n'avons pu simuler que quelques μs en raison d'un espace mémoire limité. On obtient le résultat qui ne nous permet pas de valider définitivement la PLL, mais cela nous a donné un aperçu du comportement :



1.5 Récepteur RF (Yann ZYZELEWICZ et Manon TABARDEL)

Fonctionnement et choix d'architectures

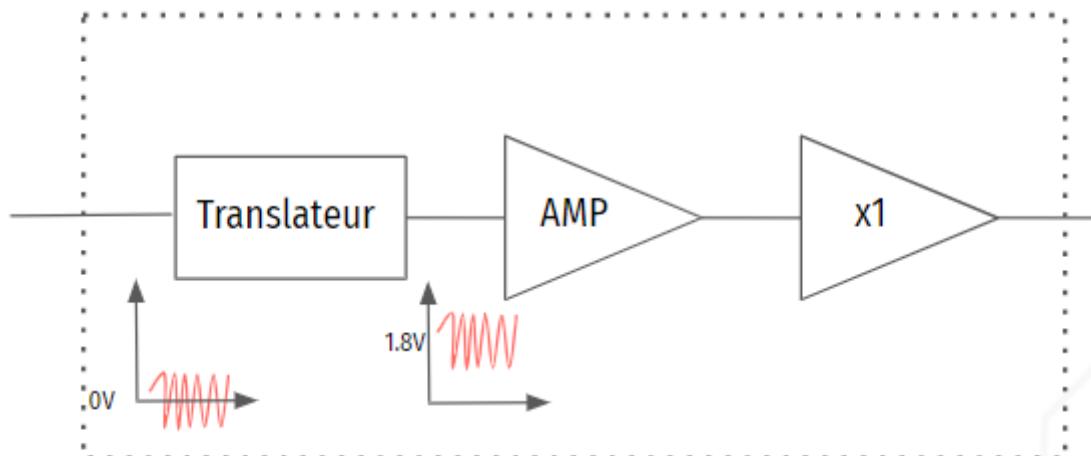
Le récepteur de notre chaîne de transmission est composé de trois éléments majeurs, le LNA différentiel, qui permet d'amplifier le signal reçu et rejeter le bruit du reste de la chaîne, le mixer qui permet de réaliser la transposition de fréquence pour le traitement numérique, et les buffers de mesure, qui permettent d'adapter l'impédance de sortie de la chaîne pour pouvoir réaliser des mesures avec les appareils qui seront utilisés. Voici le schéma du récepteur :



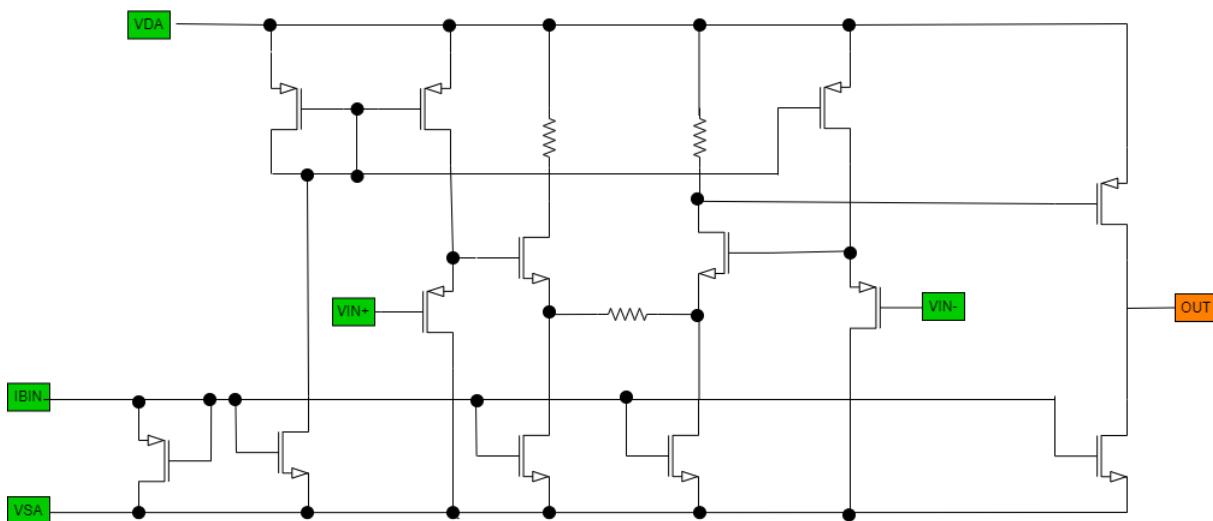
Architecture de la chaîne de réception

Des éléments notables ont été ajoutés, notamment au niveau de la gestion de mode commun, qui a été le fil conducteur de notre assemblage. Le premier élément de gestion de mode commun est la capacité de liaison placée entre le LNA et les mixers. En effet, le mode commun du signal en sortie du LNA est de 3V. En prenant en considération l'étude préalable du mode commun de l'OL, il faudrait fixer ce dernier au mode commun RF, ce qui impliquerait d'avoir une tension VOL relative maximale de 6V. Ce niveau de tension est très élevé, ce qui pourrait poser des problèmes de domaines de tension sur les MOS utilisés. C'est pour cela que des capacités de liaison ont été positionnées suite au LNA, afin de ramener ce mode commun à 0V.

Le deuxième élément de gestion du mode commun se situe au niveau du buffer. Voici son schéma bloc et son design :



Architecture simplifiée du buffer de mesure



Architecture détaillée du buffer de mesure

Le design du buffer fourni ne comportait qu'un étage d'amplification et un étage unitaire pour réaliser l'adaptation d'impédance en sortie. L'étage d'amplification est composé d'une paire différentielle chargée par deux résistances, il faut donc la polariser par un signal avec un mode commun élevé afin de rester dans la zone de saturation de la paire différentielle et des sources de courant présents dans les branches. Une étude cherchant à maximiser la linéarité de l'étage amplificateur du buffer a montré, en se basant sur des mesures de THD, que le mode commun idéal en entrée du translateur était de 1.8V. Le mode commun en sortie du mixer étant de 0V, il a été nécessaire de trouver un moyen de le réhausser afin d'assurer la linéarité du buffer.

Deux options s'offraient à nous, soit ramener un mode commun en entrée du mixer à 1.8V à l'aide de grands ponts de résistances, soit d'utiliser des translateurs réalisés par un montage drain commun en amont de l'étage amplificateur du buffer. La première option a été écartée car le layout top étant à ce point bien avancé, l'ajout de ces ponts de résistances aurait

nécessité des changements importants. La deuxième option a été privilégiée, notamment de par sa simplicité, mais elle dégrade cependant considérablement les performances en bruit de la chaîne. En effet, le NF du buffer seul était de 23 dB, contre 30 dB avec l'ajout de translateurs.

Ce bruit aurait pu être réduit au prix d'une grande surface au layout, mais si l'on prend en compte le fait que ce buffer ne sert qu'à la mesure, et ne serait pas présent sous cette forme dans une chaîne de transmission réelle, ses performances importent peu. Pour cette raison, les spécifications dans la suite de cette partie seront données avant buffer, pour l'étude système et les performances réelles, et avec buffer pour avoir une référence de comparaison lors des TP de mesures.

Simulations

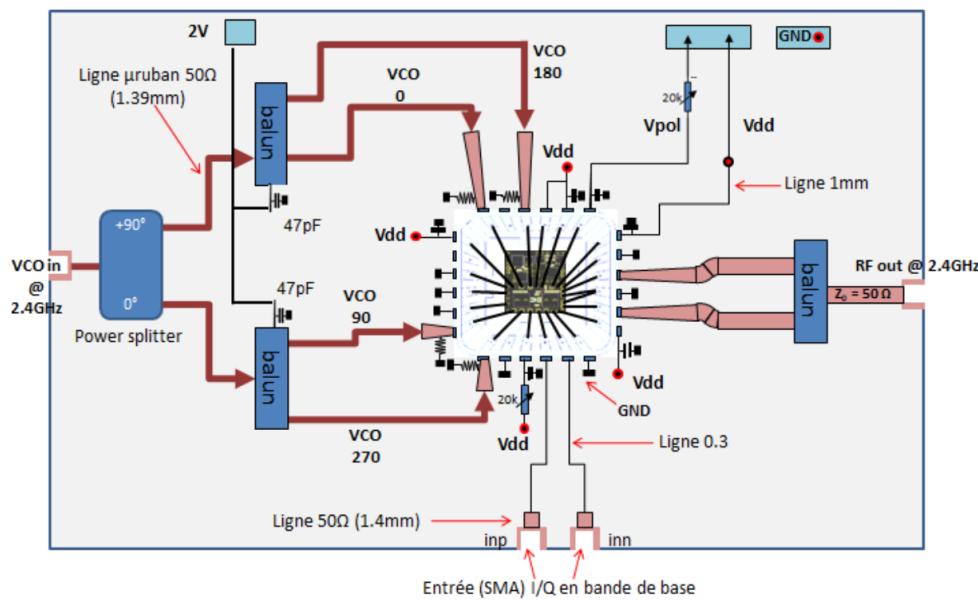
Afin de simuler la chaîne complète, on assemble les testbenches du LNA et du Mixer, à la différence que les buffers sont chargés par des ports d'impédance 1M Ohms et 3pF de capacité afin de simuler l'impédance des appareils de mesure. On utilise les mêmes méthodes de simulation que pour le mixer, ce qui nous permet d'obtenir les performances suivantes :

	<i>Cahier des charges</i>	<i>Avant buffer</i>	<i>Après buffer</i>
<i>Gain (dB)</i>	20	14,3	16,9
<i>ICP (dB)</i>	-2	-2,6	-16,2
<i>IIP3 (dBm)</i>	13	7,5	-0,5
<i>NF (dB)</i>	3,5	7,45	18
<i>Consommation (mA)</i>		16,5	16,8

Layout et simulations post layout

Pour faciliter le layout de la chaîne de réception, nous avons réalisé le layout de chacun des blocs de manière indépendante puis nous les avons assemblés. Pour que le layout final soit le plus efficace possible nous avons dû penser le floorplan de telle sorte à respecter les plots d'entrée/sortie imposés par la carte, le layout doit également être le plus compact possible afin d'utiliser le minimum de surface de Silicium.

La carte sur laquelle sera présente la chaîne de réception est présentée ci-dessous:

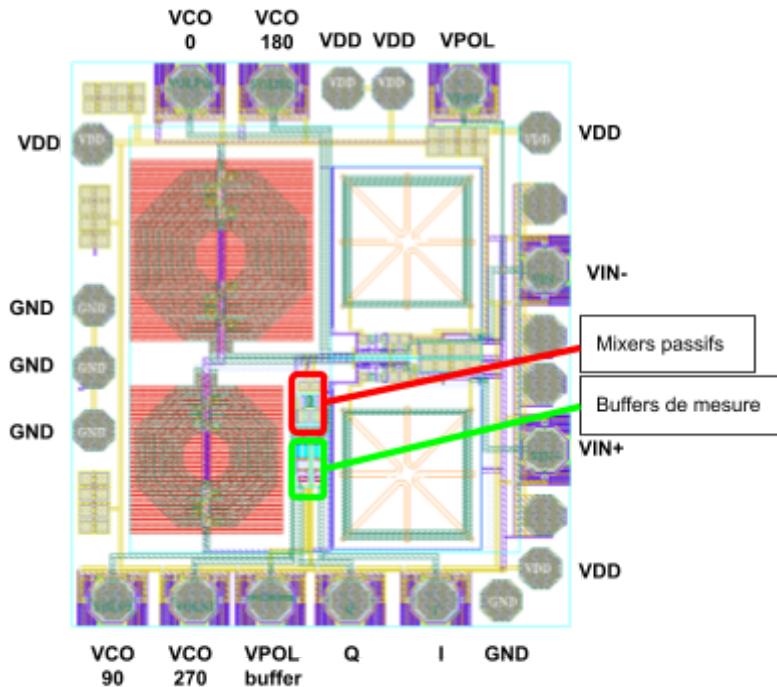


Carte de test du récepteur

Sur la carte nous pouvons repérer les entrées du LNA différentiel avec le balun de droite. Ces plots permettent l'entrée des signaux RF, il faut donc utiliser des plots capables de supporter la RF. Pour cela nous utilisons la vue layout des sondes RF.

Nous pouvons également repérer les entrées OL des mixers sur le haut et le bas de la carte. Les sorties I et Q se trouvent en bas de la carte.

Sur le layout nous avons mis 4 capacités de découplage entre le VDD et le GND. Cela permet de limiter les variations de tension lors d'appels en courant.



Layout de la chaîne de réception, taille=1153x1298 μ m

Comme nous pouvons le remarquer sur la figure ci-dessus, ce sont les inductances qui fixent la taille du layout. Pour ne pas perdre de la place, nous avons décidé de positionner les mixers et les buffers de mesures dans les espaces libres entre les inductances. Cela a permis de ne pas ajouter d'espace utilisé sur les bords du LNA et de cette façon la taille finale du circuit est définie par la taille du LNA qui elle même est imposée par la taille des inductances.

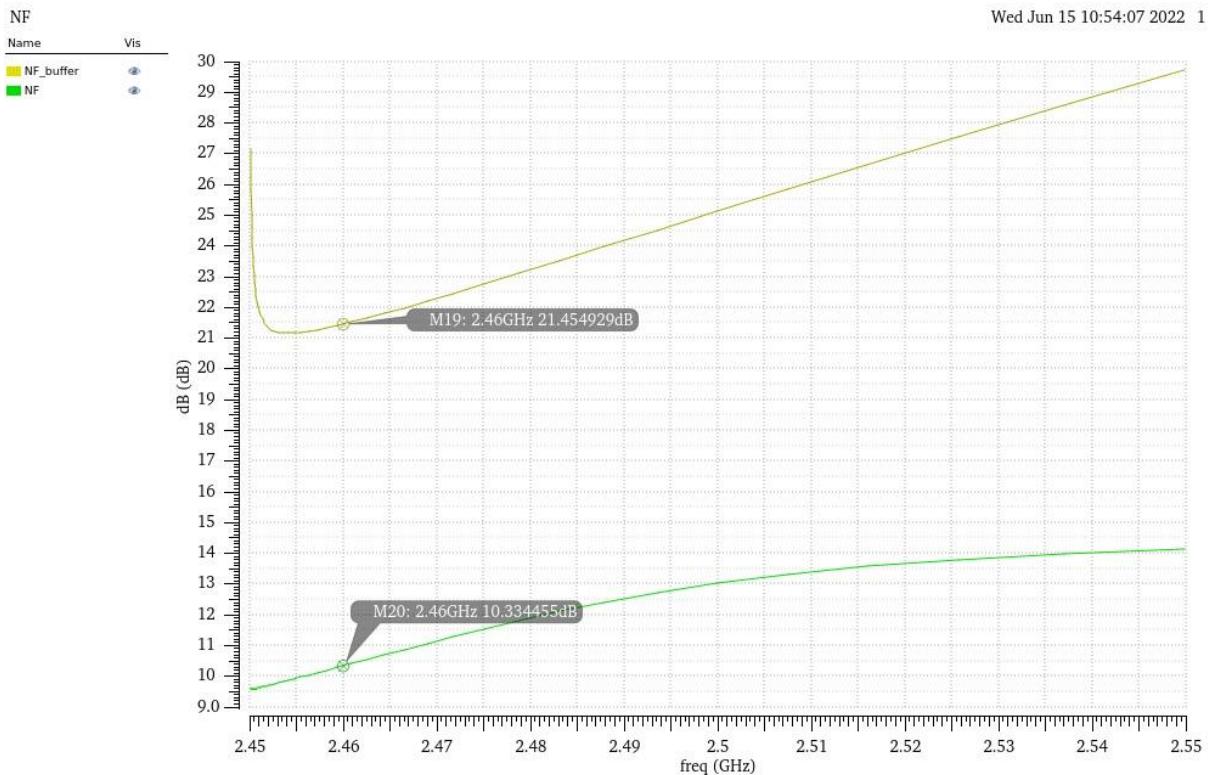
A la fin du layout nous avons rajouté des fillers pour réaliser le plan de masse. Cela permet de limiter les boucles de courant et assurer une masse commune à tout le circuit.

Les résultats des simulations post-layout sont présentés dans le tableau ci-dessous.

	<i>Cahier des charges</i>	<i>Post layout avant buffer</i>	<i>Post layout après buffer</i>
Gain (dB)	20	12,3	14,8
ICP (dB)	-2	-6	-14
IIP3 (dBm)	13	-0,373	3,8
NF (dB)	3,5	10,3	21,5

Résultat des simulations post layout

On peut remarquer que les buffers de mesures influent sur les performances du circuit. Cependant, dans une véritable chaîne de réception ces buffers n'existent pas, pour voir l'impact de la chaîne de réception au niveau du système global il ne faut pas les considérer.



Courbe du NF avec et sans buffer en post layout

On peut remarquer que les buffers de mesures dégradent grandement le bruit du récepteur. (voir la partie précédente)

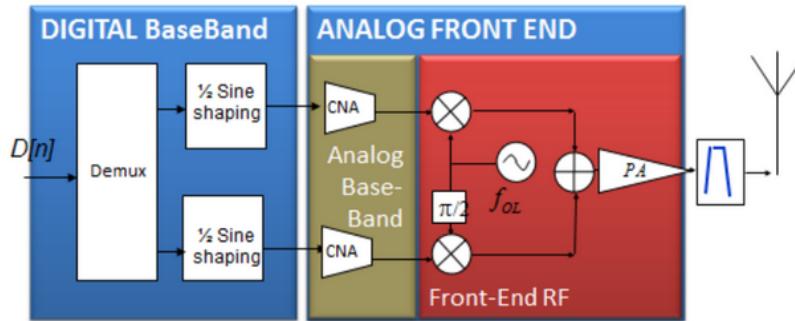
	Cahier des charges	Post layout avant buffer	Post layout après buffer
Sensibilité	-91.5	-85.2	-74
Portée maximale	43 m	27.5 m	14.5m
Portée minimale	14 cm	8 cm	21 cm

Impact de la chaîne de réception sur le système complet

La chaîne de réception que nous avons réalisée remplit le cahier des charges. Le NF demandé par le cahier des charges est difficilement réalisable mais avec le NF que nous obtenons post-layout nous pouvons affirmer que le récepteur que nous avons conçu entre dans les performances d'un récepteur Zigbee.

1.6 Emetteur RF (Pierre-Olivier Guessard & Nathan Loisy)

C'est dans cette partie que nous allons assembler la chaîne TX complète.



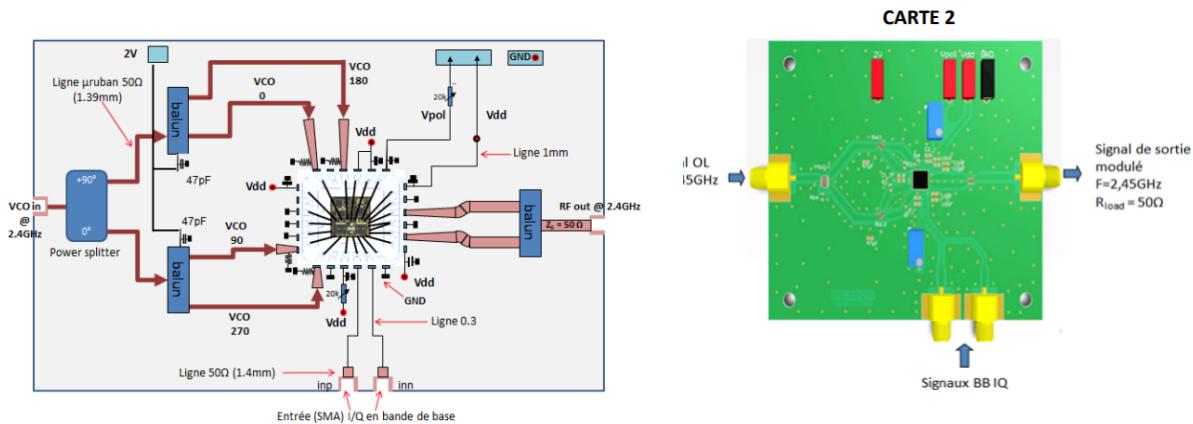
Architecture système Émetteur

Notre émetteur RF recevra les signaux des convertisseurs numériques analogique, et aussi les signaux Fol qui ne seront pas ceux de la PLL en réalité mais seront issus de générateurs lors des tests. Nous souhaitons à l'aide de cette chaîne TX obtenir la meilleure portée possible à travers l'antenne. Nous avons les objectifs suivants :

TX	CDC
Conso (mA)	X
OCP1 (dBm)	13
Psat (dBm)	14

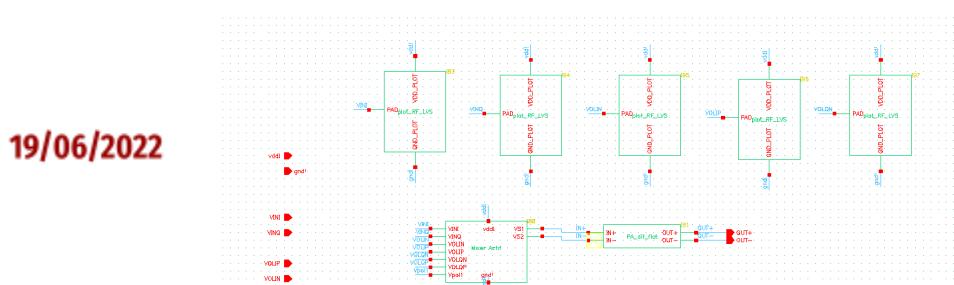
Cahier des Charges Spécifications Emetteur RF

La carte sur laquelle sera testée la chaîne de transmission est présentée ci-dessous, on se servira des sondes RF pour les signaux en sorties de la chaîne TX, des plots RF pour les signaux d'entrées et des PAD80 pour le GND et le VDD.



Carte de test Emetteur RF

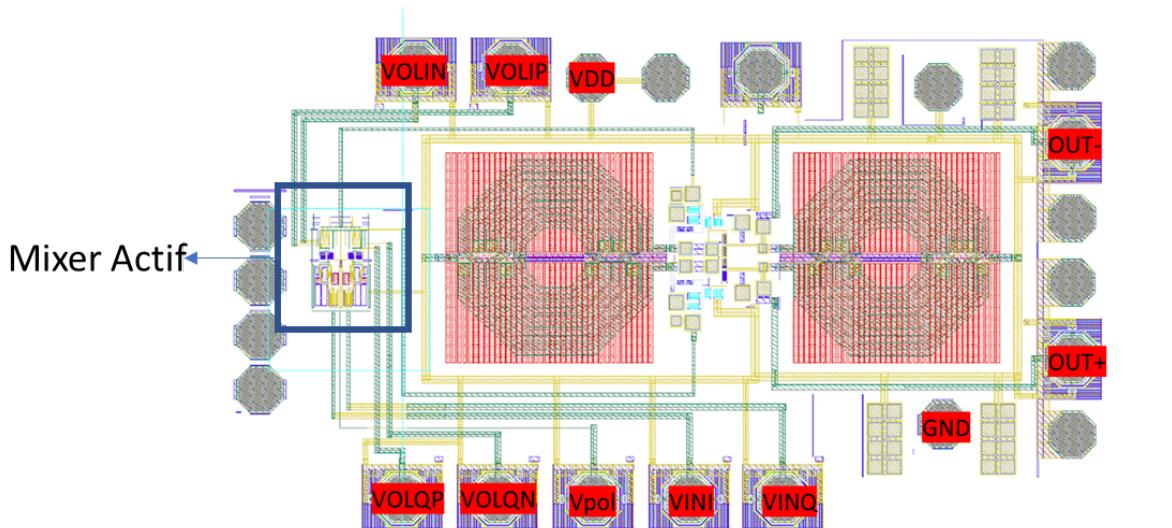
Nous assemblons donc le mixer actif et le PA ensemble afin d'obtenir le schematic suivant :



Schematic Assemblage Mixer + PA

Layout :

Nous effectuons ensuite le layout de la chaîne d'émission. Vu que le layout du PA était en avance par rapport au mixer, il y a eu un manque de communication pour la mise en commun ce qui a causé des problèmes de symétries et d'optimisation au niveau de l'assemblage des deux composants.



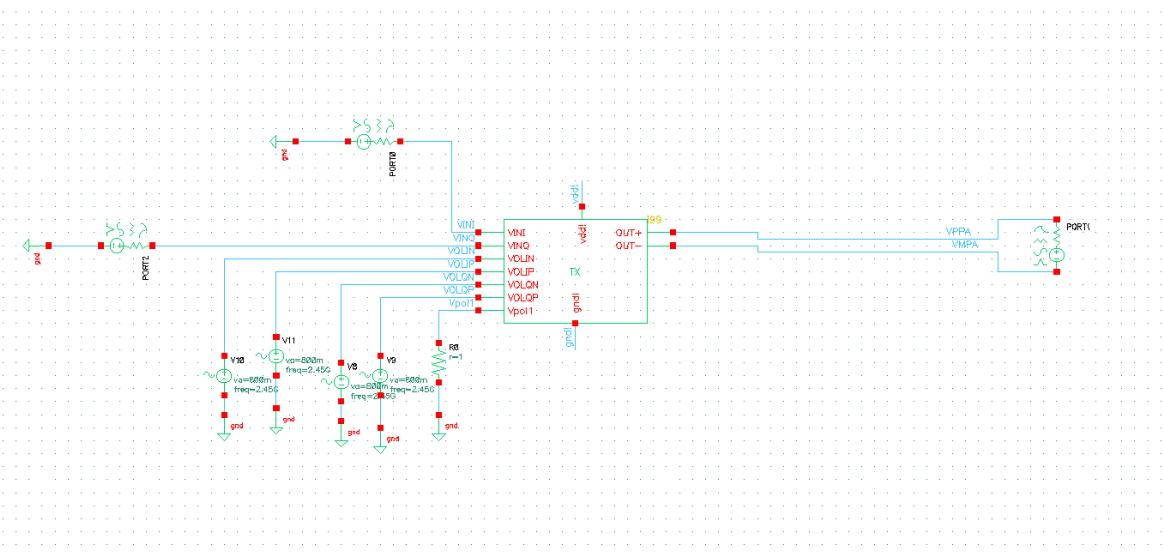
Layout Assemblage Mixer + PA

Nous avons décidé de placer le mixer à gauche du PA. Comme nous le constatons, le mixer prend très peu de place par rapport aux différents plots et surtout aux deux inductances du P.A. Malgré le fait que le mixer ne soit pas au centre du layout, nous avons essayé de le placer et de router les différentes pistes le plus efficacement possible afin de ne pas perdre trop de performance.

Après avoir posé les plots et rajouté les 4 capacités de découplage, nous effectuons les différentes simulations pour cette chaîne TX.

Simulation :

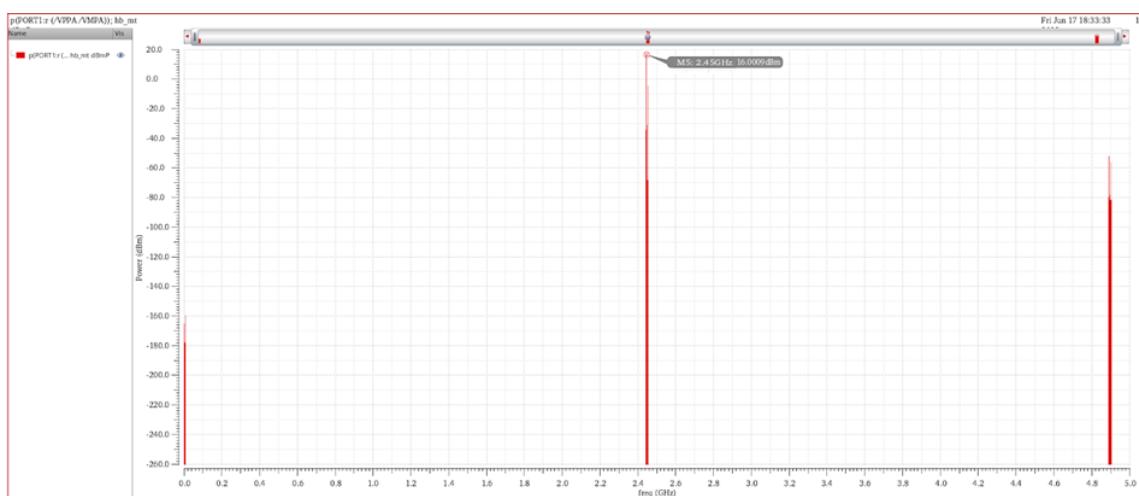
Nous testons tout d'abord la puissance en sortie de la chaîne d'émission à l'aide du bench suivant



Bench Chaîne TX

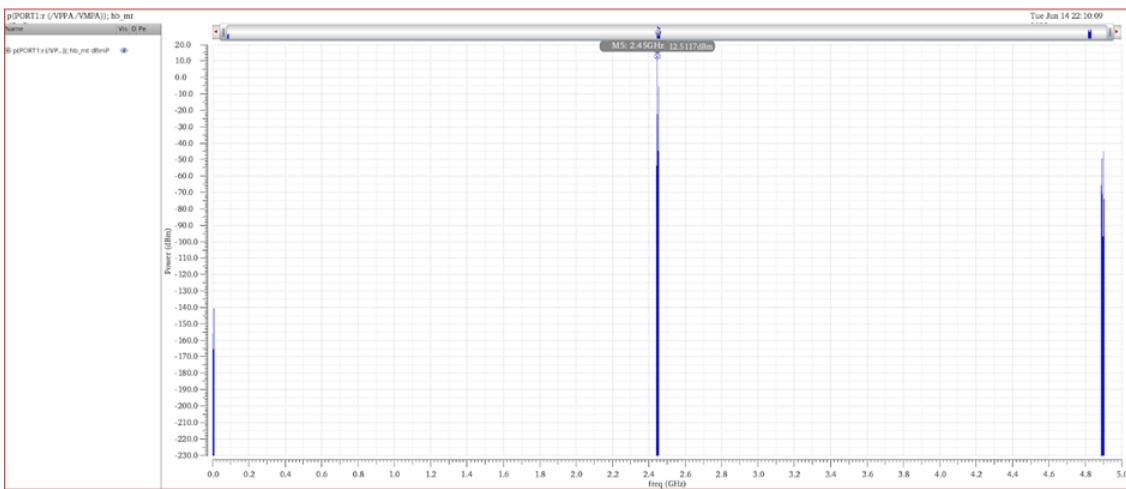
Nous envoyons dans les mixers actifs les différents signaux de 2.45 GHz de 800 mV d'amplitude déphasés de 180°, et les deux signaux IQ de 2.5 MHz dans le balun. En sortie, nous devons émettre sur une antenne différentiel d'impédance 100Ω.

En effectuant une analyse hb, nous obtenons une puissance "max" en sortie de la chaîne d'émission de 16 dBm.



Puissance chaîne TX (dBm) schematic

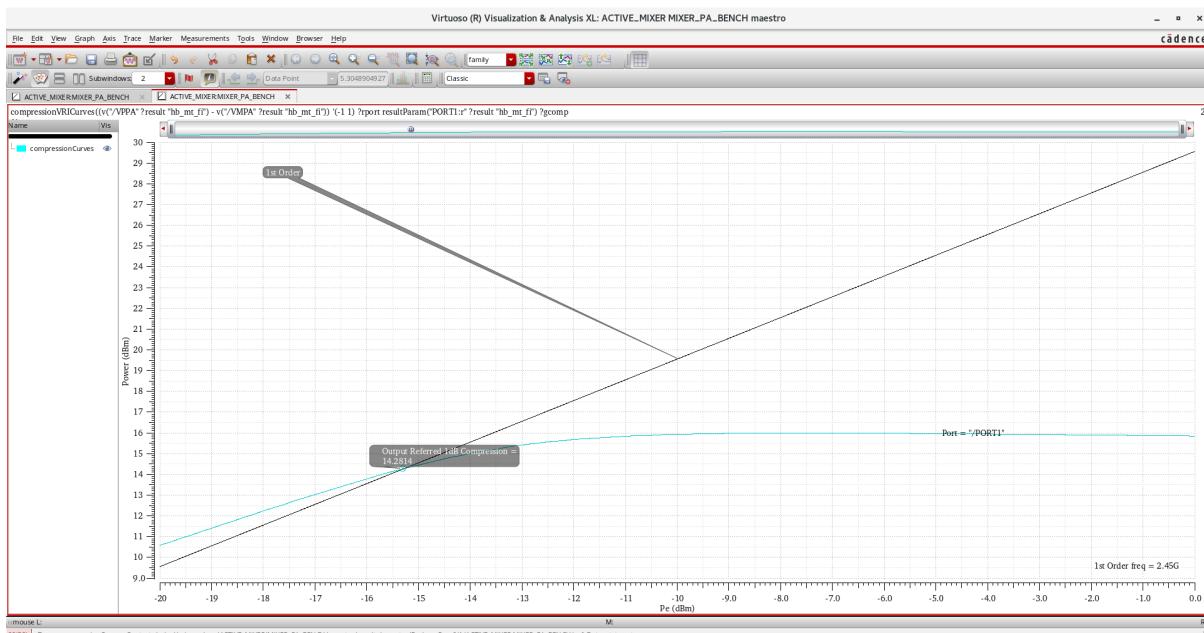
En simulation Post Layout, nous obtenons seulement 12.51 dBm



Puissance chaîne TX (dBm) PLS

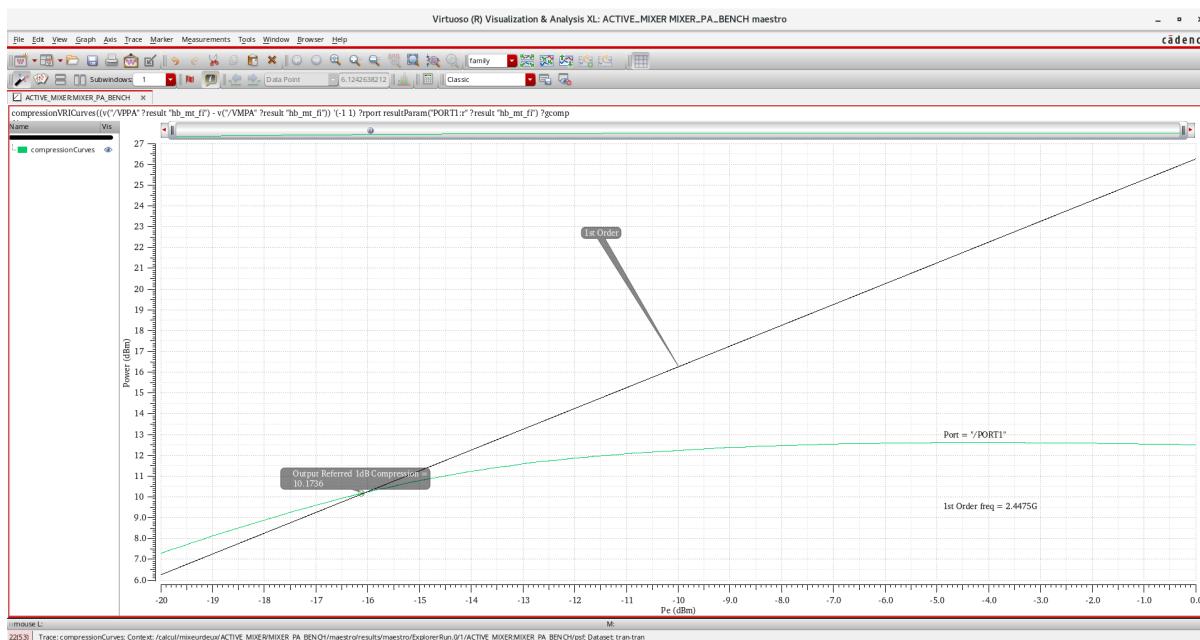
Ces pertes entre la simulation PLS et schematic sont dues à la désadaptation entre la cellule du Mixer et du P.A. Le routage entre les deux blocs a créé plusieurs capacités parasites de l'ordre de la centaine de fF, c'est pourquoi nous avons réduit la taille des pistes afin de diminuer l'effet capacitif, tout en sachant qu'à l'inverse, l'effet résistif allait augmenter.

De même, nous effectuons les simulations pour le point de compression en sortie :



Point de compression. Pout vs Pe chaîne TX (dBm) schematic

En schematic, nous obtenons un point de compression de 14.28 dBm.



Point de compression, Pout vs Pe chaîne TX (dBm) PLS

En PLS, nous obtenons un point de compression de 10.28 dBm. Là aussi, ces pertes proviennent de la désadaptation entre les deux circuits.

En résumé, voici les résultats obtenu :

TX	CDC	Schematic	PLS
Consommation (mA)	X	105	127
OCP1 (dBm)	13	14,20	10.20
Psat (dBm)	14	16.01	12.5

En schematic, les différentes caractéristiques obtenues sont correctes et respectent le cahier des charges, cependant, en Post Layout, nous obtenons des valeurs en dessous des résultats souhaités. Nous avons effectué plusieurs compromis sur la taille des pistes afin de réduire cela et d'avoir la meilleure optimisation possible.

Il aurait fallu prendre plus de temps en amont pour communiquer sur la meilleure façon d'assembler les deux blocs pour le layout, afin de diminuer les effets résistifs et capacitifs et de réduire les pertes.

1.7 ADC (Antoine Leone et Brian Martinez)

Introduction

Dans un premier temps, on cherche à déterminer la résolution nécessaire pour notre convertisseur. Pour cela on se base sur le rapport signal sur bruit à respecter afin d'avoir un taux d'erreur binaire de 10^{-6} , nécessaire au protocole des ZIGBEE, Puis on ajoute une marge. Suite à quoi on détermine à l'aide de la formule ci-dessous le nombre de bits effectifs à respecter pour notre ADC. Dans notre cas cette valeur est de 2, on choisit donc un ADC 4 bits. Dans un second temps, on cherche à déterminer la fréquence d'échantillonnage de notre convertisseur pour cela on se base sur le théorème de Shannon qui demande au minimum deux échantillons par période. De par la fréquence de 3 MHz de notre signal d'entrée sur les voies I et Q, on pourrait choisir une fréquence d'échantillonnage de 6 MHz. Toutefois on choisit d'avoir plusieurs échantillons par période et donc de prendre 20 MHz. Enfin, suite à une concertation avec l'équipe numérique, afin de déterminer leur fréquence de fonctionnement, il est fut déterminé que la fréquence d'échantillonnage serait à 10 MHz dans le but d'assurer le fonctionnement des circuits numériques. Pour finir, on utilise la figure ci-dessous afin de choisir le type de convertisseur.. Dans notre cas, on choisit un ADC flash plus facile à implanter et plus adapté à notre application qu'un convertisseur SAR. Toutefois un ADC SAR aurait moins consommé.

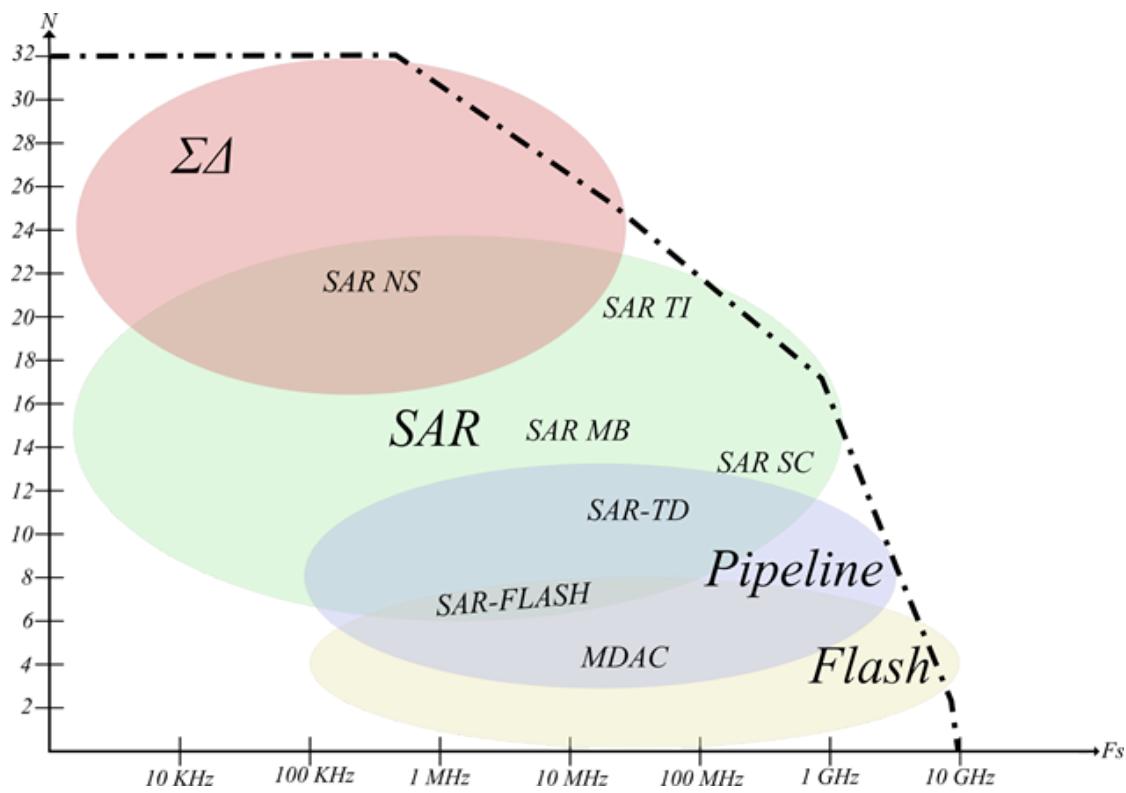


Figure 0 : Etat de l'art réalisé pour déterminer le type d'ADC à choisir

Pré-étude de l'échantillonneur bloqueur

Dans cette première partie, nous allons expliquer et justifier les choix que nous avons fait lors du dimensionnement de l'échantillonneur bloqueur. Nous allons donc commencer par l'amplificateur utilisé puis la structure des interrupteurs.

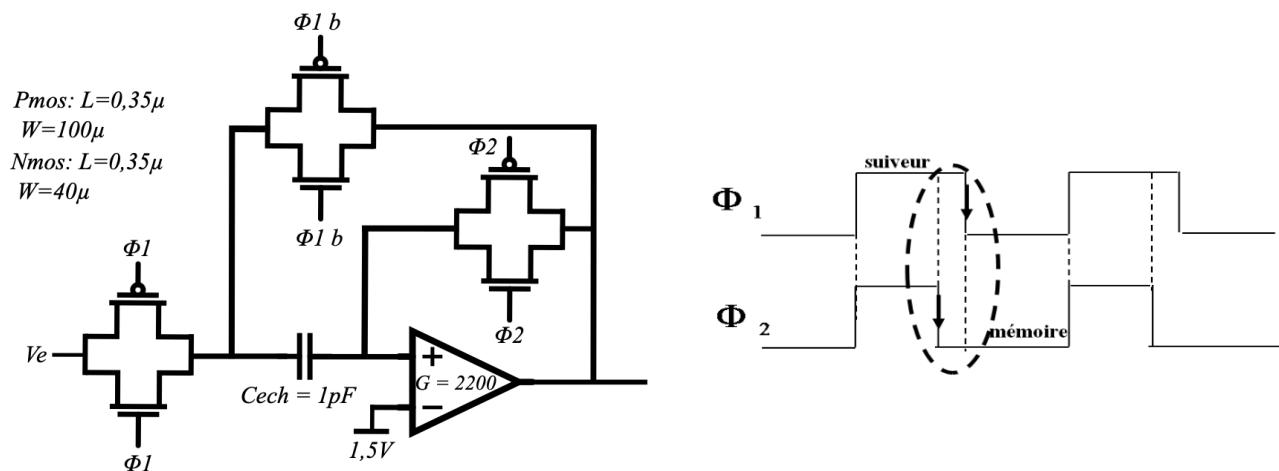


Figure 1 : Schéma de l'échantillonneur bloqueur

Bien que le montage de l'amplificateur nous soit fortement suggéré, on cherche tout de même dans un premier temps à connaître le gain minimum nécessaire pour notre montage. Pour cela, on s'intéresse à la loi de conservation des charges:

$$\frac{V_s}{V_e} = \frac{G}{1+G}$$

Or nous ne voulons pas que la différence entre l'entrée et la sortie soit supérieure à 1 LSB, on a donc pour un échantillonneur bloqueur 6 bits:

$$\frac{V_s}{V_e} = 1 \pm \frac{1}{2^6 - 1}$$

$$V_s = 0.984 \text{ ou } 1.016 V_e$$

Initialement, on était parti sur un échantillonneur bloqueur 6 bit, puis après discussion avec l'équipe numérique on a revu notre cahier des charges pour se contenter de 4 bits de manière à simplifier le travail des 2 équipes. En réalité on verra plus loin qu'on est largement au dessus de ce gain, mais comme notre montage fonctionne en suiveur, plus le gain est

élevé plus on se rapproche d'un fonctionnement suiveur donc on doit simplement avoir une valeur de gain plus élevée que celle déterminé.

$$G = 0.984 \text{ donc } 1 - 0.984 \text{ d'où } G = 61$$

Dans le montage proposé dans le TD4 de Laurent AUBARD (Cours Circuit analogique 1) on avait déterminé un gain très supérieur à 1000 ce qui fonctionne parfaitement. On reprend donc dans un premier temps les valeurs qu'on avait déterminées dans ce TD.

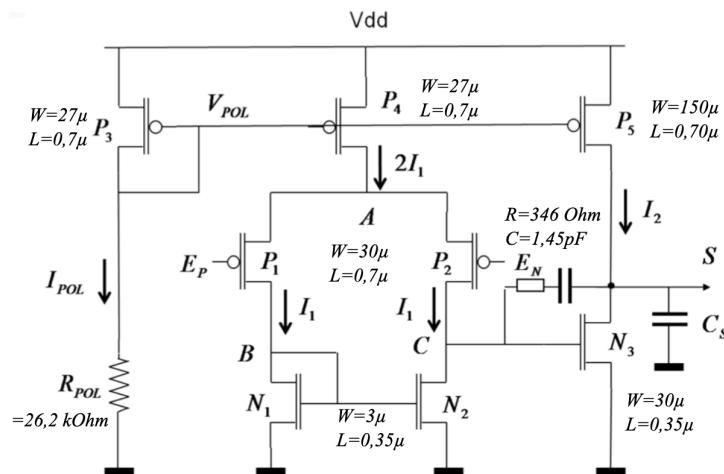


Figure 2 : Schéma de l'amplificateur

Ensuite pour dimensionner la capacité d'échantillonnage et les interrupteurs on reprend le TD1 du cours de Conversion de 2ème année. Dans un premier temps, on rappelle qu'on utilise des interrupteurs de type CMOS qui va permettre de réduire le R_{on} (le RON correspond à une résistance qui varie selon le signal qu'on applique à l'entrée du transistor) et surtout d'avoir un RON le plus constant possible: en effet on recherche avant tout un RON constant car la bande passante dépend directement de RON ($T = R_{on} \times C$) et on ne souhaite évidemment pas avoir une bande passante qui varie pour notre échantillonneur bloqueur.

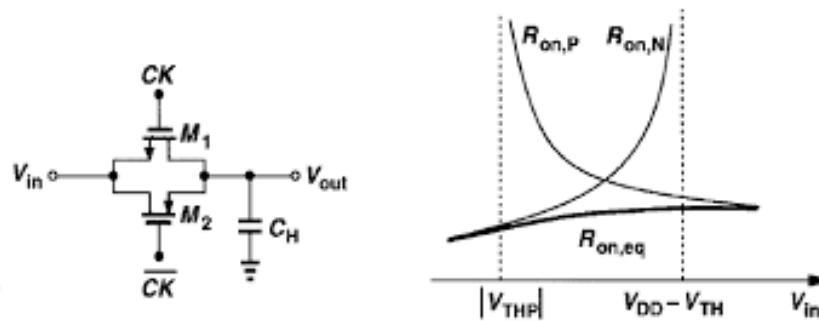


Figure 3 : Schéma de l'interrupteur CMOS

On déterminera lors des simulations le bon rapport entre le NMOS et le PMOS pour avoir le Ron le plus constant possible.

Pour ce qui est de la capacité d'échantillonnage, on nous conseille dans l'énoncé 1pf et cela pour plusieurs raisons: le choix de cette capacité est un compromis. Il faut une capacité suffisamment petite pour qu'elle ait le temps de se charger par rapport au temps d'échantillonnage mais pas trop petite non plus de manière à limiter le bruit en KT/C.

Pour s'assurer de la première condition on peut réaliser un calcul de temps de montée pour s'en assurer. On commence par calculer le RON dans les pires conditions (tension d'entrée maximale juste pour un Nmos):

$$Ron = \frac{1}{\mu_n Cox \frac{W}{L} (V_{gs} - V_{tn})} = \frac{1}{110 * 10^{-6} * \frac{100}{40} * ((3,3 - 2,5) - 0,57)} =$$

$$V_1 (1 - e^{-t/\tau}) = V_1 - 0,5 \text{ LSB} \quad \text{donc il vient : } e^{-t/\tau} = 1 - 1 + \frac{0,5}{2^4 - 1} = 0,033$$

$$\text{d'où } t = \tau * 3,40 = Ron * C * 3,40 = 1,7 \text{ ns}$$

Or pour 10 MHz le temps d'une période est de 100 ns, donc 50 pour une demi période, la capacité à donc largement le temps de se charger.

Initialement, on avait choisi des valeurs de PMOS et NMOS plus faibles, mais comme on a pu le voir plus tard en simulation, pour minimiser la résistance on doit prendre de gros transistors ce qui explique les valeurs ci-dessus. On a réalisé des simulations pour déterminer le Ron selon la tension appliquée en entrée et la taille du transistor et nous avons déterminé environ un facteur 3 pour obtenir le Ron le plus constant possible.

Par ailleurs, on précise qu'on aurait pu ajouter des architectures comme le bootstrap switch ou le bottom plate pour réduire le phénomène de charge injectées. Mais comme on a des contraintes assez faibles (vitesse raisonnable et nombre de bit réduit) on peut tout à fait s'en passer sans que cela pose de problème.

On va pouvoir s'assurer que nos calculs soient corrects en passant aux différentes simulations

Schématique et simulation de l'échantillonneur bloqueur

Notre méthode d'approche pour ce bloc fut la suivante: on a décidé de commencer par réaliser une simulation parfaite avec une transconductance parfaite, des interrupteurs idéaux de manière à simplement valider le fonctionnement général de notre architecture, cela permet également de mieux comprendre comment notre architecture fonctionne. Puis, on a

remplacé nos blocs idéaux de manière progressive par les blocs réels, dans un premier temps par un amplificateur réel puis avec des interrupteur réels en conservant une transconductance réel et enfin avec que des composants réels. Cette méthode nous a permis de rapidement cerner les limitation de chaque blocs et des conséquences répercutées sur l'échantillonneur bloqueur en lui-même.

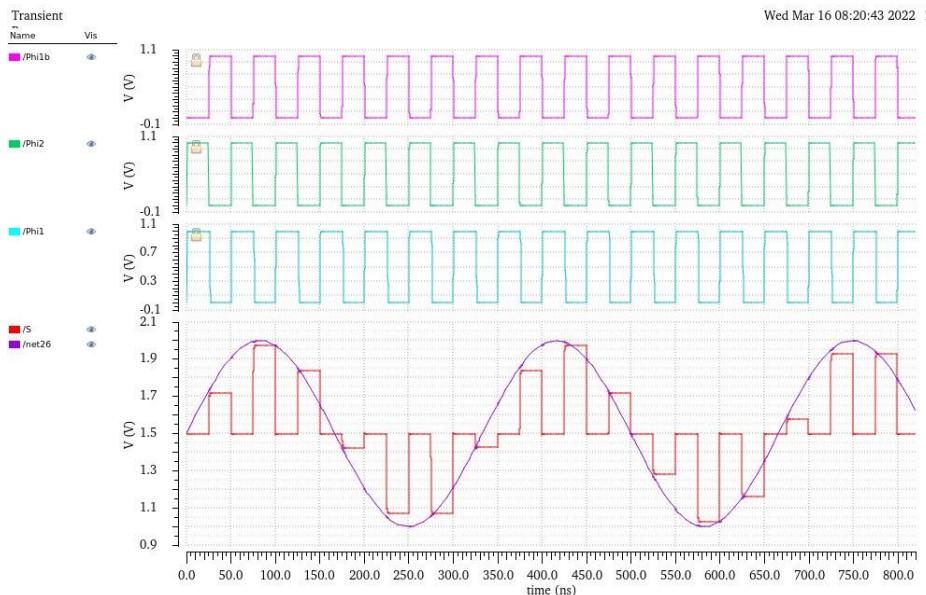


Figure 3: Simulation avec transconductance parfaite et interrupteurs idéaux (pour $V_e = 2V$ avec une fréquence 3 MHz)

Cette première simulation permet de vérifier le fonctionnement global de l'échantillonneur (les fronts sont nets, les commutations des interrupteurs également). Cette première étape permet aussi d'ajuster les phases Phi1 et Phi2. Pour un bon fonctionnement, on détermine par simulation que $\text{Phi2} = 0,9 * \text{Phi1}$. Ensuite, on souhaite intégrer notre amplificateur réel, on commence par réaliser une simulation DC afin de vérifier les différents points de polarisation qu'on avait calculés lors du TD. Cette première étape nous a tout de suite permis de nous rendre compte d'un problème: dans le TD les entrées sont inversées, la référence doit être sur En et non pas sur Ep. On s'assure ensuite avec une étude en boucle ouverte de retrouver un gain suffisant par rapport à ce qu'on a défini au départ ainsi que la même marge de phase (Dans le TD on s'assure 60° de marge au minimum).

On réalise ensuite une simulation mais en remplaçant notre transconductance par l'amplificateur réel. Une fois que les simulations sont concluantes, on peut passer au layout. Notre approche pour la partie Layout a été de nous concentrer sur les blocs élémentaires dans un premier temps (comme les inverseurs, les switchs). Il faut bien rappeler que nous n'avions aucune expérience en layout et qu'il a fallu tout apprendre. Nous avons donc passé un certain temps sur ces blocs de bases, à les refaire, les recommencer à zéro, se tromper et demander des conseils. Cela nous a permis d'aborder ensuite plus sereinement des blocs

plus complexes comme l'AOP, les comparateurs... Une fois les blocs intermédiaire terminés nous avons pu nous concentrer sur les validations de DRC, LVS qui nous ont pris un certain temps pour ensuite réaliser des simulations post layout de chaque sous bloc avant de réaliser une simulation post layout de tout l'échantillonneur bloqueur et du générateur de phase.

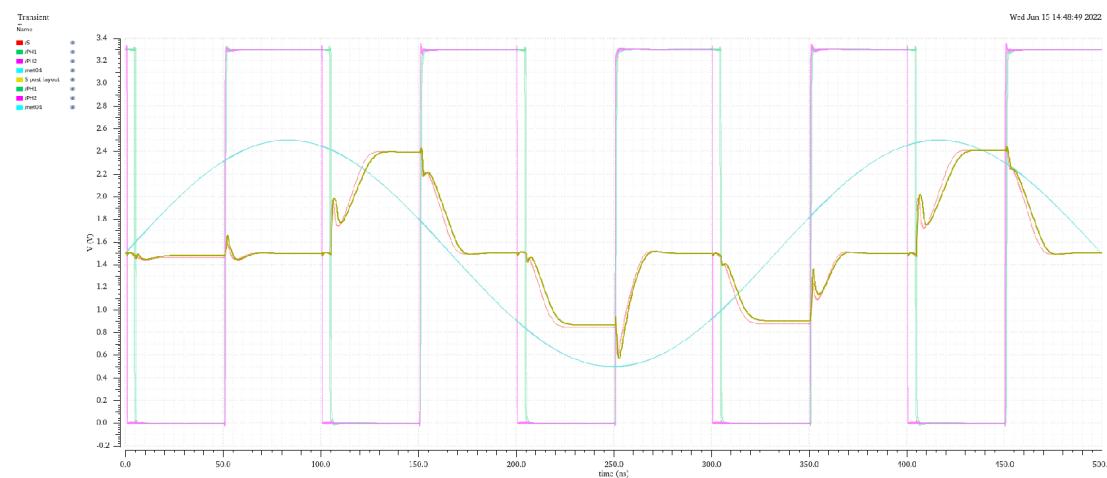


Figure 4: Simulation post layout de l'échantillonneur bloqueur et du générateur de phase comparé aux simulations schématiques respectives (pour $V_e = 2V$ avec une fréquence 3 MHz).

Sur cette simulation on observe que le signal de la simulation post extract est très légèrement déphasé, mais nous avons aucune perte de performances. Pour en arriver là, nous avons dû ajuster le layout des switch qui semblait trop résistif: on a pour cela ajouter des via pour faire chuter sa résistivité. Cela nous a permis de récupérer la légère chute de tension que nous avions pour obtenir la courbe ci-dessus.

Nous avons également eu le temps de réaliser un nouveau bloc: le générateur de phase (dont on peut observer la simulation post layout ci-dessus). Nous avons eu recours à des cellules à retard composé d'inverseurs et de capacités ainsi que des opérateurs logiques pour générer les différents signaux nécessaires au bon fonctionnement de l'ADC (PH1 PH2 PH3 et un signal de End of Conversion).

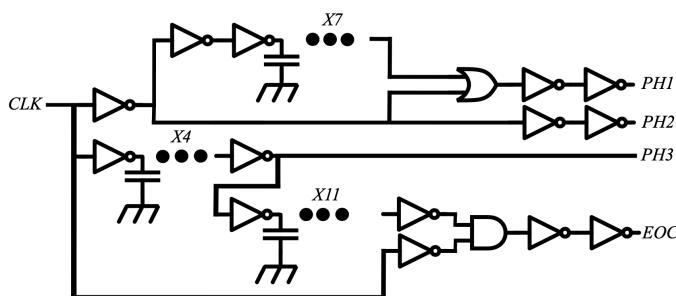


Figure 5: Schéma du générateur de phase

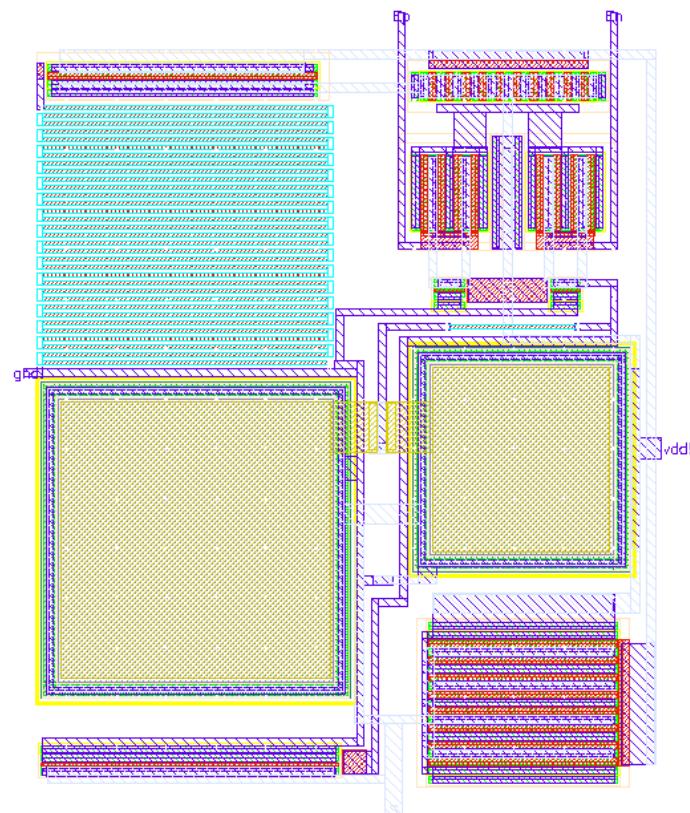


Figure 8: Layout de l'AOP

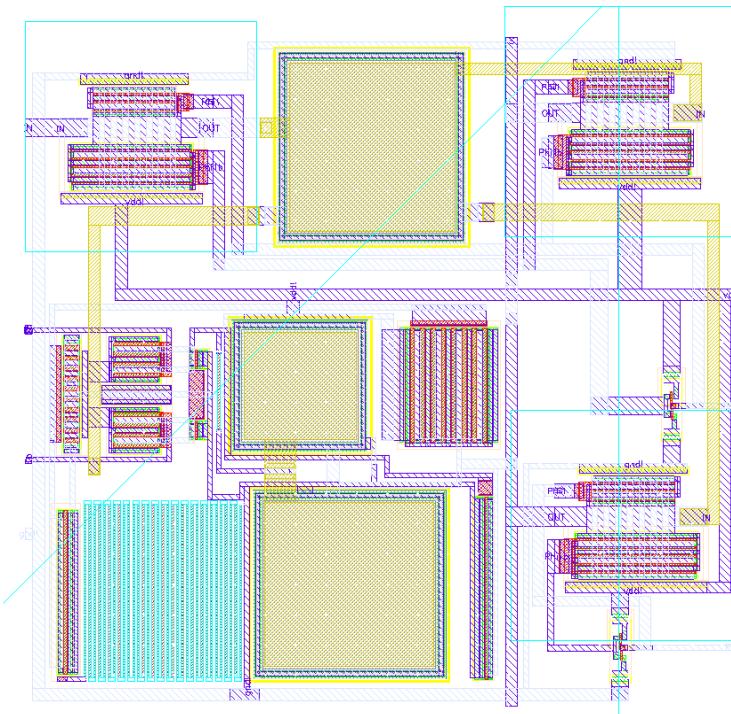


Figure 8: Layout de l'échantillonneur bloqueur

Dimensionnement des comparateurs

Dans cette partie, nous allons présenter la phase de dimensionnement et de validation du comparateur de l'ADC Flash. Celui-ci doit pouvoir avoir une résolution de 4 bits. C'est-à-dire que la différence de potentiel à ses bornes permettant un changement d'état en sortie doit être inférieur au LSB de notre ADC, ce qui correspond à notre pleine échelle divisée par le nombre de codes en sortie de notre CAN.

$$VLSB = \frac{V_{DYN}}{2^N - 1} = \frac{2,5}{15} = 0,167$$

Nous devons donc vérifier le changement d'état de la sortie du comparateur pour une différence, en entrée du comparateur, de 167 m.. Nous allons désormais présenter les calculs permettant le dimensionnement de notre comparateur. Pour cela, on utilise la structure proposée ci-dessous. Elle à l'avantage d'intégrer la partie synchrone qui permet de mémoriser l'état de la sortie durant une demi période d'horloge comme le montre le chronogramme ci-dessous. De plus, certaines parties de la structure sont communes, entre le comparateur et la bascule, ce qui permet de diminuer la surface du comparateur.

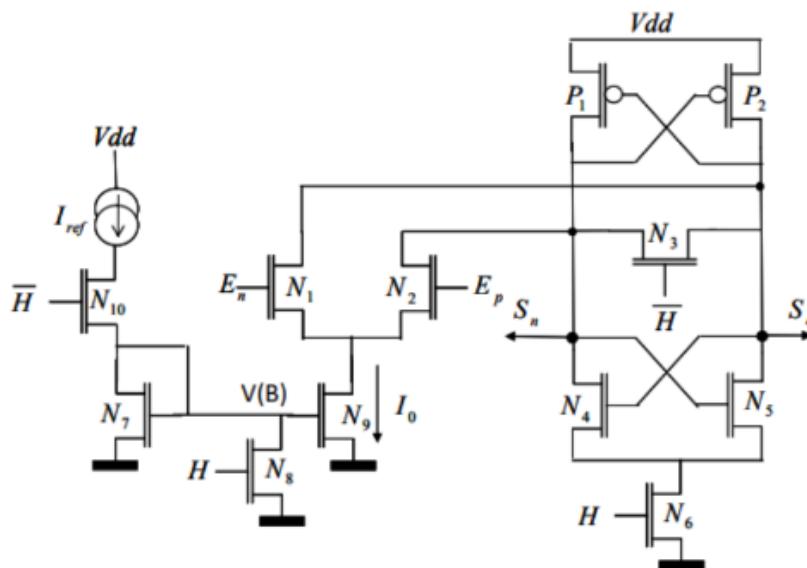


Figure 9: Architecture du comparateur synchrone utilisé

Dans un premier temps, on calcule le courant de polarisation I_0 permettant de charger la capacité aux nœuds S_n ou S_P qu'on estimera à $100fF$ en $1/4$ de la période d'horloge sachant que cette charge représente une variation de tension Vdd . Pour cela, on utilise la formule suivante et double la valeur obtenue afin de s'affranchir de certain problème développé par la suite :

$$I_0 = \frac{C_s \cdot dV}{dT}, \quad dV = 3,3 \text{ V} \Rightarrow dT = 2 \cdot \frac{\frac{1}{4}}{10M} \text{ soit } I = 100fF$$

Dans un second temps, on dimensionne la paire croisée formée par P1 et P2 afin d'avoir une tension de mode commun de 1,65 V en sortie. Dans ce but, on se place dans le cas où VEN = VEP et suis le déroulement proposé ci-dessous.

$$IO = -Kp \frac{W}{L} (vgs - vt)^2$$

$$\frac{W}{L} (P1 \text{ et } P2) = \frac{-IO}{Kp(vgs-vt)^2} = \frac{26}{25(-1.65-0.73)^2} \approx 0.3$$

$$W(P1 \text{ et } P2) = \frac{W}{L} L = 0.3 \cdot 0.7\mu m \approx 0.2\mu m \Rightarrow 0.35\mu m$$

Dans un troisième temps, on dimensionne la paire différentielle formée par N1 et N2 afin de présenter une capacité CGS de 80fF pour le sample hold. Il est à noter que lors du dimensionnement du sample c'est 100fF qui sont considérés. Mais on considère qu'au moins 20 fF seront apportés lors du layout. Enfin, on applique le déroulement proposé ci-dessous.

$$CGS \approx \frac{2}{3} COX \cdot W \cdot L = 80fF \text{ avec } L = 0.35\mu m \text{ et } COX = 5.5fF/\mu m^2$$

$$W(N1 \text{ et } N2) = 60.9\mu m$$

Pour finir, on dimensionne alors le transistor de polarisation N9. Pour cela, on remplace N9 par un générateur de courant de valeur IO (26 μA) et on mesure la tension au nœud A. On mesure alors une tension VA de 0.38 V. On en déduit alors VB tel que vgs - vt = VDS - 0.2V donc vgs (N9) = 0.75V. Enfin, on applique les formules ci-dessous afin de déterminer les dimensions N9.

$$\frac{W}{L} N9 = \frac{IO}{K_N - (vgs - vt)^2} \Leftrightarrow WN9 = 3.5\mu m$$

On prend les mêmes dimensions pour N7 afin de diminuer la valeur de la résistance du circuit de polarisation et donc diminuer la surface.

En définitive, on détermine par simulation que le reste des transistors doit avoir les dimensions minimales afin d'assurer le fonctionnement du circuit. On obtient alors le tableau ci-dessous.

Instance	R _{ref} Ohm	N10-N8-N6-N4-N5 (W/L)	N1-N2 (W/L)	N3 (W/L)	P1-P2 (W/L)	N7	N9
Valeur	500	1 (LMIN)	174 (LMIN)	2.85 (LMIN)	2 (LMIN)	W=0.7μm L=3.5μm	W=0.35μm L=3.5μm

Figure 10: Valeur utilisée pour l'architecture du comparateur synchrone utilisé

Simulation et validation des comparateurs

Dans un premier temps, on retrouve ci-dessous le schéma utilisé comme banc de test du comparateur.

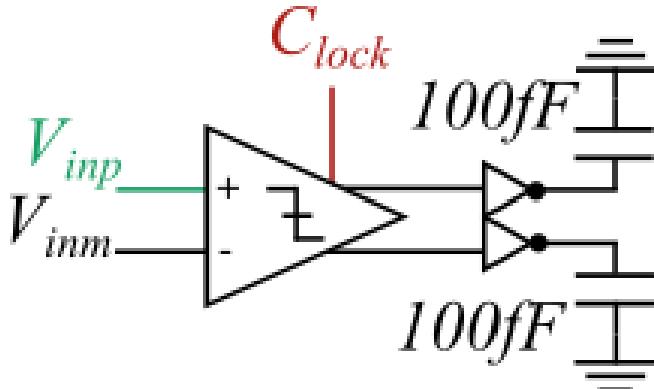


Figure 11: Banc de test utilisé pour le comparateur synchrone

Dans un second temps, on remarque que les sorties de notre comparateur sont reliées à des inverseurs. Le but de ces inverseurs est de permettre de transformer la sortie différentielle de notre comparateur en sortie unipolaire. En effet le décodeur ne possède que des entrées unipolaires, il va donc falloir relier uniquement la sortie négative de notre comparateur à ce décodeur. On utilise la sortie négative car celle-ci est inversée par l'inverseur et nous permet de retrouver le fonctionnement attendu par notre comparateur. Par la suite, on réalise une simulation transitoire de 250ns paramétrique on l'on fait varier Vinp de 1.6 V à 1.7 V en 100 points pour Vinm à 1.65 V. Le but étant de voir à quel point faire basculer la sortie du comparateur. On observe le résultat ci-dessous et on définit via la formule suivant le nombre de bit du comparateur effectif acceptable par la comparateur : $\log_2 \left(\frac{2.5}{V_{MIN}} \right) = 21.92$ bits. Notre comparateur remplit donc amplement le cahier des charges.

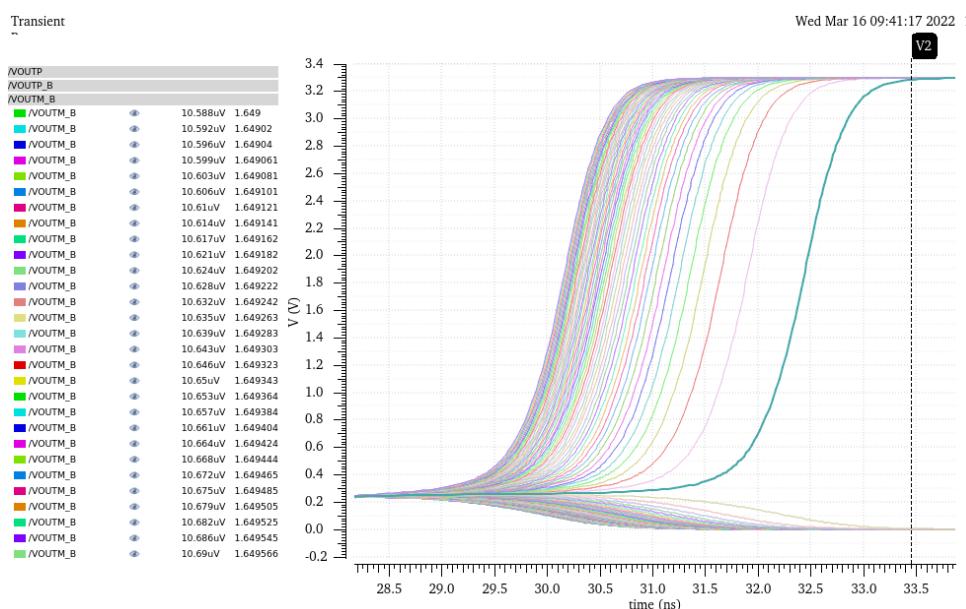


Figure 12: Résultat de simulation pour quantifier la résolution du comparateur

Enfin, on réalise le layout du comparateur ainsi que de l'inverseur en gardant en tête l'objectif d'être le plus symétrique possible afin d'assurer le bon fonctionnement des cellules de quantification. On retrouve ci-dessous les layouts du comparateur ainsi que de l'inverseur.

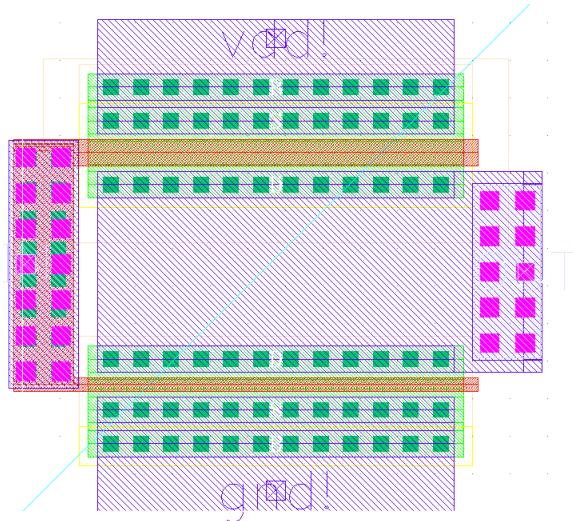


Figure 13: Layout de l'inverseur utilisé en sortie du comparateur

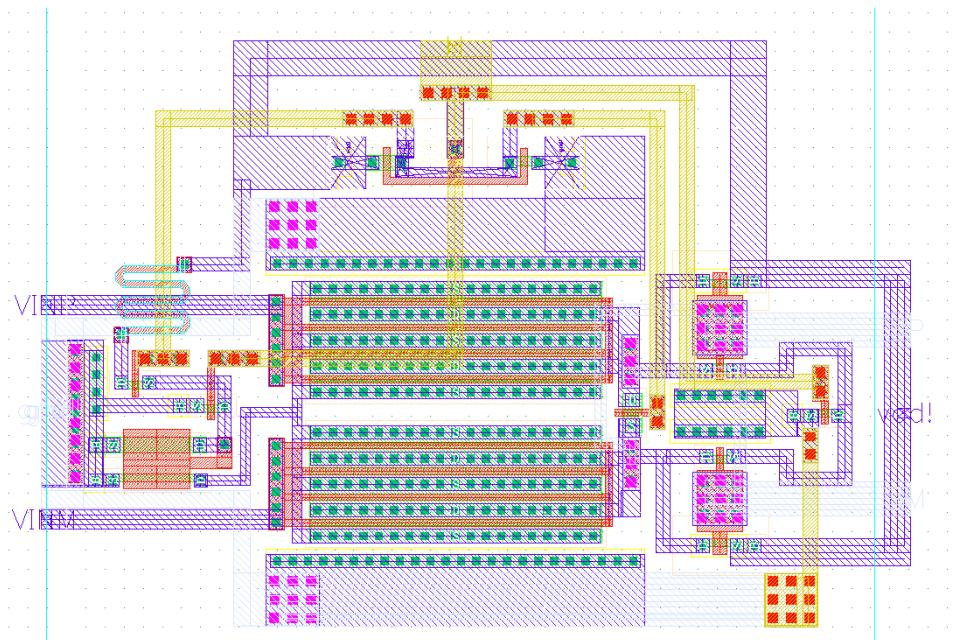


Figure 14: Layout du comparateur synchrone de l'ADC

En définitive on réalise de nouveau les simulations présentées précédemment avec les parasites issus du descendant des masques pour le comparateur et l'inverseur. On obtient alors des résultats légèrement différents. En effet, en appliquant la formule présentée précédemment une perte au niveau de la résolution du comparateur, qui chute à 10,91 bits effectifs, est constaté. Ceci s'explique principalement par la baisse de la consommation et donc de la polarisation du comparateur toutefois le comparateur répond encore aux spécifications techniques fixées pour notre application.

DECODEUR

Travers cette partie 1 présenterons le choix d'architecture réalisé afin d'implanter le décodeur thermométrique vers binaire naturel 4 bits par la suite nous présenterons les résultats de simulation ainsi que le dessin des masques. Pour la conception de ce bloc nous avons choisi d'utiliser un arbre de wallace comme mis en avant sur la figure suivante. Nous disposons de Full-Adder disponibles dans les librairies du fondeur. L'avantage de cet arbre est qu'il est simple à mettre en place, il est combinatoire et donc rapide et simple.

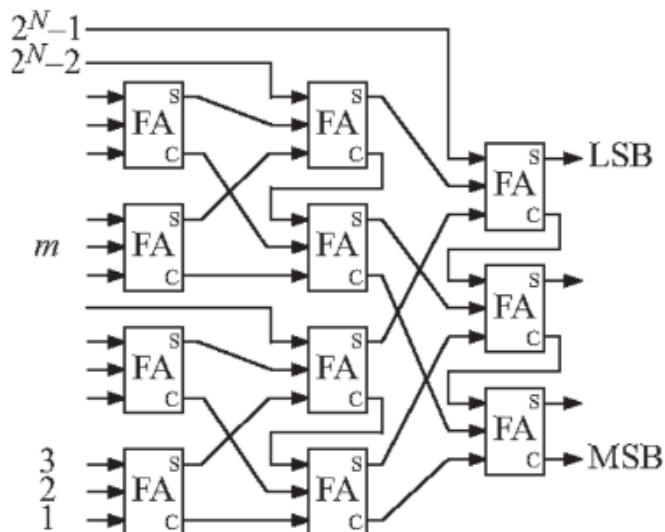


Figure 15: Arbre de wallace mis en place pour le décodeur

Une fois le schéma terminé, nous pouvons simuler le testbench (figure 21). Nous retrouvons une capacité de charge de 100 femto Farads sur chaque bit en sortie. Nous pouvons simuler un code en mettant une partie des entrées à '0' logique, et le reste à '1'. Ce qui correspond donc à 0V et 3.3V. Après avoir testé les valeurs limites et des codes aléatoires, on peut valider le fonctionnement en conditions idéales. De plus, afin d'avoir une représentation graphique du fonctionnement du décodeur, la figure ci-dessous est disponible et démontre le fonctionnement du décodeur pour les 2 bits de points faibles thermométriques.

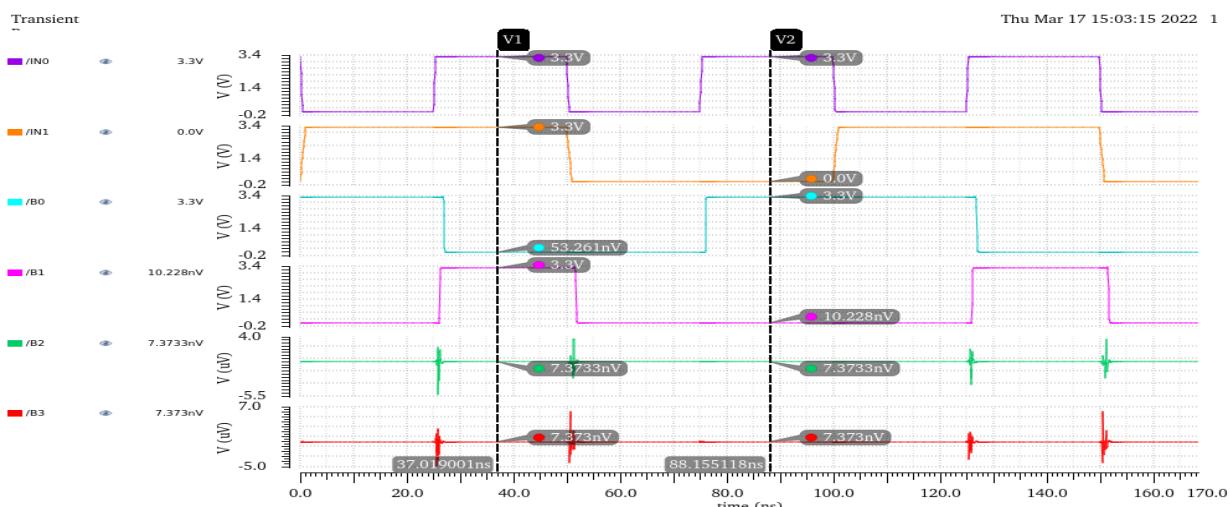


Figure 16: Exemple de fonctionnement du décodeur pour les 2 LSB thermométrique

Pour finir, le layout du décodeur est à base de standard cells, ce qui permet d'avoir un les très compact comme on peut le voir ci-dessous .

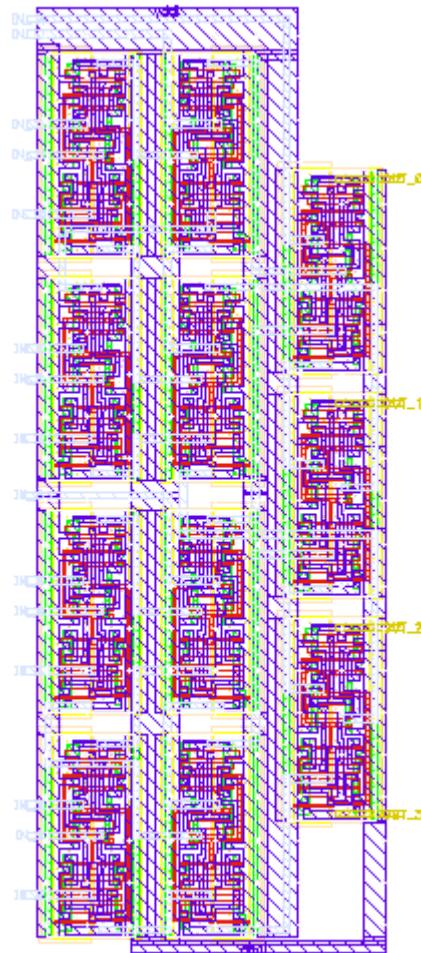


Figure 17: Layout du décodeur thermométrique vers binaire à base de cellule standard

Simulation et validation de l'ADC

Dans un premier temps nous assemblons l'ensemble de nos sous bloc afin de former le schéma fonctionnel un indice et flash 4 bits. On peut voir à cet effet une figure simplifiée ci-dessous. Nous présenterons les différences entre la partie schématique et layout en conclusion.

Dans un second temps, suite à cet assemblage cherchons à quantifier la qualité ainsi que le fonctionnement de notre ADC. Pour cela nous utilisons 2 types de signaux. Le premier est un signal rampe couvrant l'ensemble de la dynamique d'entrée du convertisseur afin de déterminer la linéarité de celui-ci. Ceci se traduit donc par un ADC avec ou sans codes manquant ce qui est directement lié à l'INL et au DNL. le second signal et un signal sinusoïdal de 3 mega Hertz ayant une amplitude couvrant l'ensemble de la dynamique du convertisseur afin d'en définir le bruit

de quantification ainsi que le taux de distorsion harmonique. Enfin à l'aide de ces paramètres on peut déterminer le nombre de bits effectif de notre convertisseur.

Dans un troisième temps, nous nous intéresserons à la consommation de notre convertisseur afin d'établir la figure de mérite de celui-ci et donc pouvoir conclure sur la différence entre le schématique et le layout.

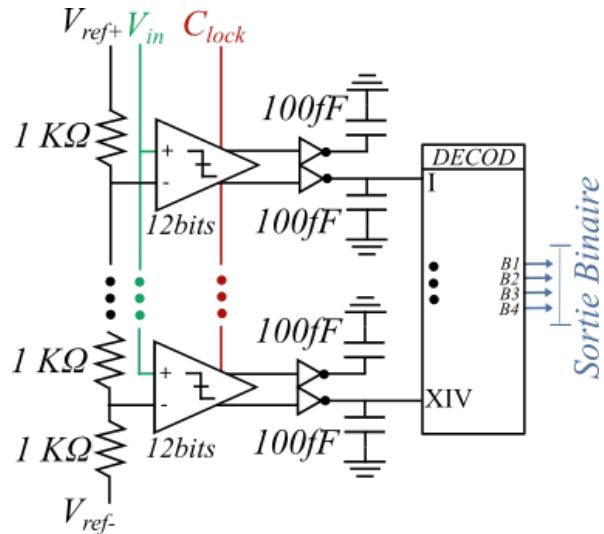


Figure 18: Assemblage réalisation pour la partie quantification de l'ADC

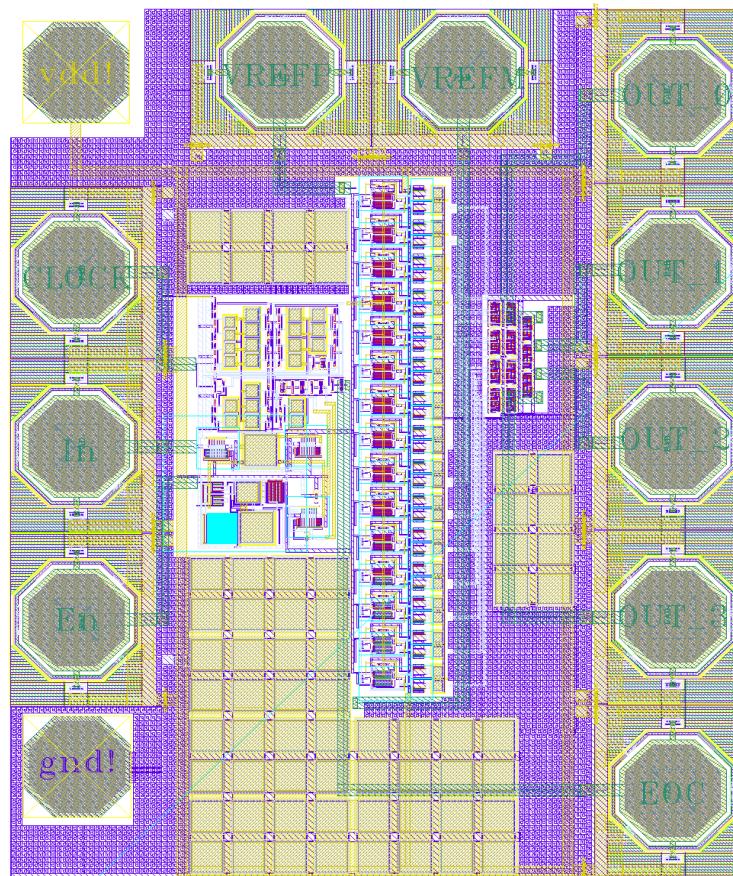


Figure 19: Layout global de l'ADC avec plan de masse et capacité de découplage

Comme dit précédemment la première étape a été le réaliser le layout de l'assemblage de l'ensemble des sous blocs de l'ADC et on y retrouve le résultat ci-dessous. Enfin, on remarque l'ajout de capacité de découplage sur l'ensemble du circuit. On retrouve en tout 66 pF de découplage afin de combler l'ensemble des espaces vides. Les espaces proviennent du fait que notre circuit est limité en taille par les plots qui permettent de relier notre puce au QFN 24. Pour finir, nous avons aussi ajouté un plan de masse sur l'ensemble de notre layout sur la première couche de métallisation. (Surface de 0.5mm²)

Dans un second temps, suite à l'extraction de tous les parasites, nous avons réalisé les simulations discutées précédemment. On remarque notamment ci-dessous la simulation pour un signal de type rampe en entrée. Cette simulation valide totalement le fonctionnement de notre ADC puisque que l'on observe aucune différence supérieure à 1 LSB entre la rampe et la sortie de notre convertisseur. Ceci nous permet donc de dire que notre ADC n'a pas de code manquant sur la dynamique d'utilisation. Par la suite nous réalisons l'autre simulation pour un signal de type sinusoïdal ce qui correspond à l'utilisation typique. On remarque de nouveaux résultats corrects. Si l'on traite ces résultats on obtient SNR De 21,01 dB, ce qui permet de dire que notre convertisseur à 3,19 bits effectif. Cette résolution est supérieure au 2 bits annoncé en introduction. En définitive notre ADC est donc validé pour notre application.

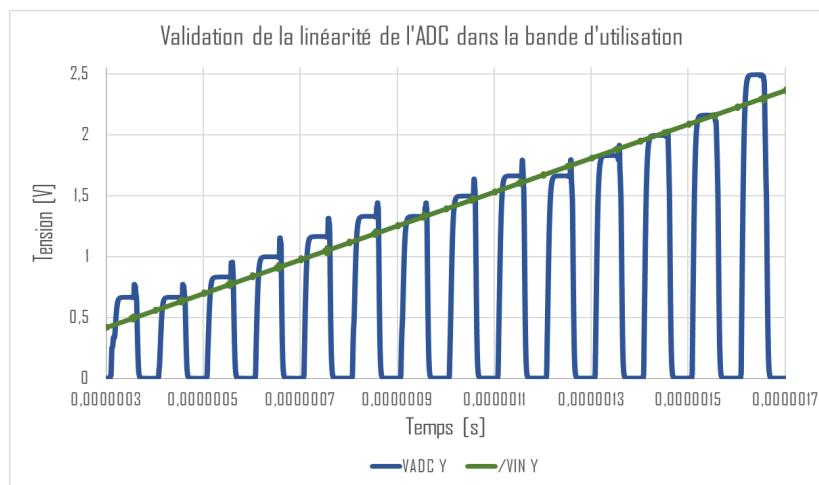


Figure 20: Simulation de l'ADC post-layout pour un signal rampe en entrée

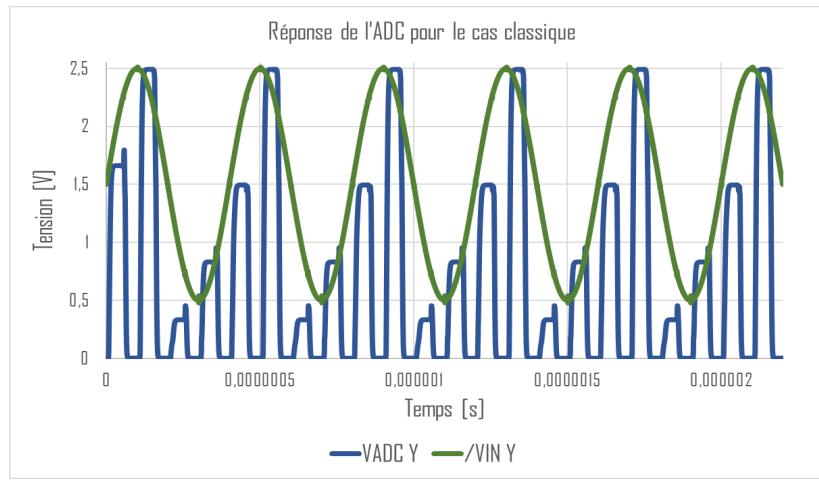


Figure 21: Simulation de l'ADC post-layout pour un signal sinusoïdale en entrée

Pour finir, on mesure une consommation moyenne de 7.1 mA après layout. Ce qui nous permet de calculer la figure de mérite de notre ADC à l'aide de cette formule :

$$FOM_{walden} = \frac{Power_{ENOB}}{Fs \cdot 2} \Rightarrow 32.5 \text{ pJ/Conv}$$

C'est ici que réside la seule différence majeure entre les simulations avec et sans considération des parasites. En effet, on remarque une différence de 5pJ/conv suite à la considération des parasites. Ceci s'explique par la baisse de la consommation, et légère des performances, à cause des polarisations suite à notre layout.

2 WP3 - Section numérique :

Pour chaque partie présenter :

1 / Spécification et implémentation RTL:

- a) Présenter le fonctionnement général du circuit (*Algorithme et découpage en blocs*), en justifiant notamment
 - a. le choix d'architecture
 - b. le dimensionnement des signaux et des coefficients.
- b) Le partitionnement en Partie Opérative et Partie Contrôle, ainsi que la FSM régissant cette dernière (*si applicable*)
- c) Le testbench, le plan de validation et son rapport avec les spécifications systèmes
- d) Résultats de simulations commentés et analysés

2/ Synthèse et Place&Route :

- a) Vérification du code avec Spyglass: erreurs et corrections
- b) Synthèse avec Design Vision :
 - a. Résultats commentés et analysés
 - b. Erreurs éventuelles et leur correction
 - c. Simulation post-synthèse
- c) Placement & Routage avec SoC Encounter : résultats commentés et analysés, erreurs éventuelles et simulation post-P&R

Attention, chaque livrable doit être clairement identifiable (numéro correspondant à liste des livrables). Voici une liste des livrables numériques attendus dans le projet initial.

2.1 Interfaces externes Rx/Tx (Axel Baldacchino, Tom Désesquelle)

2.1.1 Spécification et implémentation RTL

2.1.1.1 Contexte

Le protocole ZigBee est un protocole de communication entre une antenne d'émission et une antenne de réception. Pour envoyer des données via ce protocole, il faut fournir à la puce d'émission les données à transmettre. De la même façon, en réception, pour traiter les données reçues, il faut les stocker. Le but des interfaces externes est de faire le lien, respectivement, entre la chaîne de transmission et la chaîne de réception et le monde extérieur.

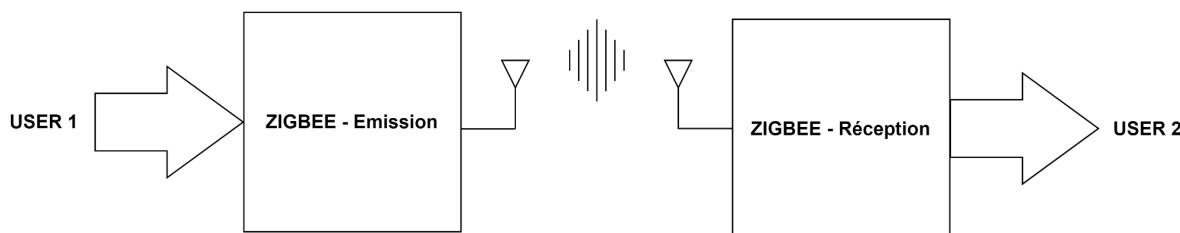


Figure 0 : Nécessité des interfaces extérieurs

L'interface externe en chaîne d'émission permet de faire le lien entre un CPU, auquel est connecté une mémoire RAM, et le CODER IQ, qui correspond au premier bloc de la chaîne d'émission. Nous savons que la transmission du protocole ZigBee se fait bit par bit, or le CPU envoie des données de taille variable en entrée de la chaîne.

Les interfaces externes se traduisent par un FIFO (First In First Out), qui prend la forme d'un tableau de registres. Cette FIFO va nous permettre de stocker les données sur bits reçues par le CPU en entrée, et ainsi les transmettre bit par bit au CODER IQ comme l'illustre la figure ci-dessous.

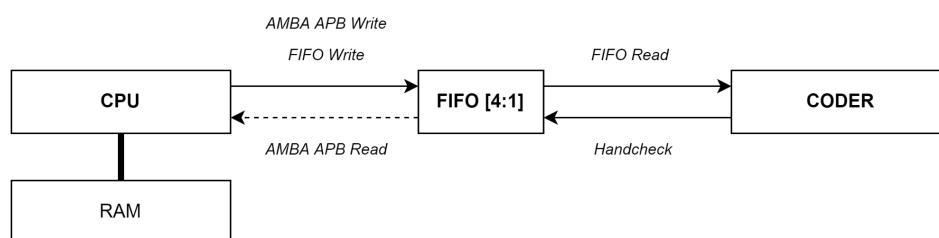


Figure 1 : Place de la FIFO dans la chaîne d'émission

Concernant la chaîne de réception, le but de l'interface extérieur (outFIFO) est de faire le lien entre le dernier bloc de la chaîne de réception, à savoir le CDR (Clock and Data Reset), et la RAM qui est connectée au CPU (Figure ci-dessous). En effet, en réception, le but est de stocker les données qui arrivent bit par bit, dans des registres, pour reconstruire la donnée sur 4 bits.

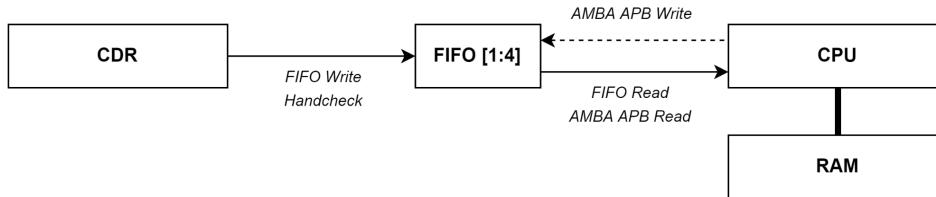


Figure 2 : Place de la FIFO dans la chaîne de réception

La description des interfaces externes va être décomposée en plusieurs parties. Dans un premier temps nous détaillerons le modèle que nous avons réalisé pour modéliser le fonctionnement du CPU. Nous verrons par la suite, le fonctionnement de la FIFO en détail. Enfin nous détaillerons la méthode de vérification que nous avons mise en place pour valider le fonctionnement de chaque bloc.

2.1.1.2 Emulation du CPU

Dans un premier temps, nous avons commencé par émuler le comportement d'un CPU. Pour ce faire, nous avons choisi le protocole AMBA APB de ARM, car il s'agit d'un protocole réputé simple et communément utilisé. Avec ce protocole il est à la fois possible d'écrire dans la mémoire à une adresse spécifique sur demande du CPU, ou de lire et de fournir sur le bus de sortie la donnée contenue dans la case mémoire pointée par l'adresse spécifiée.

Son fonctionnement peut être résumé à partir d'une machine à états (figure X). Lors d'une requête sur le signal PENABLE, la procédure d'écriture ou de lecture est déclenchée en fonction de la FSM et de l'état de certains signaux. La FSM commence par déterminer le type d'accès que le CPU va devoir faire sur sa RAM. Pour déterminer le type d'accès (lecture ou écriture), la FSM regarde le niveau du signal P_nRead_Write.

Dans le cas d'une lecture, la FSM va ensuite lire l'adresse présente sur le bus PADDR, puis aller lire la donnée stockée à l'adresse correspondante. Cette donnée va être finalement rendue disponible sur le bus PRDATA.

Dans le cas d'une écriture, la FSM va aussi lire l'adresse présente sur le bus PADDR, mais aussi récupérer la donnée à stocker présente sur le bus PWDATA. Cette donnée va être finalement inscrite dans la case mémoire du CPU correspondante à l'adresse disponible sur le bus ADDR.

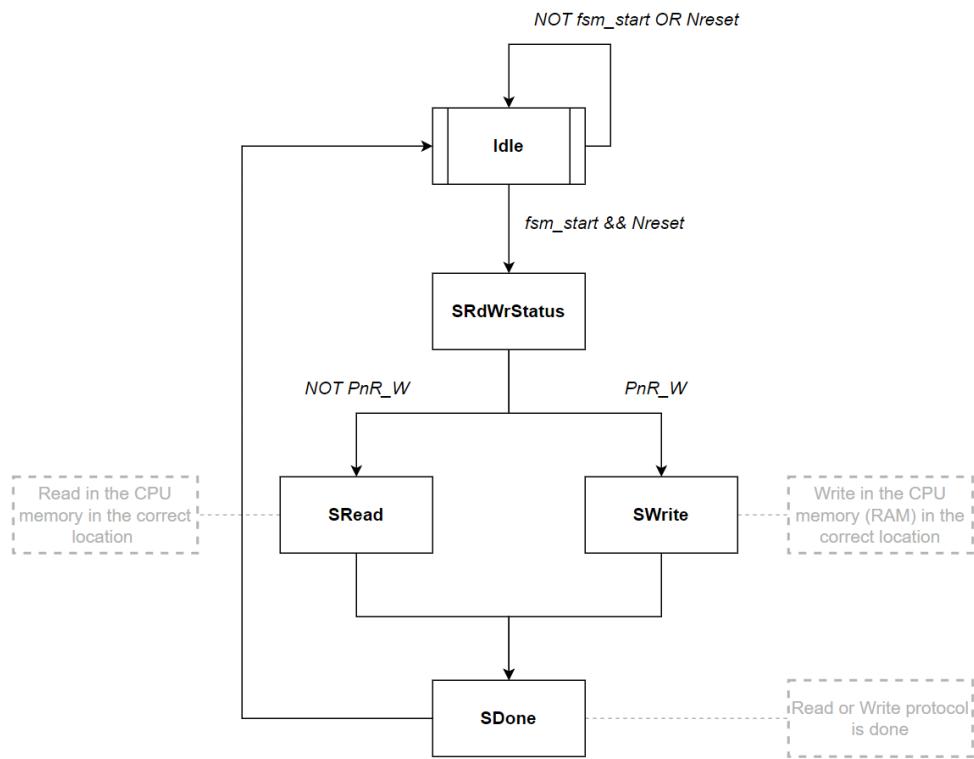


Figure 3 : Machine d'états du protocole AMBA APB

D'un point de vue architectural, la mise en œuvre du protocole est représentée en figure 3. Le déclenchement de la FSM par une requête de lecture ou d'écriture est réalisé par un One Shot. Le One Shot, ou monostable non redéclenchant, est un outil numérique qui permet de capturer le front montant et/ou descendant d'un signal. Ce One Shot permet donc de démarrer la FSM qui va alors pouvoir traiter les données d'entrées (P_nRead_Write & PADDR). Par un jeu de multiplexeurs et de démultiplexeurs, en fonction du type de requête reçue, les données vont être aiguillées vers la RAM du CPU ou vers l'utilisateur.

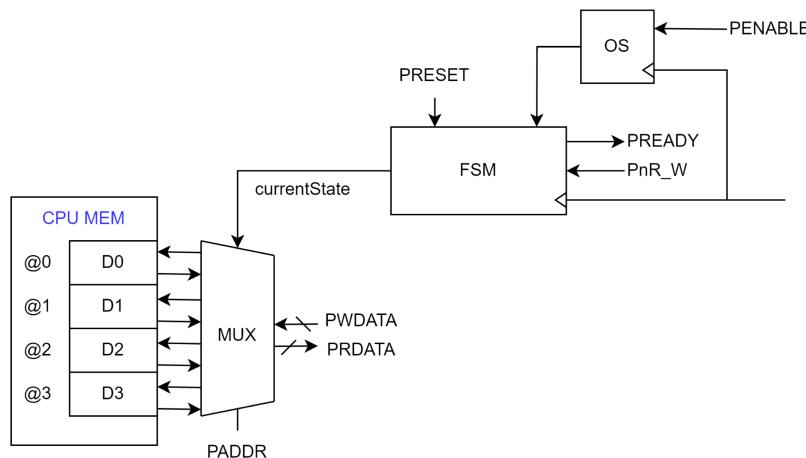


Figure 4 : Architecture du protocole AMBA APB

2.1.1.3 Réalisation des interfaces de transmission inFIFO et outFIFO

Dans un second temps, nous avons réalisé les interfaces de transmission inFIFO et outFIFO. Ces deux interfaces sont semblables en tout point. Ce qui les différencie est le nombre de bits reçus en entrée et le nombre de bits envoyés en sortie de la FIFO. Le principe de fonctionnement (figure 5) est donc le même pour ces deux interfaces.

La FSM est déclenchée dès lors qu'une requête d'écriture (inWriteEnable) ou une requête de lecture (inReadEnable) ait été reçue. La FSM procède à une gestion d'erreur en fonction de la requête reçue : pour envoyer une donnée, il ne faut pas que la FIFO soit vide; pour recevoir une donnée, il ne faut pas que la FIFO soit pleine. Si la procédure de gestion d'erreur est correcte, les données peuvent alors être stockées dans la FIFO ou envoyées au bloc suivant.

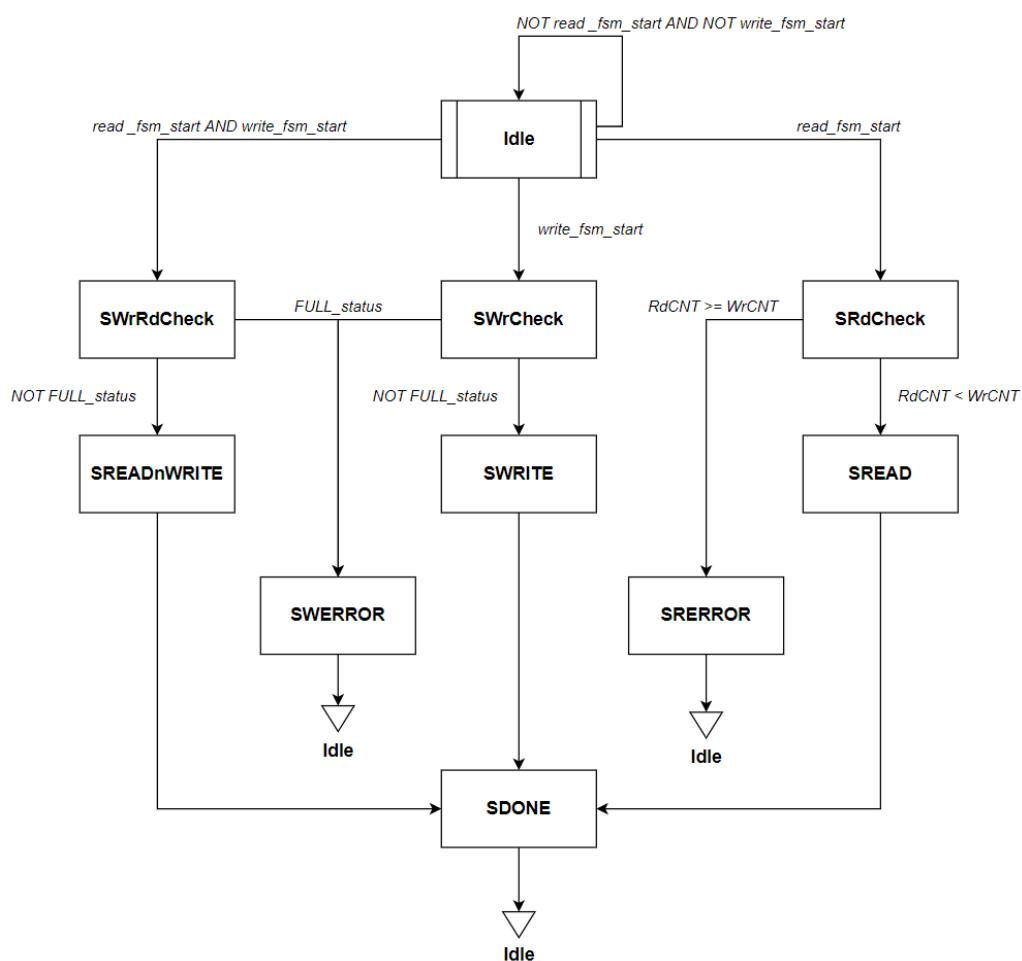


Figure 5 : Machine d'états des FIFOs

D'un point de vue architectural, deux monostables non déclenchantes permettent de lancer la machine d'états selon s'il s'agit d'une requête d'écriture ou d'une requête de lecture. La FSM incrémente, en fonction du front montant en entrée, le compteur de lecture ou le compteur d'écriture. Ses compteurs permettent de connaître l'espace disponible/occupé dans la FIFO. Ces compteurs permettent donc de gérer des registres de status. Le registre de status de la FIFO indique si la FIFO est vide, presque vide, presque pleine ou pleine. Le registre d'erreurs indique

si une requête ne peut pas avoir lieu : FIFO vide dans le cas d'une tentative de lecture ou FIFO pleine dans le cas d'une tentative d'écriture.

Lorsque les compteurs sont mis à jour, les pointeurs de lecture et d'écriture sont aussi déplacés. Les pointeurs de lecture et d'écriture sont respectivement déplacés dans le cas d'une lecture et d'une écriture. Le pointeur de lecture ne peut pas dépasser le pointeur d'écriture : si les deux pointeurs désignent la même case, la FIFO est alors vide.

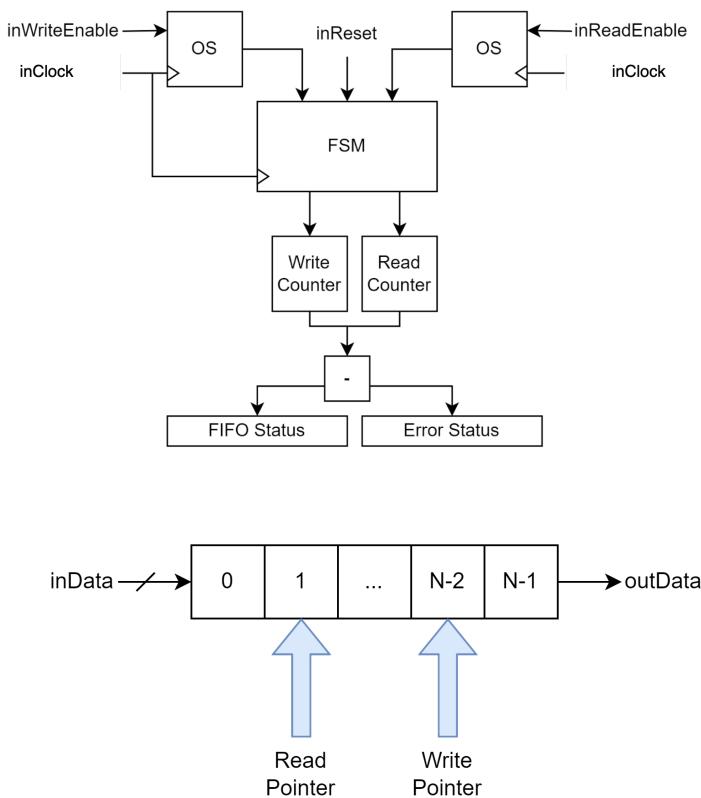


Figure 6 : Architecture des FIFOs

2.1.2 Testbench & plan de validation

2.1.2.1 Emulation du CPU

L'objectif de cette partie est de valider le fonctionnement du CPU modélisé avec le protocole AMBA APB. Pour ce faire, nous avons préchargé des données dans la RAM du CPU, représentée en vert sur la figure 7. Les points notés 1 décrivent l'amorce d'une requête de lecture de données depuis le CPU. Le signal P_nRead_Write est correctement interprété (lecture sur niveau bas) et l'adresse contenue dans PADDR est correctement décodée. La donnée souhaitée, représentée en rose, est donc correctement lue. Les points notés 2 décrivent l'amorce d'une requête d'écriture de données dans le CPU. Le signal P_nRead_Write est correctement interprété (écriture sur niveau haut) et l'adresse contenue dans PADDR est correctement décodée. La donnée souhaitée, représentée en orange, est donc correctement écrite à l'adresse souhaitée.

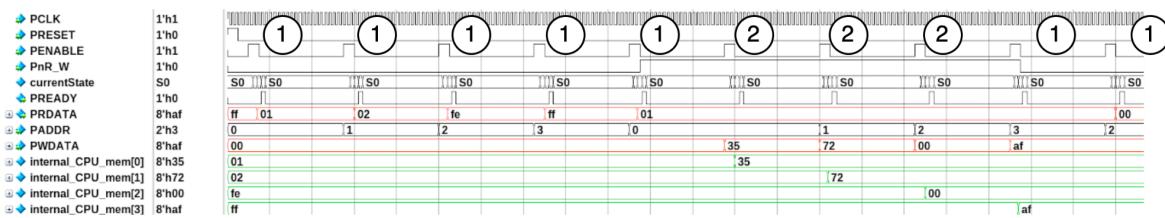


Figure 7 : Emulation du CPU avec le protocole AMBA APB de ARM

2.1.2.2 Différences entre inFIFO et outFIFO

Comme décrit précédemment, les interfaces de transmission et de réception partagent beaucoup de similitudes. La différence entre ces deux interfaces se situe de la taille des données reçues et de la taille des données envoyées. La FIFO de l'interface entre le CPU et le CODER, appelée inFIFO, reçoit 4 bits provenant du CPU et envoie ses données à raison d'un bit par un bit au CODER. La FIFO de l'interface entre le CDR et le CPU, appelée outFIFO, reçoit quant à elle 1 bit depuis le CDR et envoie 4 bits au CPU.

Les données de la inFIFO sont envoyées depuis le LSB jusqu'au MSB. Pour reconstruire ces données en réception, la outFIFO reçoit d'abord le MSB jusqu'au LSB.

2.1.2.2.1 Simulation de l'interface de transmission inFIFO

Pour ce qui est de la validation de l'interface de transmission inFIFO, nous avons écrit un *testbench* qui a pour but de reproduire le comportement attendu, mais aussi de faire face à des erreurs. Pour information, après réflexions sur le nombre d'entrées/sorties disponibles sur la puce et aussi sur la taille de la puce, nous avons fini par faire le choix d'implémenter une FIFO de 128 cases de 4 bits. Pour ce qui est de la simulation, nous avons utilisé le même code RTL en réduisant simplement la taille de la FIFO à 4 cases de 4 bits pour tester plus rapidement les cas limites (FIFO vide et FIFO pleine).

Les points 1 montrent que des données peuvent être inscrites dans la FIFO. Toutefois, lorsque la FIFO est pleine et qu'une requête d'écriture est reçue, la donnée n'est pas conservée et le registre d'erreur pour l'écriture est mis à jour (point 4). Les points 2 montrent que les données peuvent être lues bit à bit en commençant par le LSB. Si toutefois il n'y a plus de données disponibles dans la FIFO, le registre d'erreur pour la lecture est mis à jour (point 3).

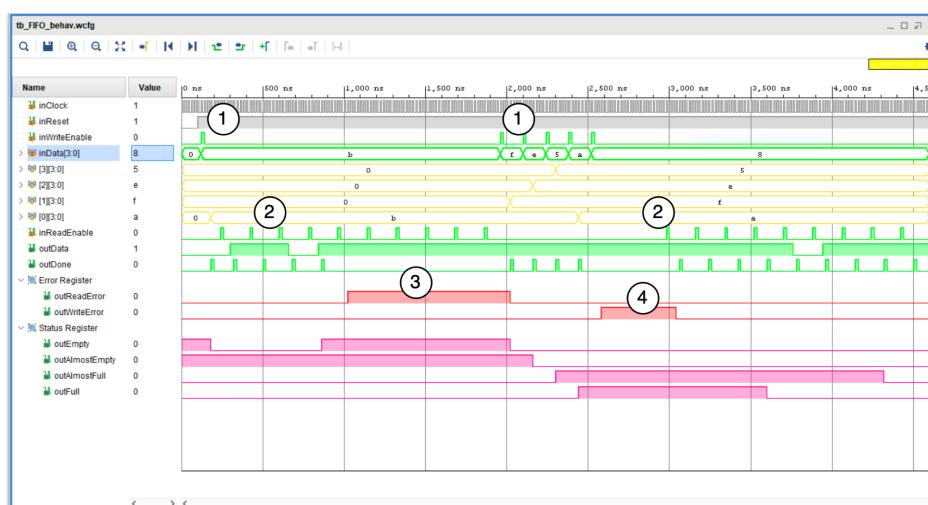


Figure 8 : Simulation de l'interface inFIFO

La figure 8 décrit le stockage de la donnée B (4 bits), puis la lecture de 8 bits ce qui engendre, pour les quatre dernières tentatives de lecture, une erreur de lecture. Ensuite, la tentative d'écriture de cinq données a lieu : la cinquième donnée n'est pas stockée et engendre une erreur d'écriture. Enfin, les données stockées sont lues pour montrer que la FIFO ne reste pas bloquée dans un état d'erreur lorsqu'une ou plusieurs requêtes valides ont lieu après des requêtes erronées.

2.1.2.2.2 Simulation de l'interface de réception outFIFO

De la même manière que pour l'interface de transmission, nous avons écrit un *testbench* qui a pour but de reproduire le comportement attendu de l'interface de réception outFIFO, mais aussi de faire face à des erreurs.

Les points 1 montrent que des données peuvent être inscrites dans la FIFO. Toutefois, lorsque la FIFO est pleine et qu'une requête d'écriture est reçue, la donnée n'est pas conservée et le registre d'erreur pour l'écriture est mis à jour (point 4). Les points 2 montrent que les données peuvent être lues bit à bit en commençant par le LSB. Si toutefois il n'y a plus de données disponibles dans la FIFO, le registre d'erreur pour la lecture est mis à jour (point 3).

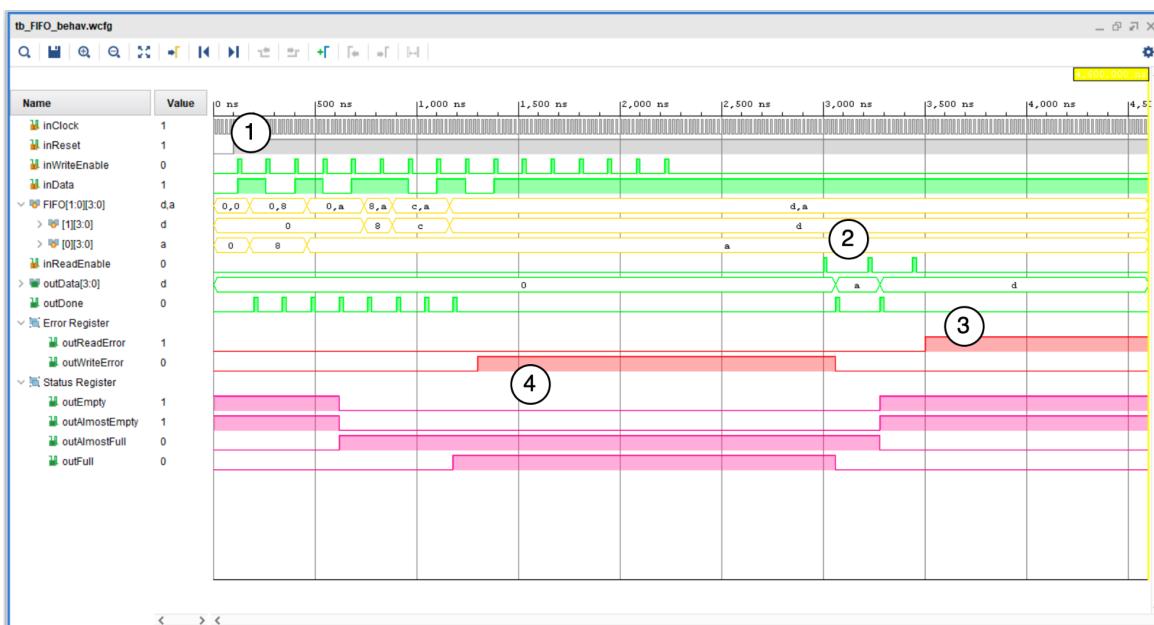


Figure 9 : Simulation de l'interface inFIFO

2.1.3 Synthèse

Lors de l'étape de la synthèse, nous avons eu affaire, dans un premier temps, à des latchs. En effet, nos listes de sensibilités ne comportaient pas que l'horloge. Une fois que nous avons résolu ce problème de listes de sensibilité, nous avons pu dérouler la synthèse correctement n'ayant plus que des bascules dans notre design. Pour ce qui concerne les simulations post-synthèse de nos deux designs, nous obtenons exactement les mêmes résultats de simulation après la synthèse (figure 10 et figure 11).

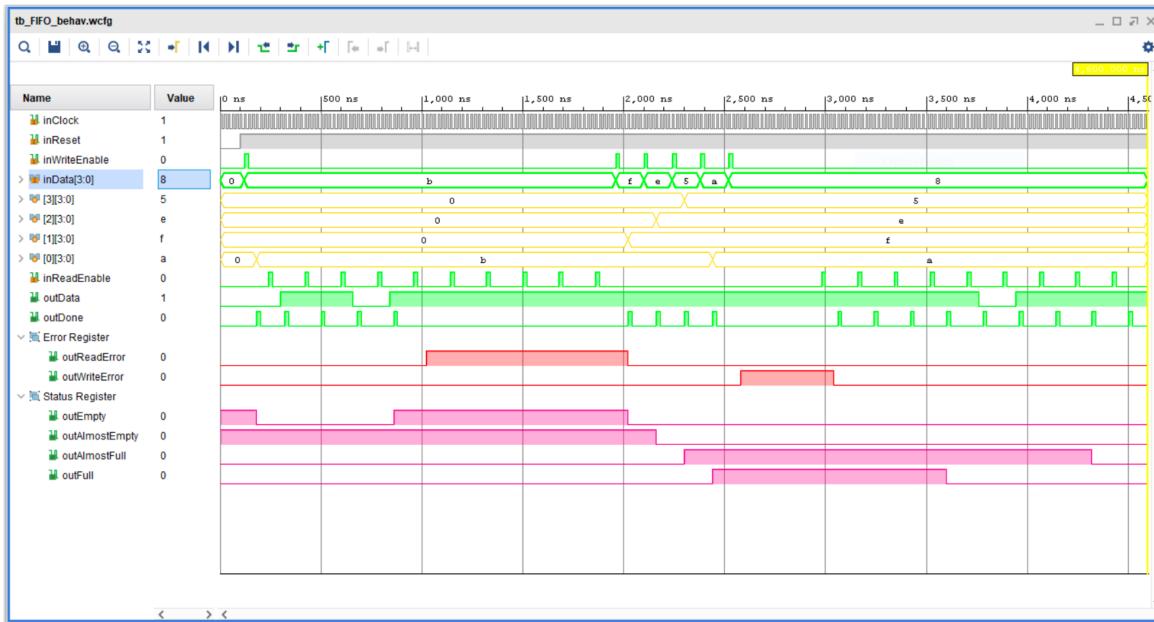


Figure 10 : Simulation post-synthèse de l'interface inFIFO

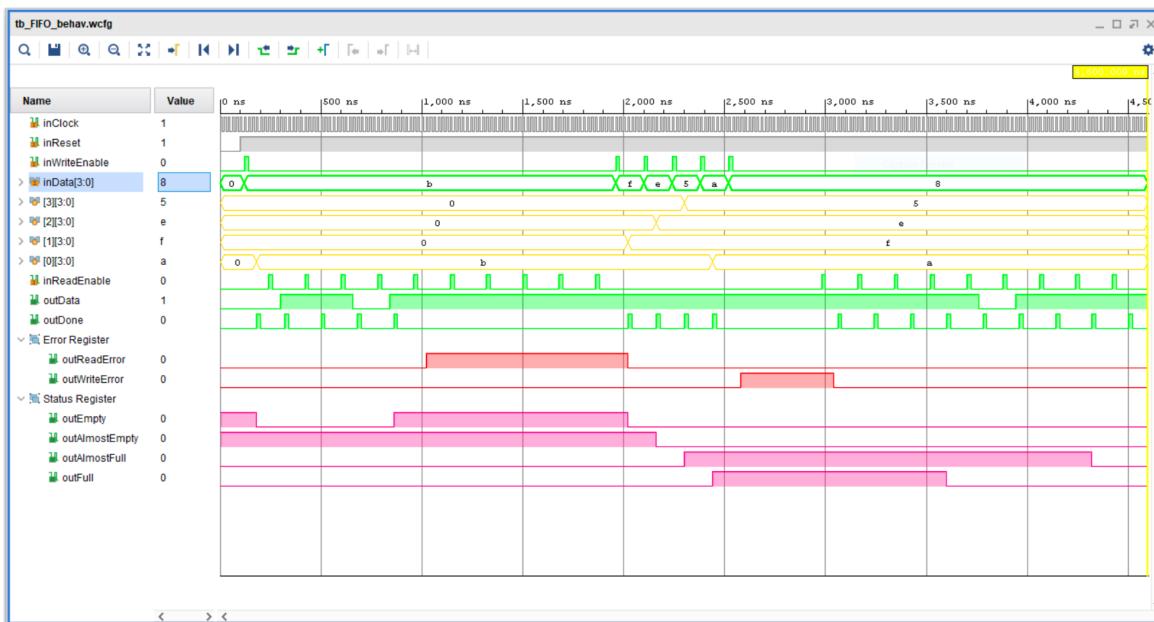


Figure 11 : Simulation post-synthèse de l'interface outFIFO

Nous avons ainsi pu générer les rapports de surface et de *timing* pour chacun de nos deux designs.

L'interface inFIFO utilise une place d'environ 0.3 mm^2 (figure 12) et présente un *slack* typique (figure 13) de 13.6 ns : c'est-à-dire que, pour notre période d'horloge de 20 ns , nos signaux se propagent suffisamment vite pour être traités avant le prochain coup d'horloge. En d'autres termes, nous disposons d'une marge de 13.6 nanosecondes entre le traitement de données dans notre design.

		Point	Incr	Path
Number of ports:	31			
Number of nets:	1398	clock inClock (rise edge)	0.00	0.00
Number of cells:	1335	clock network delay (ideal)	0.00	0.00
Number of combinational cells:	780	i_FIFO_reg[1]/C (DFE1)	0.00	0.00 r
Number of sequential cells:	555	i_FIFO_reg[1]/Q (DFE1)	2.03	2.03 r
Number of macros/black boxes:	0	U1095/Q (BUF2)	1.08	3.11 r
Number of buf/inv:	169	U740/Q (BUF2)	1.15	4.27 r
Number of references:	24	U598/Q (IMUX40)	0.63	4.90 f
Combinational area:	76075.999298	U484/Q (IMUX40)	0.40	5.29 r
Buf/Inv area:	7498.400032	U610/Q (IMUX40)	0.25	5.55 f
Noncombinational area:	181326.603333	U491/Q (IMUX21)	0.24	5.78 r
Macro/Black Box area:	0.000000	U628/Q (MUX41)	0.36	6.15 r
Net Interconnect area:	35703.000000	U1104/Q (NAND22)	0.07	6.22 f
Total cell area:	257402.602631	U1103/Q (INV3)	0.09	6.31 r
Total area:	293105.602631	sigOutData_reg/D (DFE1)	0.00	6.31 r
		data arrival time		6.31
		clock inClock (rise edge)	20.00	20.00
		clock network delay (ideal)	0.00	20.00
		sigOutData_reg/C (DFE1)	0.00	20.00 r
		library setup time	-0.12	19.88
		data required time		19.88
		data required time		19.88
		data arrival time		-6.31
		slack (MET)		13.58

Figure 12 et Figure 13: Rapport de surface et rapport de timing de l'interface inFIFO

L'interface outFIFO utilise, quant à elle, une place d'environ 0.35 mm^2 (figure 14) et présente un *slack* typique (figure 15) de 14.4 ns .

		Point	Incr	Path
Number of ports:	31	clock inClock (rise edge)	0.00	0.00
Number of nets:	1988	clock network delay (ideal)	0.00	0.00
Number of cells:	1926	sigWRCOUNT_reg[0]/C (DFE1)	0.00	0.00 r
Number of combinational cells:	1368	sigWRCOUNT_reg[0]/Q (DFE1)	0.91	0.91 f
Number of sequential cells:	558	U1761/Q (NOR21)	0.29	1.20 r
Number of macros/black boxes:	0	U1760/Q (INV3)	0.10	1.30 f
Number of buf/inv:	300	r98/U2_1/C0 (ADD32)	0.37	1.67 f
Number of references:	23	r98/U2_2/C0 (ADD32)	0.38	2.06 f
Combinational area:	115206.000137	r98/U2_3/C0 (ADD32)	0.38	2.44 f
Buf/Inv area:	14760.199814	r98/U2_4/C0 (ADD32)	0.38	2.82 f
Noncombinational area:	182309.403351	r98/U2_5/C0 (ADD32)	0.38	3.20 f
Macro/Black Box area:	0.000000	r98/U2_6/S (ADD32)	0.63	3.83 r
Net Interconnect area:	48654.000000	U1604/Q (NOR40)	0.43	4.26 f
Total cell area:	297515.403488	U1602/Q (INV3)	0.16	4.42 r
Total area:	346169.403488	U1600/Q (NOR40)	0.55	4.97 f
		U16/Q (OAI222)	0.25	5.22 r
		U1605/Q (AOI211)	0.20	5.42 f
		U12/Q (OAI212)	0.20	5.62 r
		currentState_reg[2]/D (DF3)	0.00	5.62 r
		data arrival time		5.62
		clock inClock (rise edge)	20.00	20.00
		clock network delay (ideal)	0.00	20.00
		currentState_reg[2]/C (DF3)	0.00	20.00 r
		library setup time	-0.01	19.99
		data required time		19.99
		data required time		19.99
		data arrival time		-5.62
		slack (MET)		14.37

Figure 14 et Figure 15: Rapport de surface et rapport de timing de l'interface outFIFO

2.2 Codeur (Anthony TEISSIER, Bruno BAPTISTE)

2.2.1 Spécification et implémentation RTL

a) Fonctionnement général

Le principe du codeur est de réceptionner les bits transmis par le bloc précédent, la FIFO, pour en déduire la forme des sinus numériques I et Q en sortie qui seront recombinés en un seul signal afin qu'il soit converti en analogique par le bloc suivant, le DAC.

Le bloc étudié dans cette partie est contenu dans l'émetteur RF, il est composé d'un codeur IQ et d'une mise en forme de sinus. Ainsi, nous devons recréer le fonctionnement d'une modulation MSK qui est le meilleur compromis en termes de linéarité et de facilité de réalisation, comparé à différents modèles relativement proches (QPSK, OQPSK). En effet, la modulation MSK permet de faire varier la phase de façon continue et sur une enveloppe constante ce qui évite de perdre de la puissance en non linéarité de l'ampli. Le fait qu'elle varie en phase permet de ne jamais passer par un module nul ce qui limite les interférences entre symboles.

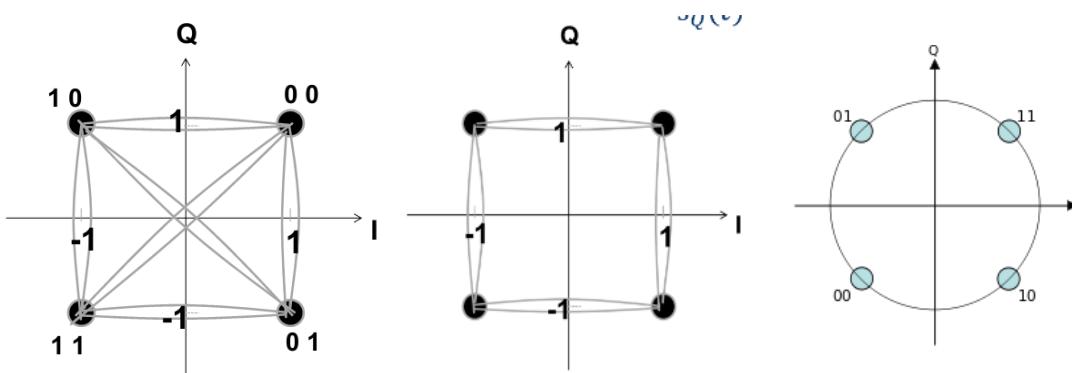


Figure 1 : Différents types de modulation : QPSK, OQPSK et MSK

Nous rappelons que lorsqu'un bit est transmis :

- Si $b(i) = 0 \Rightarrow S_I(i) \neq S_Q(i)$ et la phase diminue de $\pi/2$
- Si $b(i) = 1 \Rightarrow S_I(i) = S_Q(i)$ et la phase augmente de $\pi/2$

Avec S_I et S_Q les amplitudes des signaux I et Q.

Nous avons remarqué que les signaux I et Q ont leurs amplitudes qui peuvent s'inverser (ou non) une fois sur deux chacun leur tour. Nous avons donc réalisé que ces deux signaux devront agir exactement de la même manière mais en étant décalés d'une demi période d'arche de sinus ou plutôt d'un quart de période de sinus.

Une fois cela compris, il est facile d'associer le comportement des sinus en sortie du modulateur aux conditions sur les bits en entrée du codeur. En effet, en mémorisant l'état du bit précédent et en le comparant au bit actuel on arrive aux conditions suivantes :

- Si les deux bits sont différents (donc un changement de valeur du bit), alors un seul des sinus entre I et Q (celui qui est à cet instant égal à zéro) inversera son amplitude
- Si les deux bits sont égaux, alors aucun changement d'amplitude

Nous pouvons traduire ce fonctionnement par un algorigramme :

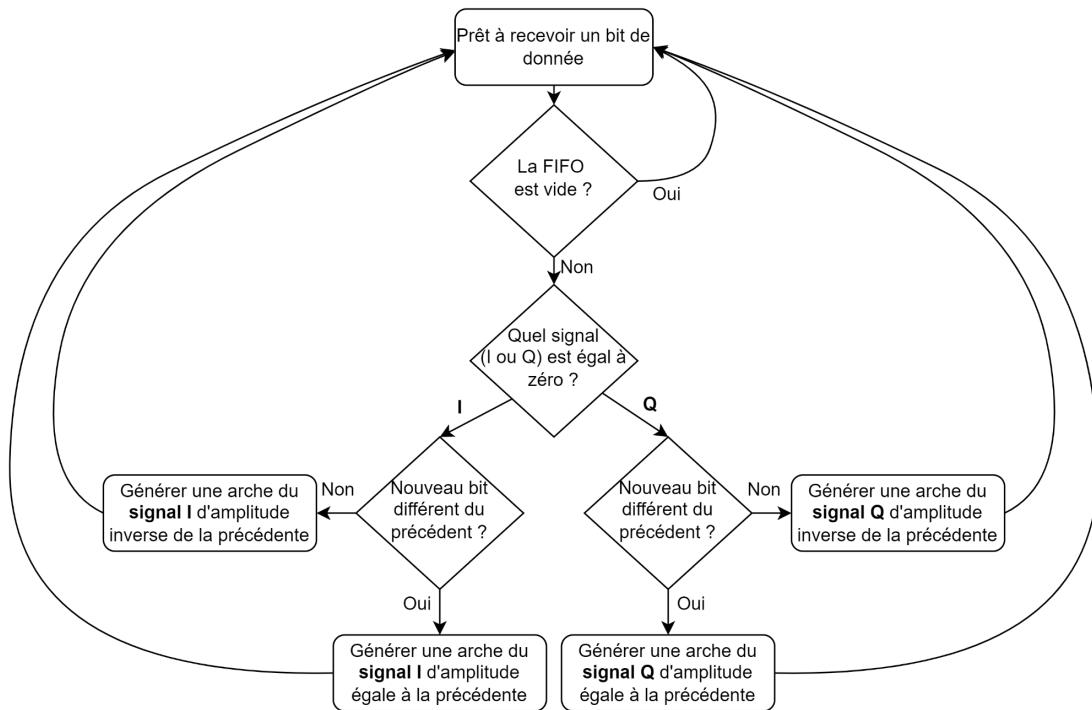


Figure 2 : Algorigramme du Codeur MSK

Pour la génération des deux signaux sinusoïdaux, nous avons choisi de placer en mémoire des tableaux de valeurs qui correspondent aux échantillons de sinus formant deux arches séparées, un positif et un négatif. Ces valeurs sont des décimales sur 4 bits entre -7 et 7 (avec le bit de poids fort pour le signe). Ces arches seront ensuite générées point par point par incrémentation d'un compteur de 10 échantillons (nous expliquerons plus tard ce choix de nombre d'échantillons).

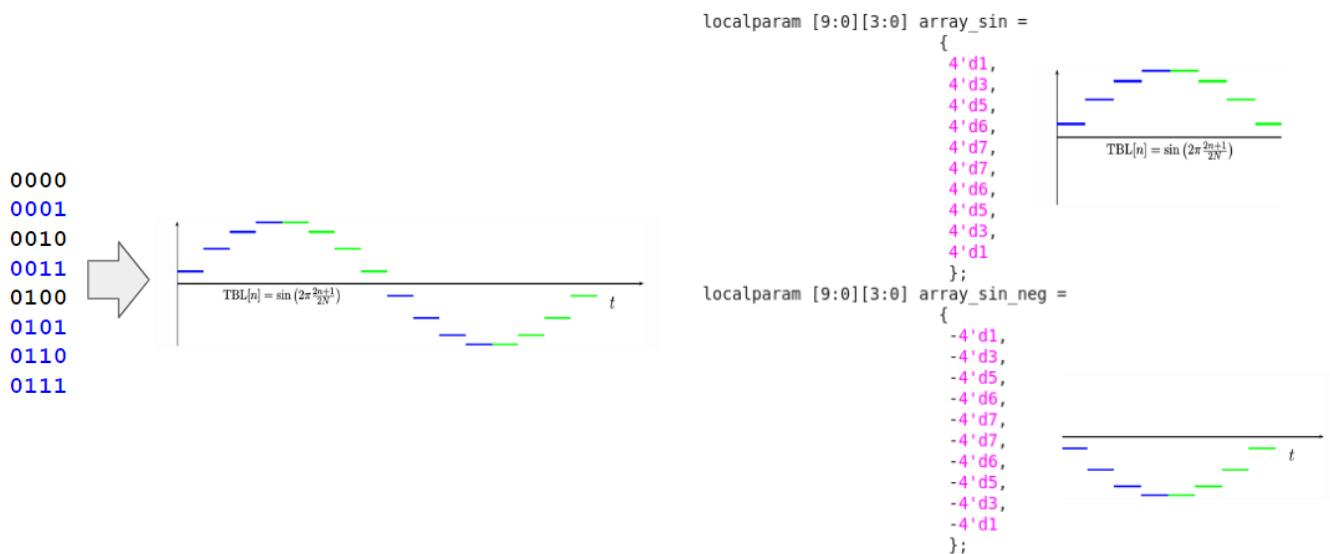


Figure 3 : Représentation des sinus avec tableau de valeurs décimales

Cette façon de découper nos sinus par arche (demi-période), nous permet d'être libre sur la

construction du sinus. En effet, on enchaînera une arche positive suivie d'une arche négative (et ainsi de suite) pour faire apparaître un sinus de base. Mais dès qu'un changement de valeur de bit d'entrée est détecté, alors une arche de même signe que la précédente arche sera générée pour traduire ce changement d'amplitude.

Le cahier des charges nous impose un débit binaire de 2 Mbps. Sachant cela, nous devons définir la fréquence de nos deux signaux I et Q. D'après le fonctionnement de la modulation MSK vu en cours, nous savons qu'il faut une demi-période de sinus pour transmettre deux bits de données.

$$\text{Soit } T/2 = 2 \text{ Tb} \Leftrightarrow T = 4\text{Tb} \Leftrightarrow F_{\text{SIN}} = Db/4 = 2*10^6/4 = 500 \text{ kHz}$$

Nous aurons donc deux sinusoïdes à une fréquence de 500 kHz.

Ensuite, nous savons que le CAN du côté récepteur fonctionne à une fréquence d'échantillonnage de 10 MHz. Nous décidons donc de fixer également cette fréquence comme repère. Nos signaux I et Q étant à une fréquence de 500 kHz, pour pouvoir être en adéquation avec la fréquence d'échantillonnage du CAN, il nous faudra $10 \text{ MHz}/500 \text{ kHz} = 20$ échantillons par période de sinus, soit 10 par arche de sinus.

b) FSM - Partie opérative et partie commande

Étant donné que le codeur reçoit directement des données de la FIFO, il y a un besoin de signaux d'interaction. Nous avons exigé le moins de signaux d'interaction possible entre ces deux blocs. En effet, d'après le fonctionnement choisi du codeur, il suffit juste de signaler à la FIFO que l'on est prêt à recevoir une donnée. A ce moment-là, et si la FIFO n'est pas vide, elle met à disposition sur sa sortie un nouveau bit de données. Nous pourrons alors le lire au prochain front d'horloge. C'est à cet instant que le codeur compare ce nouveau bit à l'ancien pour décider du signe du prochain arche et entamer sa construction. Nous avons donc distingué uniquement deux états de FSM. Un état pour signaler que le codeur est prêt à recevoir, et un état où le bit est reçu. Et nous aurons également deux signaux d'interaction (ready et empty) entre la FIFO et le codeur, hormis la donnée elle-même.

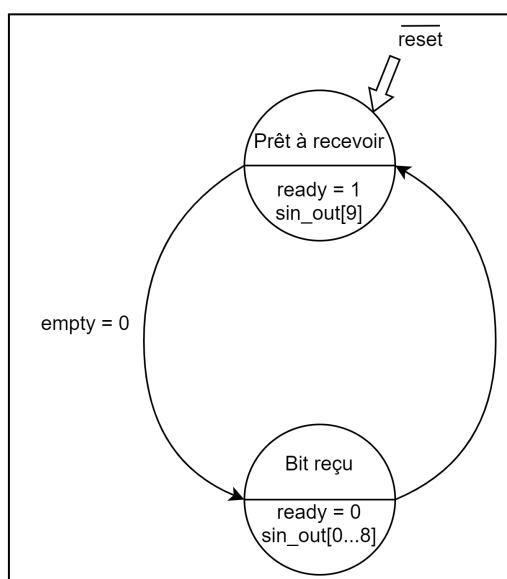


Figure 4 : FSM du codeur MSK

Puisque nous avions remarqué que les deux signaux I et Q ont exactement le même comportement mais en étant seulement décalés d'un quart de période de sinus, nous avons fait le choix d'implémenter deux FSM identiques. La FSM illustré sur la figure 9 est donc en deux exemplaires (une pour I et une pour Q).

Comme on peut le voir sur ce schéma, la dernière valeur du tableau est générée lorsque l'on se trouve dans l'état "prêt à recevoir". Cela est la conséquence du fait que les valeurs décimales du tableau doivent se suivre sans interruption pour générer un sinus en continu. La seule condition qui peut interrompre la génération du sinus est de déclencher un reset.

En ce qui concerne la partie opérative et la partie commande, elle peuvent se distinguer simplement par les actions suivantes :

PARTIE OPÉRATIVE	PARTIE COMMANDE
<ul style="list-style-type: none"> - Comparaison de l'ancien bit avec le nouveau bit reçu - Détermination du sens de l'arche à générer en fonction du précédent et du changement de valeur du bit reçu 	<ul style="list-style-type: none"> - Envoi d'un signal "prêt" (ready) - Génération des échantillons de sinus

c) Testbench et validation des résultats

Afin de valider la simulation de ce bloc, nous décidons de la comparer au chronogramme présenté dans le cours de techniques de transmission de l'information. Nous construisons le testbench de façon à respecter la même suite de bits de données du cours.

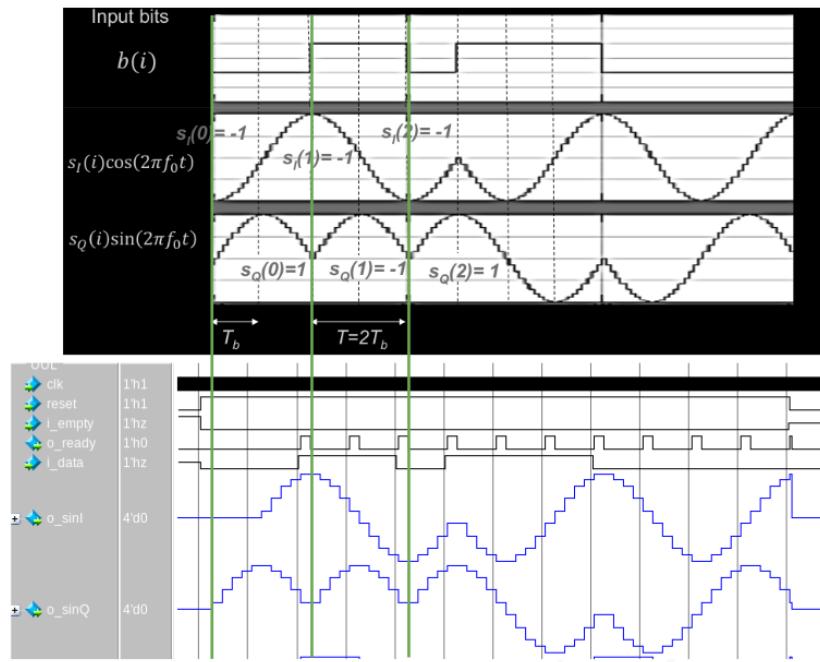


Figure 5 : Comparaison entre la simulation du codeur et le cours

Nous constatons que le résultat de simulation est strictement identique au cours puisque les chronogrammes des sorties I et Q sont identiques. D'autres tests doivent s'ajouter à celui-ci pour être sûr du bon fonctionnement du codeur. Nous avons donc simulé un appel au reset pendant la transmission des données, ce qui a eu pour effet d'arrêter la génération du sinus comme nous le souhaitions, et de reprendre la transmission une fois le reset désactivé.

D'autre part, pour des raisons de bon fonctionnement des tests inter-blocs (notamment avec le décodeur), il ne faut rien transmettre lorsque le reset est activé mais aussi quand la FIFO est vide. Pour cela, nous faisons en sorte que les signaux de sorties soient strictement nuls tant que nous recevons le signal "vide" de la FIFO. Ces options ont pu être correctement vérifiées grâce à notre testbench, et la construction de notre description RTL fait en sorte de terminer la transmission du dernier bit même si nous avons reçu le signal "empty" à 1 précisant que la FIFO est vide.

2.2.2 Synthèse et Place & Route

a) Vérification du code avec Spyglass : erreurs et corrections

Les premières synthèses de notre description RTL avec Spyglass nous ont tout d'abord apporté des erreurs liées à la construction de notre FSM. En effet, nous avions mal utilisé les boucles *always* puisque nous n'avions pas séparé la boucle contenant le *case* pour les changements d'état et la boucle des assignations de signaux.

Une fois cette erreur corrigée, nous sommes passés directement sur Design Vision afin de faire une synthèse plus précise puisque Spyglass ne reconnaît pas certaines erreurs de description RTL.

b) Synthèse avec Design Vision

Erreurs et leurs corrections

Etant donné que nous avons décidé de générer les signaux I et Q à une fréquence d'échantillonnage de 10 MHz alors que nous avons une fréquence de travail de 50 MHz, il nous a semblé nécessaire de devoir créer une pseudo horloge à 10 MHz en plus de 50 MHz. Cependant, cela nous a posé quelques soucis. Une des erreurs de notre part fût de placer cette pseudo horloge dans la liste de sensibilité d'une boucle always. En effet, cette pseudo horloge étant construite à partir de l'horloge de base, il est impossible de la considérer comme une vraie *clock*.

Nous avons pu corriger cette erreur en ajoutant une variable intermédiaire qui sera testé par une condition *if* à chaque front de l'horloge.

Résultats et analyses

Worst conditions

<i>clock clk (rise edge)</i>	20.00	20.00
<i>clock network delay (ideal)</i>	0.00	20.00
<i>j_reg[19]/C (DFE1)</i>	0.00	20.00 r
<i>library setup time</i>	-0.20	19.80
<i>data required time</i>		19.80

<i>data required time</i>		19.80
<i>data arrival time</i>		-14.50

<i>slack (MET)</i>		5.30

Best conditions

<i>clock clk (rise edge)</i>	20.00	20.00
<i>clock network delay (ideal)</i>	0.00	20.00
<i>j_reg[19]/C (DFE1)</i>	0.00	20.00 r
<i>library setup time</i>	-0.15	19.85
<i>data required time</i>		19.85

<i>data required time</i>		19.85
<i>data arrival time</i>		-4.43

<i>slack (MET)</i>		15.42

Typical conditions

<i>clock clk (rise edge)</i>	20.00	20.00
<i>clock network delay (ideal)</i>	0.00	20.00
<i>j_reg[19]/C (DFE1)</i>	0.00	20.00 r
<i>library setup time</i>	-0.17	19.83
<i>data required time</i>		19.83

<i>data required time</i>		19.83
<i>data arrival time</i>		-8.23

<i>slack (MET)</i>		11.61

Figure 6 : Report timing Design Vision

Simulation post-synthèse

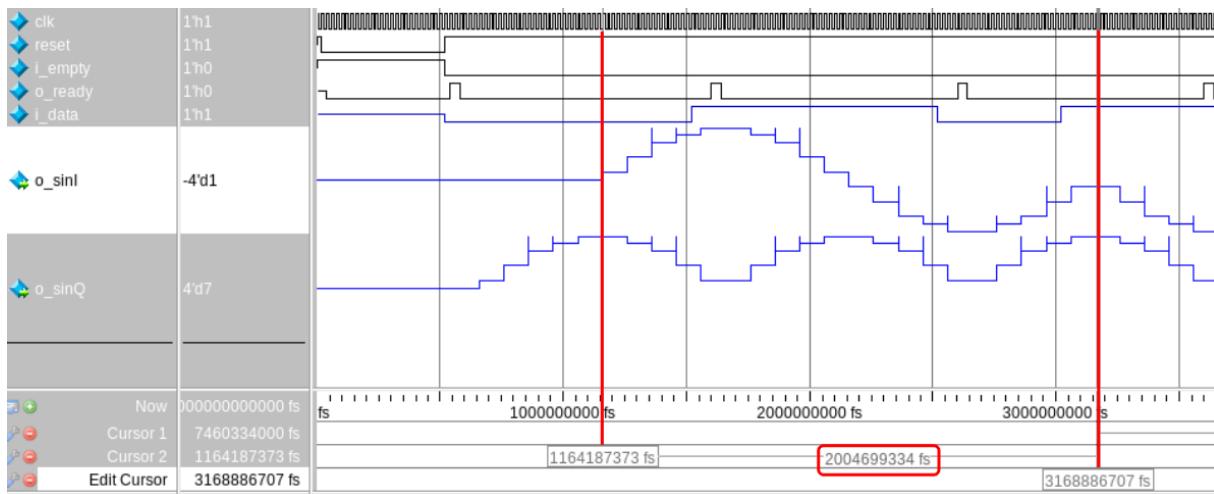


Figure 7 : Simulation post-synthèse sur Modelsim

Nous avons toujours le bon comportement du circuit après la synthèse. D'après le double curseur sur la figure on a une période de sinus de 2×10^9 fs = 2 µs, ce qui correspond bien à une fréquence de 500 kHz que nous souhaitons.

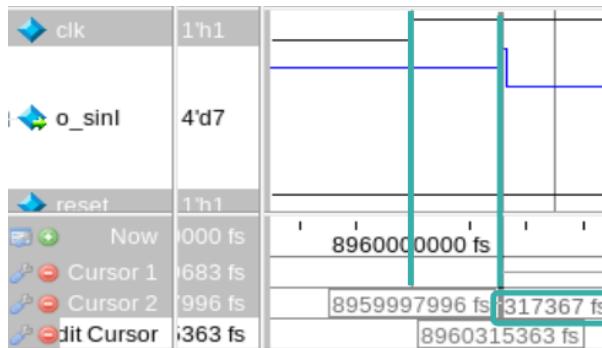


Figure 8 : Zoom sur front d'horloge

On peut observer des légers pics correspondant à de la métastabilité, et un léger décalage entre le front d'horloge et le changement de valeur du sinus en sortie, ce qui nous confirme bien que l'on a procédé à la simulation post-synthèse.

c) Placement & Routage

Le Codeur étant le premier bloc terminé, c'est sur celui-ci que nous avons commencé à réaliser les premières parties du flow. Pour nous aider dans cette tâche, nous nous sommes inspirés du TP filtre déjà réalisé au cours de l'année. En effet, celui-ci étant réalisé sur la même technologie, il nous a permis d'obtenir une base pour l'organisation de notre flow.

Pour ce qui est de la réalisation de la partie place & route, celle-ci se déroule sur l'outil Innovus de la gamme CADENCE, les erreurs de dessin seront analysées avec l'outil Calibre.

Nous avons alors récupéré les scripts de base du TP filtre qui nous ont permis de débuter comme le init.tcl qui initialise les paramètres nécessaires pour commencer à réaliser la puce sur Innovus. Ce fichier comprend le setting des variables qui seront utilisées plus tard dans les scripts du flow, des fonctions pratiques, l'initialisation des fichiers verilog tels que la netlist et le

fichier verilog des IOs qui fait la liaison entre les pads d'entrées/sorties et les signaux de la netlist. Il y'a aussi les chemins d'accès vers les fichiers .LEF de la technologie que nous utilisons.

Nous pouvons ensuite donner un nom à notre module comme msk_modulator_io par exemple. Le fichier init.tcl contient aussi les noms des nets d'alimentation et des clocks.

Une fois ce fichier sourcé, nous pouvons dire à Innovus de créer le design avec la commande init_design.

L'objectif de notre travail a vraiment était de préparer le terrain en prévision de la netlist du top de la puce. C'est pourquoi nous avons commencé par créer un espace de travail avec des répertoires particuliers en fonction de la nature des fichiers que nous mettrons à l'intérieur. Nous avons alors choisi l'organisation suivante.

WORK : Pour les lancements d'Innovus, cela permet de ne pas avoir des logs d'Innovus partout dans son espace de travail

SCRIPTS : Pour tous les scripts utilisés (init.tcl, design_config.tcl, clock_tree_synthesis.tcl, ...)

INPUT_DATA : Pour la netlist, les contraintes liées à la clock max capacitance, max transition etc..

CONSTRAINTS : Pour la génération de la clock + les pads IOs

Le design créé, nous pouvons créer le premier floorplan, celui-ci a une densité de 70% avec des écarts entre les pads de 80 µm. Nous avons aussi réalisé un script qui génère la grille d'alimentation de manière automatique en fonction de la taille du core. Voici le résultat.

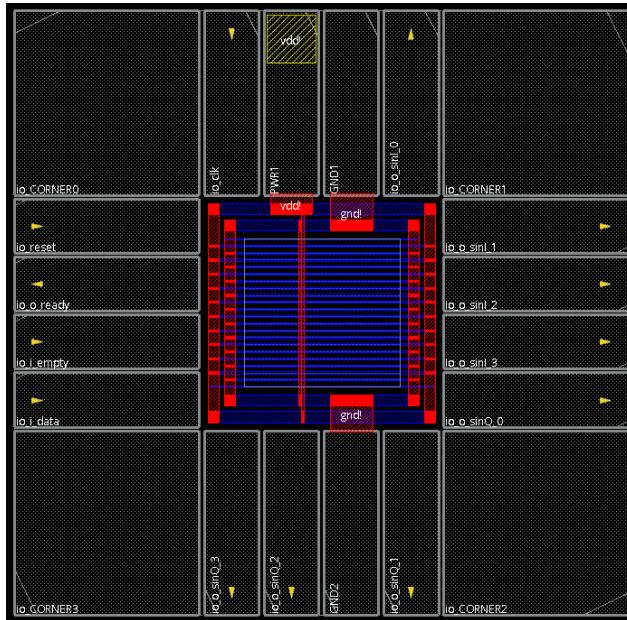


Figure 9 : 1er Floorplan du modulateur MSK avec grille d'alimentation

Nous reconnaissions bien les entrées/sorties décrites dans notre netlist en plus des alimentations, deux GND et un VDD

Il faut savoir que lors de cette étape, nous avons observé des erreurs de librairies avec des fall et des rise transition qui se trouvaient à 0.0. Nous avons alors cherché à régler l'erreur pendant environ une semaine avec l'aide des ingénieurs du CIME et du CMP mais au final l'erreur semble y être depuis toujours et les circuits fonctionnent tout de même. Malgré une perte de temps conséquente, nous avons alors repris le travail pour avancer le flow.

Après avoir fait ceci, nous pouvons passer à l'étape suivante, le placement des standard cells.

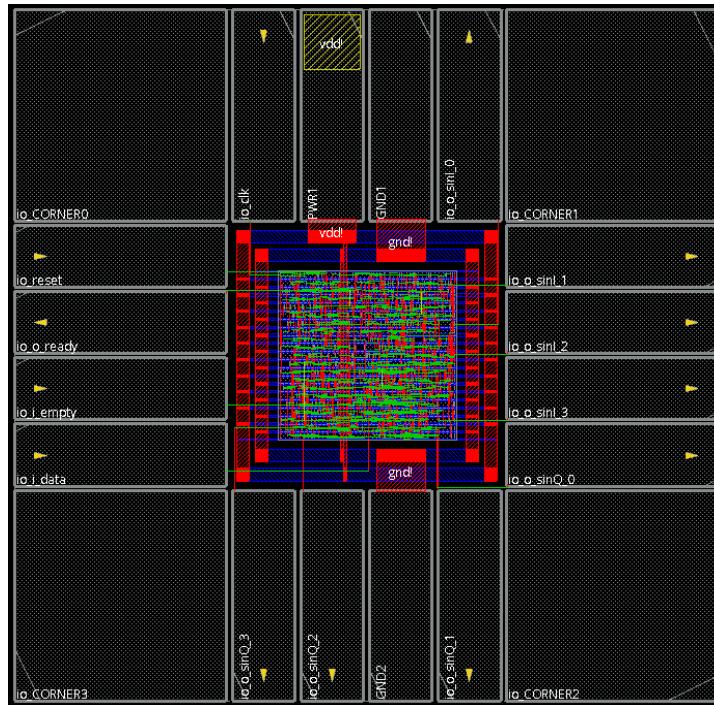


Figure 10 : Placement des standard cells

L'étape suivante est la génération de l'arbre d'horloge, celle-ci faite, nous avons ensuite voulu générer un fichier GDSII afin de regarder les erreurs de DRC.

Le fichier GDS s'ouvre à l'intérieur de virtuoso sous la forme d'un layout. Une fois brairie créée, nous pouvons observer ceci :

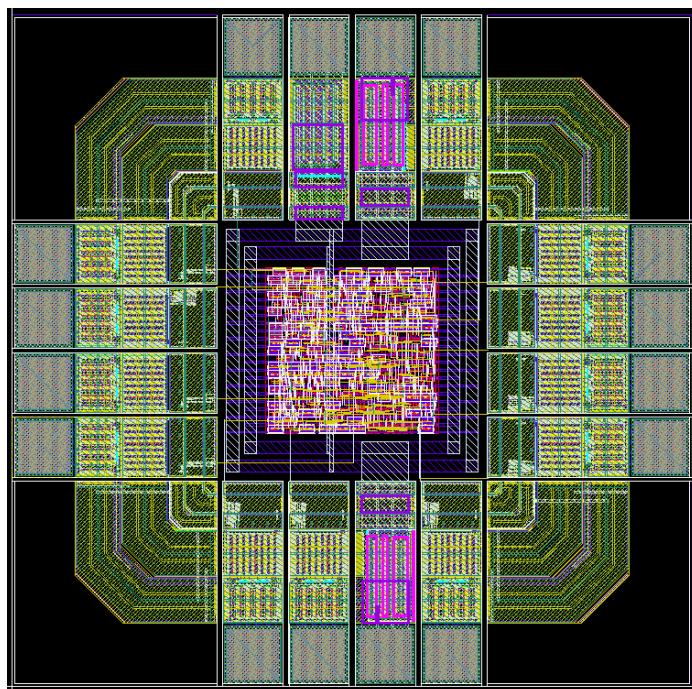


Figure 11 : Vue de la die dans Virtuoso

Après avoir lancé un premier run avec l'outil Calibre, nous pouvons observer nos premières erreurs de DRC

Check / Cell	
⊕ ✗ Check INFO_PROCESS_C35B4C0	✗ Check ILL_MET2_NOHOLES_AMW0
⊕ ✗ Check INFO_QRC_MODE_OFF	✗ Check LENGTH_M2SL0_AML1
⊕ ✗ Check INFO_RMETX_DEVICE_SHORTED	✗ Check ILL_MET1_NOT_VIA_CONT_ERC
⊕ ✗ Check INFO_RMETX_VIAx_DEVICE_SHORTED	✗ Check INFO_SUBDEF
⊕ ✗ Check INFO_METRES_VIA1_UNUSED	✗ Check SPAC_SUBDEFAREA_CONN
⊕ ✗ Check INFO_METRES_VIA2_UNUSED	✗ Check ILL_PGATE_BULK_CONN_BONDPAD_ERC
⊕ ✗ Check INFO_NWELL_HOT	✗ Check ILL_NWELL_NOT_CONN_VDD_SD_ERC
⊕ ✗ Check WIDTH_NTUBHOT_NWW2	✗ Check ILL_NWELL_RDIFFF_NOVDD_ERC
⊕ ✗ Check SPAC_NDIFF_NTUBHOT_ODC3	✗ Check ILL_NWELL_RPOLY1_NOSUPPLY_ERC
⊕ ✗ Check ILL_POLY1_DIE_RATIO_POR1	✗ Check INFO_BOND_PAD
⊕ ✗ Check SPAC_MET1_M1S1	✗ Check ILL_FLOATING_NET_ERC
⊕ ✗ Check SPAC_MET1_NOTCH_M1S1	✗ Check SHORTED_PADS_EQ2_SERC
⊕ ✗ Check SPAC_MET2_M2S1	✗ Check INFO_APERTURE
⊕ ✗ Check SPAC_MET2_NOTCH_M2S1	✗ Check INFO_PAD_4ST_STANDARD
⊕ ✗ Check ILL_MET2_DIE_RATIO_M2R1	✓ Check DENSITY_PRINT_FILES
⊕ ✗ Check SPAC_MET3_M3S1	✓ Check NET_AREA_RATIO_PRINT_FILES
⊕ ✗ Check SPAC_MET3WIDE_M3S2	✓ Check LAYOUT_INPUT_EXCEPTION_RDBS

Figure 12 : 1ère liste d'erreur de DRC

Seulement, nous nous sommes heurtés à un premier problème. Nous avions besoin d'une référence pour savoir quelles étaient les règles de dessin à respecter. Nous nous sommes alors rendu compte que Calibre nous renvoyait à la documentation ENG-256. Nous avons alors voulu chercher cette documentation et avons demandé aux ingénieurs du CIME pour savoir s'ils étaient en possession de cette documentation ce à quoi ils nous ont répondu qu'ils avaient accès à un DRM (Design Rules Manual) de la technologie C35 - 0,35µm CMOS. Le seul problème était que la documentation ENG-256, alors après quelques jours d'échange avec les ingénieurs du CIME et du

CMP qui nous ont confirmés que cette documentation n'existe pas, nous avons décidé de nous servir de la documentation ENG-183_rev10 correspondant au 0,35 µm C35 CMOS Design Rules.

PROCESS:	C35 - 0.35µm CMOS
PROCESS PARAMETERS	ENG-182_rev8 , 15.01.2014 0.35 µm C35 CMOS Process Parameters ENG-248_rev2 , 31.01.2007 0.35 µm C35O CMOS Opto Process Parameters ENG-470_rev1 , 12.03.2013 0.35 µm C35O CMOS Opto (BARC) Photodiode Process Parameters ENG-327_rev5 , 27.05.2014 0.35 µm C35 30V CMOS Module Process Parameters
DESIGN RULES	ENG-183_rev10 , 23.03.2017 0.35 µm C35 CMOS Design Rules ENG-325_rev4 , 17.08.2017 0.35 µm C35O CMOS Opto Design Rules ENG-485_rev6 , 27.09.2017 0.35 µm C35O CMOS Opto Filter Design Rules ENG-326_rev7 , 19.03.2014 0.35 µm C35 30V CMOS Module Design Rules ENG-365_rev2 , 27.11.2009 0.35 µm C35 CMOS Schottky Barrier Diode Module Design Rules ENG-410_rev2 , 12.01.2012 0.35 µm Pad over Active Design Rules ENG-562_rev2 , 13.02.2017 0.35 µm WLCSP / 3D-WLCSP Design Rules
ESD DESIGN RULES	ENG-236_rev3 , 01.10.2012 IV ESD Design Rules
SPICE MODELS	ENG-472_rev2 , 21.05.2014 0.35 µm C35 CMOS SPICE Models
RF SPICE MODELS	ENG-188_rev5 , 03.11.2005 0.35 µm C35 CMOS RF SPICE Models
NOISE PARAMETERS	ENG-189_rev7 , 20.05.2014 0.35 µm C35 CMOS Noise Parameters
MATCHING PARAMETERS	ENG-228_rev4 , 12.03.2013 0.35 µm C35 CMOS Matching Parameters ENG-339_rev1 , 30.04.2009 0.35 µm C35 30V CMOS Matching Parameters
DFM RULES	ENG-301_rev8 , 18.04.2018 0.35 µm CMOS C35 DFM Rules

Figure 13 : HTML du DRM de la technologie C35 - 0,35µm CMOS

En effet, celle-ci répertorie les largeurs minimales des pistes par layer, les spacing nécessaires, la taille des vias etc... Ce qu'il nous fallait pour continuer à travailler.

Ainsi, nous avons pu commencer à essayer de résoudre certaines erreurs de DRC, puis des personnes ayant terminé leurs bloc ont pu nous rejoindre sur cette tâche. Nous pourrons continuer cette partie dans la mention prévue à cet effet, à savoir, Place & Route Top.

d) Test

Il fallait aussi prévoir un moyen de tester le bloc du CORDIC si jamais le bloc du Décodeur ne devait malheureusement pas fonctionner. Pour pallier ce problème, nous avons décidé de muxer l'entrée du CORDIC à la sortie du Codeur. Seulement le Decodeur envoyant des sinus échantillonnés à quatre échantillons par période et le Codeur envoyant en sortie des sinus échantillonnés à 20 échantillons par période, nous devions nous aussi réduire nos sinus à quatre échantillons par période. Mais pour éviter de réduire la précision du module zigbee, nous avons choisi de garder nos sinus échantillonnés à 20 mesures par périodes et de rajouter deux sorties supplémentaires pour le CORDIC. Le schéma du Codeur ressemble alors à ceci.

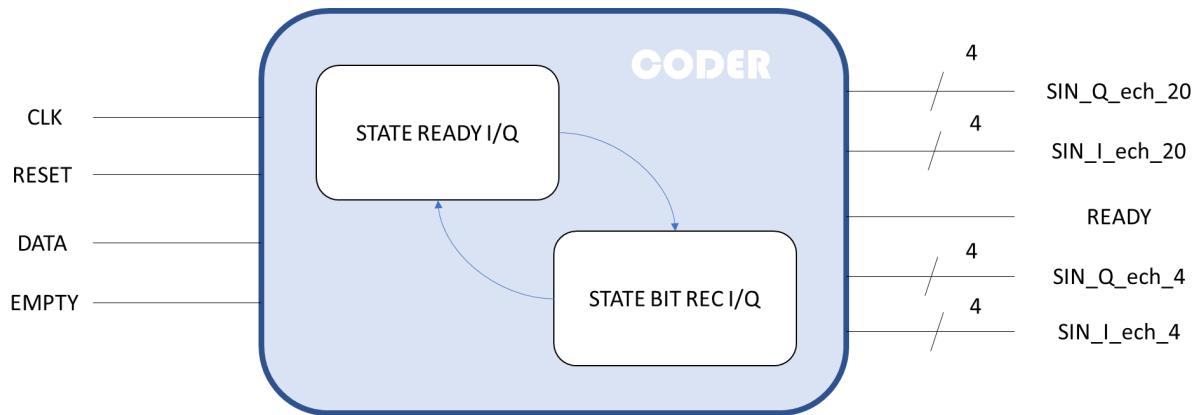


Figure 14 : Représentation finale de l'architecture du codeur avec les signaux de test pour le CORDIC

e) Documentation

Nous avons rédigé quelques documentations pour expliquer comment fonctionnent nos scripts et faire en sorte de les rendre utilisables pour les autres membres de l'équipe. Notamment sur le placement et routage où nous expliquons comment réaliser un premier placement, clock tree ou encore comment générer le GDS et regarder les erreurs de DRC. Cette étape de rédaction de documentation peut sembler parfois être une perte de temps mais au contraire elle a permis à d'autres groupes qui avaient fini eux aussi le développement de leur bloc de nous rejoindre sur le PNR de façon plus rapide que s'ils avaient dû comprendre par eux-même. De plus, si nous avions dû expliquer en détail à chacun, nous aurions perdu encore plus de temps.

2.3 Démodulateur IQ et filtres à réjection d'image (Mamadou Hawa DIALLO, Matisse REBOUD)

2.3.1 Spécification et implémentation RTL

2.3.1.1 Principe de fonctionnement

Le démodulateur IQ incluant la fonctionnalité de réjection de fréquence image est un bloc qui intervient dans la chaîne de réception du Zigbee tel que sur la *Figure1* ci-dessous.

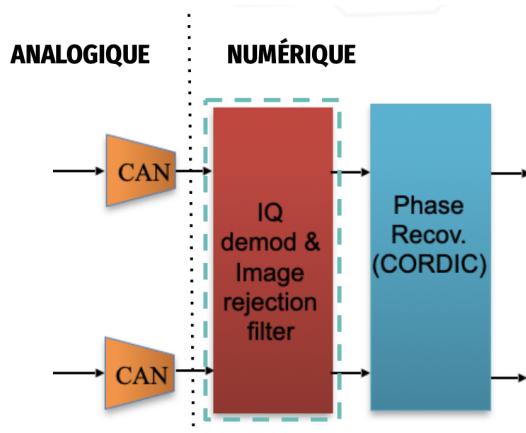


Figure1 : Le démodulateur IQ dans la chaîne de réception du Zigbee

Ainsi, il reçoit, en entrées, les signaux I et Q à la fréquence intermédiaire issus des CANs et, en sorties, il envoie les signaux I et Q, restitués en bande de base, au CORDIC.

Le principe de fonctionnement de ce bloc peut-être représenté comme suit :

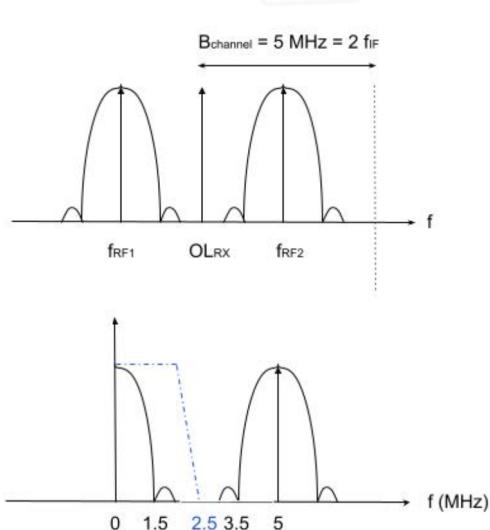


Figure2 : Principe de fonctionnement de ce bloc

Tel que sur la *Figure2* ci-dessus, le rôle du démodulateur IQ est de récupérer les signaux I et Q à $f_{IF} = 2,5$ MHz et de les ramener en bande de base à 500 KHz. Par la suite, les filtres passe-bas permettent de rejeter la fréquence image $f_{im} = 5$ MHz.

Pour ce faire, en collaboration avec le binôme chargé des ADCs et l'équipe digitale, selon les besoins de chaque bloc, nous avons choisi les spécifications suivantes :

- Résolution des ADCs = 4 bits
- Fréquence d'échantillonnage = 10 MHz
- Fréquence de fonctionnement = 50 MHz

Afin de concevoir notre bloc, nous avons utilisé Matlab tout au long de ce projet. En effet, l'étude système effectuée au début nous a permis d'étudier et de comprendre l'architecture du bloc représentée sur la *Figure3* ci-dessous :

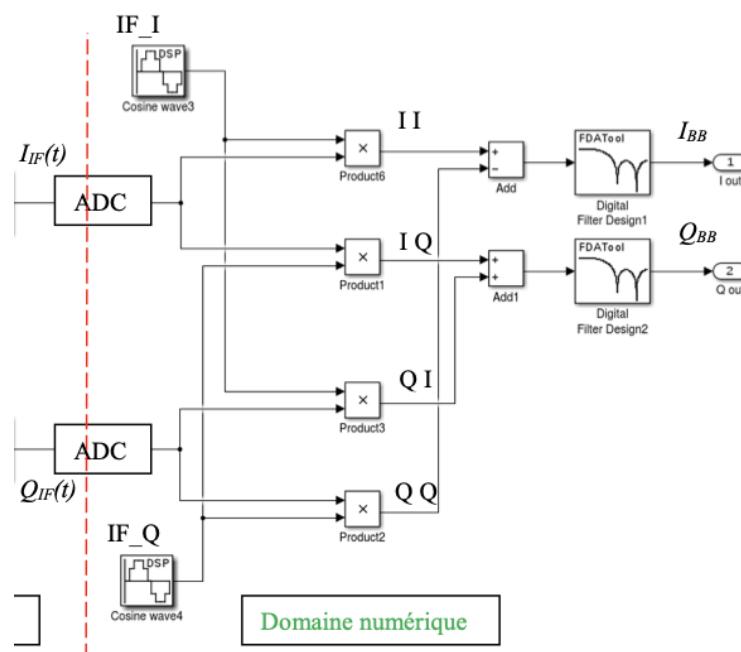


Figure3 : Architecture du démodulateur IQ incluant la fonctionnalité de réjection de fim

Ainsi, nous distinguons deux principales parties :

- le démodulateur IQ constitué d'un générateur de cosinus, un générateur de sinus, 4 multiplicateurs, 1 additionneur et 1 soustracteur ;
- les filtres à réjection d'image.

L'étude système nous a également permis de comprendre les expressions et les formes des signaux à chaque étape afin de vérifier progressivement nos résultats. Ainsi, nous avons :

- Les signaux d'entrée $I_{IF}(t)$ et $Q_{IF}(t)$

$$I_{IF}(t) = \frac{a}{2} * \cos(2\pi f_{IF} t + \varphi(t))$$

$$Q_{IF}(t) = -\frac{a}{2} * \sin(2\pi f_{IF} t + \varphi(t))$$

- Les cosinus et sinus générés IF_I et IF_Q

$$IF_I = \cos(2\pi f_{IF} t)$$

$$IF_Q = -\sin(2\pi f_{IF} t)$$

- Les signaux en sortie des multiplicateurs : $II(t)$, $QQ(t)$, $IQ(t)$ et $QI(t)$;
- Les signaux en sortie de l'additionneur et du soustracteur : $IQ(t) + QI(t)$ et $II(t) - QQ(t)$
- Les signaux en sortie des filtres :

$$I_{BB} = II - QQ \text{ et } Q_{BB} = IQ + QI$$

Également, pour notre testbench de validation, vu que nous n'avons pas les ADCs à côté pour nous fournir les signaux d'entrée, nous utilisons Matlab pour extraire $I_{IF}(t)$ et $Q_{IF}(t)$.

2.3.1.2 Design du démodulateur IQ

Le démodulateur se présente sous la forme suivante :

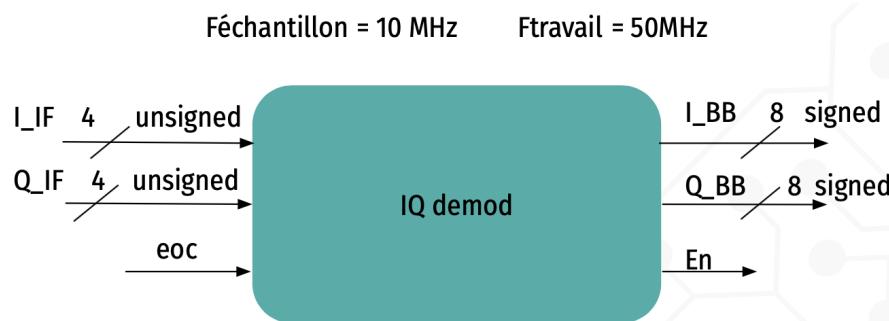


Figure : Représentation du démodulateur IQ

L'interface que nous avons choisie en commun avec le binôme se chargeant de l'ADC est la suivante :

- Donnée non signée sur 4 bits pour les voies I et Q plus un signal "end of conversion" afin d'indiquer au démodulateur quand un échantillon est prêt à être lu sur I et Q.
- La fréquence d'échantillonnage est de 10 MHz.
- En sortie du démodulateur, les signaux I et Q ramenés en bande de base sont envoyés sur 8 bits signés.

- Le signal “En” permet d’indiquer à la FSM du filtre quand un échantillon est envoyé en sortie.

2.3.1.3 implémentation du démodulateur IQ

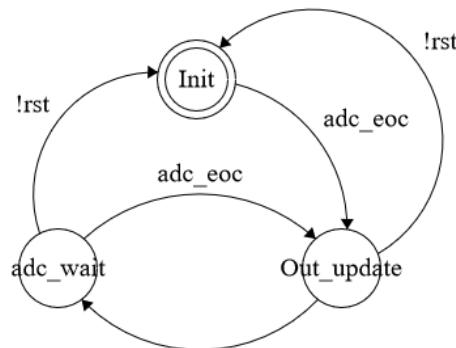


Figure4 : FSM du démodulateur IQ

En observant la FSM du démodulateur, nous constatons son fonctionnement :

- La mise à 1 du flag “adc_eoc” (en entrée) permet d’entrer dans l’état “Out_update”.
- Dans cet état, le résultat de la démodulation est envoyé dans les registres de sortie, et le flag “En” est mis à 1.
- Le traitement des données reçues s’effectue donc sur un coup d’horloge (passage en bits signés + multiplication + addition)

2.3.1.4 Design des filtres

Le dimensionnement des filtres passe-bas a été effectué sur Matlab en se basant sur l’étude système effectuée en début de projet. Ainsi, pour l’implémentation, nous avons dû effectuer un choix technique important : le type de filtre à réaliser. Nous avons trouvé différents types de design de filtres à réponse impulsionnelle finie, FIR (plus efficaces que les filtres à réponse impulsionnelle infinie, IIR qui reposent sur la rétroaction). Nous utilisons plus précisément la méthode “Equiripple” afin de simplifier l’implémentation. Le but étant de choisir la structure la plus adaptée à notre utilisation.

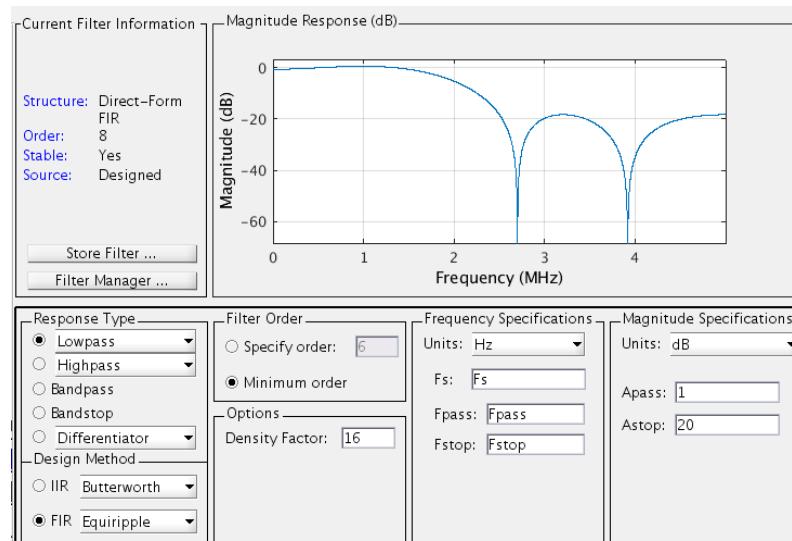


Figure5 : Caractéristiques des filtres

Ensuite, pour le choix des fréquences, étant donné que les signaux I_BB et Q_BB sont à 500 KHz et la fim à 5 MHz, nous choisissons une Fpass = 1,5 MHz et une Fstop = 2,5 MHz, valeurs qui nous semblent largement suffisantes pour rejeter la fréquence image.

Toutefois, il a fallu faire des compromis pour l'atténuation. En effet, en augmentant cette dernière, nous augmentons également la taille des filtres. Ainsi, afin d'obtenir un bon filtrage tout en optimisant la taille, nous avons décidé d'atténuer à 20 dB pour un ordre de 8 soit 9 coefficients symétriques codés sur 8 bits signés. Par la suite, enfin de les utiliser en systemVerilog, les coefficients ont été convertis en hexadécimal tel que sur la *Figure6* ci-dessous.

Hexadécimal	Binaire	Valeur décimale arrondie	Valeur décimale
F5	11110101	-0,04365	-0,043650227900904
E9	11101001	-0,08884	-0,088841443666288
1A	00011010	0,10109	0,101093197982018
4B	01001011	0,29103	0,291027839630325
69	01101001	0,41092	0,410917357587676

Figure6 : Coefficients des filtres

2.3.1.5 implémentation des filtres

La contrainte principale est la fréquence de fonctionnement, dans notre cas, nous travaillons à 50 MHz avec une fréquence d'échantillonnage de 10 MHz. Il faut donc traiter la donnée en 5 coups d'horloge maximum.

Comme le filtre est d'ordre 8, nous avons directement exclu les structures séquentielles. car il aurait fallu une dizaine de coups d'horloge pour traiter une information.

Nous nous sommes donc intéressés aux structures parallèles :

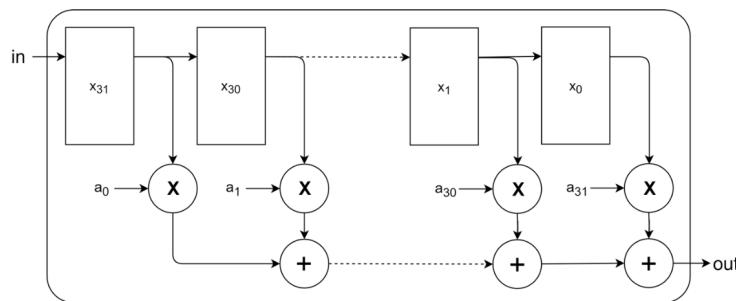


Figure7 : Architecture parallèle canonique

Les structures parallèles ont l'avantage de calculer "out" en une période d'horloge. Cependant, il faut apporter une attention particulière au chemin critique : sur la *Figure7* ci-dessus, on voit que le chemin le plus long entre les 2 registres (supposer un registre en sortie) est d'un multiplicateur + (n-1) additionneurs (n = nombre de coefficients). Le multiplicateur prenant déjà beaucoup de temps de calcul, et nos données sur les additionneurs ayant une taille conséquente (entre 16 et 20 bits), il nous a fallu trouver une manière d'optimiser ce chemin critique.

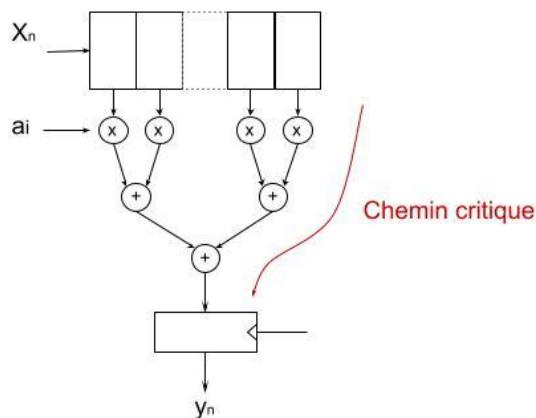
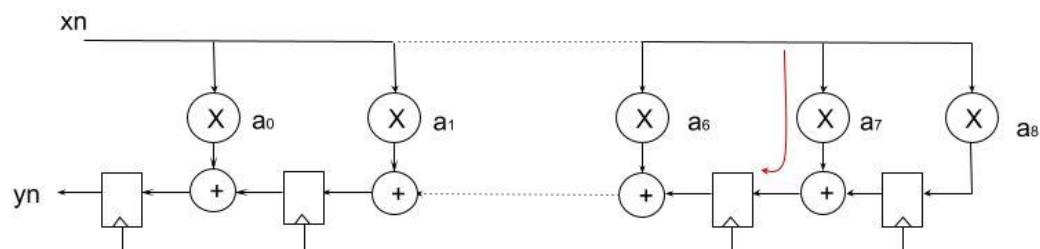


Figure8 : Autre type d'architecture parallèle

Une première solution, présentée sur la *Figure8* ci-dessus, permet de diminuer le nombre d'additionneurs. Cette structure permet également d'ajouter des registres entre chaque étage d'additionneurs pour réduire davantage le chemin critique. Cependant, nous avons préféré la structure suivante :



$$y_n = a_8x_{n-8} + a_7x_{n-7} + a_6x_{n-6} + \dots + a_1x_{n-1} + a_0x_n$$

Figure9 : Architecture parallèle avec retiming

Cette structure est la structure de base avec “retiming” : les registres sont déplacés afin d'avoir le meilleur chemin critique possible. Ce dernier est indiqué par la flèche **rouge** et est constitué d'un multiplicateur + un additionneur.

Remarque : pour améliorer davantage le chemin critique après avoir observé des slacks < 0 en worst, lors de la synthèse, nous avons ajouté un étage pipeline au filtre (ajout de registres entre les multiplicateurs et les additionneurs) .

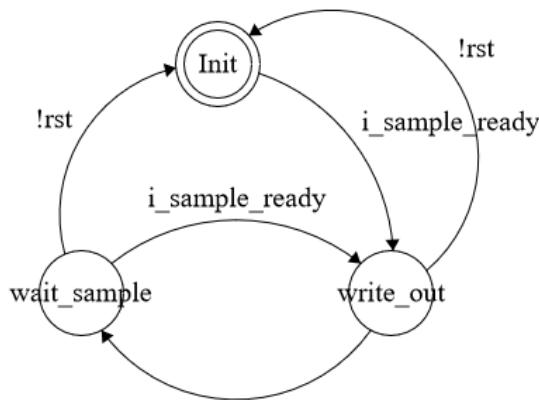


Figure10 : FSM décrivant le fonctionnement du filtre

La FSM ci-dessus décrit le fonctionnement du filtre. Pour chaque nouvel échantillon sur le filtre ($i_{sample_ready} = 1$), les données vont transiter dans les deux étages pipeline du filtre.

L'état initial met tous les registres du filtre à zéro.

Pour la partie opérative, il s'agit uniquement d'assignments avec des multiplications / additions entre signaux.

En sortie du filtre, un masquage est effectué pour ramener la valeur sur 4 bits pour l'interface avec le CORDIC :

```

o_I_postfilter <= I_data_add_0[14:11];
  
```

Le bit 14 est le MSB par rapport aux valeurs traitées (observé en simulation avec des entrées pleine amplitude).

2.3.1.6 Résultats de simulations

Pour le testbench, afin d'avoir une simulation au plus proche de la réalité, nous avons utilisé la modélisation matlab de l'étude système. Nous avons ensuite récupéré les valeurs en sortie des ADC pour les transformer sous forme de données (sur 4 bits) non signées.

Ainsi, nous pouvons simuler un fonctionnement réel, et observer les courbes démodulées en sortie du bloc. Nous pouvons aussi appliquer des valeurs en entrée avec une fréquence du canal seul, ou avec la fréquence du canal voisin en plus, afin d'observer l'utilité du filtre numérique pour enlever la fréquence Image.

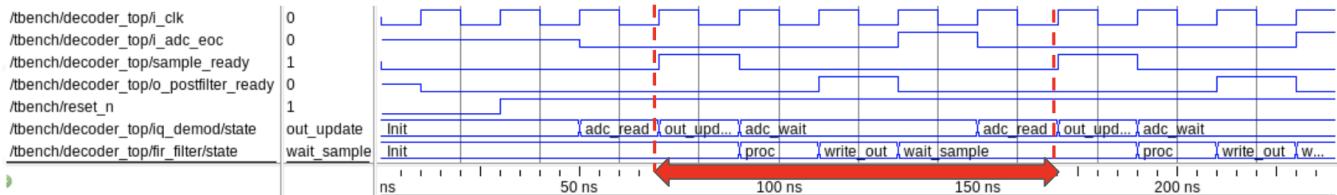


Figure11 : Partie contrôle du démodulateur

On observe ci-dessus le fonctionnement de la partie contrôle du démodulateur, notamment les 2 FSM décrites précédemment et leurs signaux de communication. La section en rouge correspond au cycle de gestion d'un échantillon.

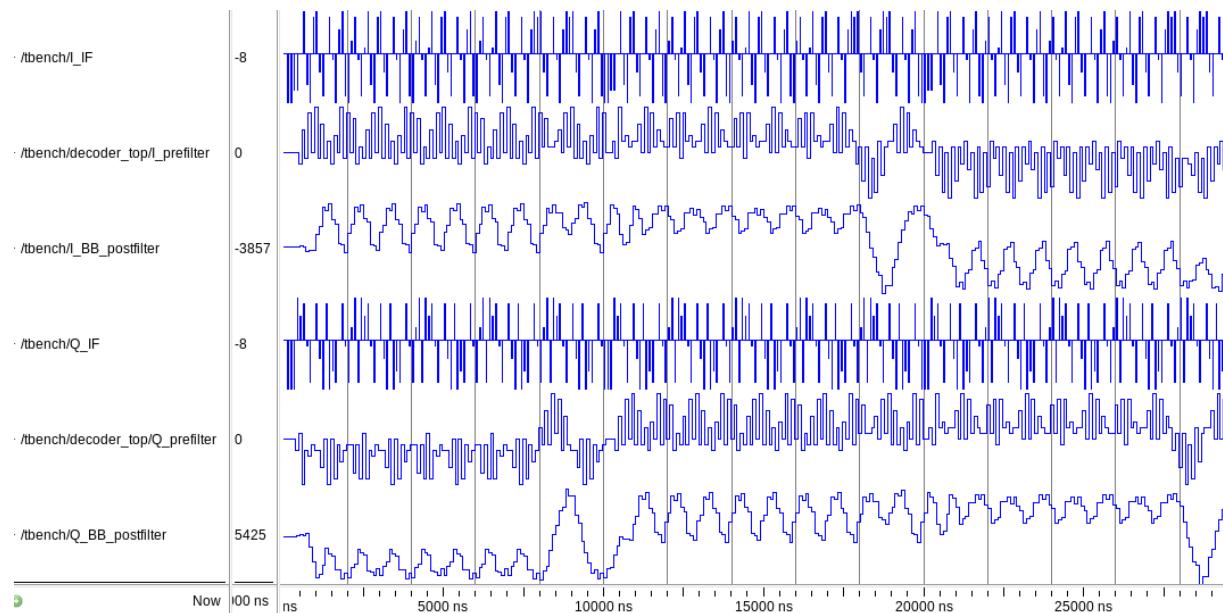


Figure12 : Simulation pré-synthèse du bloc

Les résultats de simulation ci-dessus montrent le fonctionnement global du bloc. En entrées (I_{IF} et Q_{IF}), nous voyons que les données en f_{IF} sont illisibles.

En sortie du démodulateur ($I_{prefilter}$ / $Q_{prefilter}$), nous arrivons à deviner les demi-sinus caractéristiques de la modulation MSK, cependant nous observons la présence d'une composante HF en plus du signal : c'est la fréquence image.

Finalement, en sorties du filtre ($Q_{BB_postfilter}$ / $I_{BB_postfilter}$), nous retrouvons le signal composé de demi-sinus de la MSK.

Les déformations que nous pouvons observer sur le sinus en sortie sont principalement des parasites induits par la fréquence image qui n'est pas entièrement atténuée par le filtre.

Nous avons également effectué la caractérisation du filtre seul, avec entre autres la réponse à une impulsion :

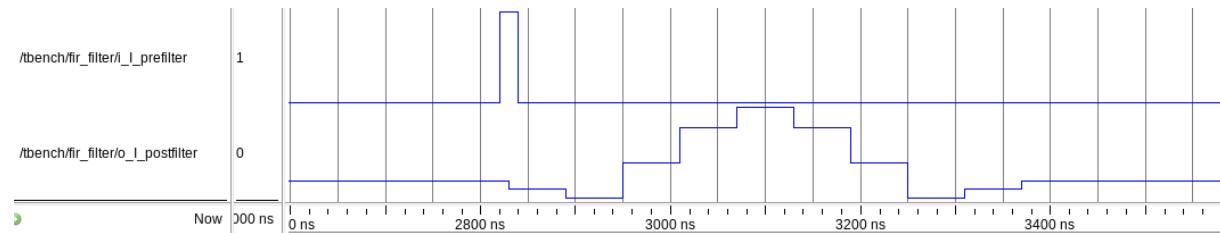


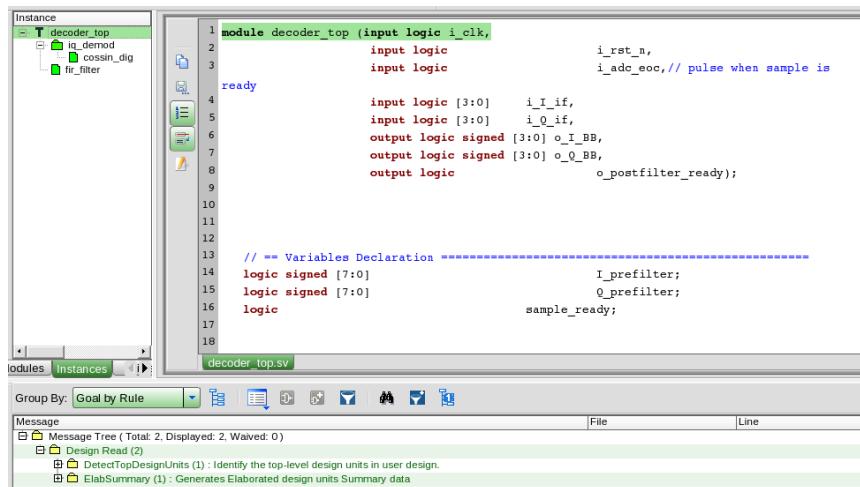
Figure13 : Réponse à une impulsion

Nous voyons bien, sur la *Figure13* ci-dessus, qu'il s'agit d'un filtre passe-bas (amortissement de l'impulsion, avec du retard).

2.3.2 Synthèse

2.3.2.1 Vérification du code avec Spyglass

Une fois le fonctionnement du bloc complet validé par le biais de différentes simulations pré-synthèse, nous vérifions sur Spyglass que notre code est synthétisable.



The screenshot shows a software interface for semantic verification. On the left, there's a tree view of an instance named 'decoder_top' containing components like 'iq_demod', 'cossin_dig', and 'fir_filter'. The main area is a code editor displaying Verilog-like code for a module 'decoder_top'. The code includes input and output logic declarations, variable declarations, and a 'ready' signal assignment. Below the code editor is a toolbar with various icons. At the bottom, there's a message panel titled 'Message' with several entries, including 'Design Read (2)' and 'ElabSummary (1) : Generates Elaborated design units Summary data'. The status bar at the bottom right shows 'File' and 'Line'.

```

Instance
- T decoder_top
  - iq_demod
  - cossin_dig
  - fir_filter

1 module decoder_top (input logic i_clk,
2   input logic i_rst_n,
3   input logic i_adc_eoc, // pulse when sample is
4   ready
5   input logic [3:0] i_I_if,
6   input logic [3:0] i_Q_if,
7   output logic signed [3:0] o_I_BB,
8   output logic signed [3:0] o_Q_BB,
9   output logic o_postfilter_ready);
10
11
12
13 // == Variables Declaration -----
14 logic signed [7:0] I_prefilter;
15 logic signed [7:0] Q_prefilter;
16 logic sample_ready;
17
18
decoder_top.sv

```

Figure14 : Vérification de la sémantique du code

Ainsi, nous constatons qu'il n'y a aucune violation et nous passons donc à la synthèse sur Design Vision.

2.3.2.2 Synthèse avec Design Vision

Nous effectuons la synthèse du bloc complet afin d'obtenir un circuit synthétisé constitué de portes logiques et dont la netlist sera utilisée pour l'étape de placement et routage.

Pour ce faire, nous définissons une `clock = 20 ns` et afin de se laisser une marge de sécurité, nous nous fixons pour objectif : un slack en worst conditions de minimum 5.

Ainsi, lors de la première synthèse, nous avions un slack de -7 en worst donc nous avons amélioré notre code en rajoutant des registres dans l'implémentation des filtres. De ce fait, après plusieurs optimisations, nous obtenons les caractéristiques finales suivantes :

Worst conditions	
data required time	19.99
data arrival time	-14.64
slack (MET)	5.34

Typical conditions	
data required time	20.00
data arrival time	-7.83
slack (MET)	12.17

Best conditions	
data required time	20.00
data arrival time	-4.40
slack (MET)	15.60

Figure15 : Report timing

Ces valeurs du slack dans les 3 conditions sont alors largement satisfaisantes.

Parallèlement, nous vérifions la surface et la puissance :

Combinational area:	222076.400452
Buf/Inv area:	23132.200523
Noncombinational area:	154044.799988
Macro/Black Box area:	0.000000
Net Interconnect area:	59256.000000
Total cell area:	376121.200439
Total area:	435377.200439

Figure16 : Report area

- La surface du bloc complet est de $0,44 \text{ mm}^2$, ce qui est plutôt raisonnable pour cette architecture globale renfermant des filtres. Nous constatons également sur la *Figure16* ci-dessus que nous n'avons pas de black box et que la partie combinatoire occupe la moitié de la surface.

Power Group	Internal Power	Switching Power	Leakage Power	Total Power (mW)	Attrs
io pad	0.0000	0.0000	0.0000	0.0000 (0.00%)	
memory	0.0000	0.0000	0.0000	0.0000 (0.00%)	
black box	0.0000	0.0000	0.0000	0.0000 (0.00%)	
clock network	0.0000	0.0000	0.0000	0.0000 (0.00%)	
register	10.0229	0.5385	1.5708e+03	10.5614 (75.37%)	
sequential	0.0000	0.0000	0.0000	0.0000 (0.00%)	
combinational	1.2678	2.1828	1.7830e+03	3.4506 (24.63%)	
Total	11.2907 mW	2.7213 mW	3.3537e+03 pW	14.0120 mW	

Figure17 : Report power

- Nous pouvons observer sur la *Figure 17* ci-dessus que la puissance totale est de 14 mW avec 10,6 mW provenant des registres, ce qui est cohérent vu que nous en avons rajoutés en plus afin d'optimiser le slack.

2.3.2.3 Simulation Post-Synthèse

Afin de vérifier que la synthèse s'est bien déroulée, nous effectuons plusieurs simulations post-synthèse.

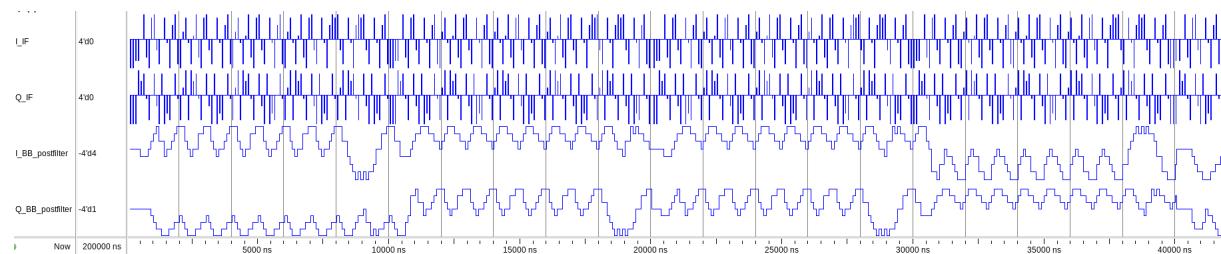


Figure18 : Simulation post-synthèse du bloc

Nous pouvons constater sur la *Figure 18* ci-dessus que nous obtenons les mêmes résultats que lors des simulations pré-synthèse, les signaux de sortie I_BB_postfilter et Q_BB_postfilter sont

bien ramenés en bande de base à 500 KHz et l'influence de la fréquence image a été rejetée. Ainsi, cela nous permet de valider la synthèse et le fonctionnement final de notre bloc complet.

2.3.3 Stratégie de test

L'objectif est d'anticiper le test du démodulateur IQ en visualisant le moyen de tester le bloc sur silicium.

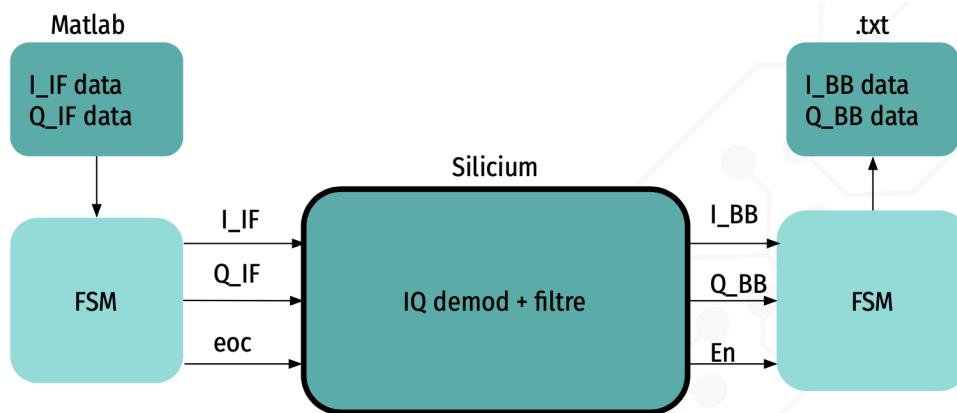


Figure19 : Représentation de la stratégie de test

La stratégie de test est la suivante : à l'aide d'un module FPGA, nous reproduisons les interfaces du bloc. Pour les données à envoyer, comme lors de nos testbenches, nous récupérons des données issues du modèle matlab du module Zigbee. Ces données sont transformées sur 4 bits non signés à l'aide d'un excel.

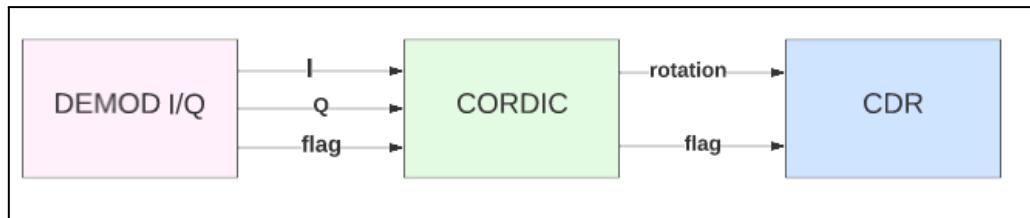
Pour la sortie, nous pourrions stocker les résultats d'échantillonnage dans un fichier pour en vérifier le contenu manuellement (en comparant avec le résultat de la simulation matlab). Une autre solution serait de créer une comparaison automatique.

2.4 CORDIC (Paul-Arthur MEHL, Megi MYFTARAJ)

Spécification et implémentation RTL

Interface:

Le Cordic récupère les signaux I et Q démodulés sur 4 bits et renvoie au bloc CDR le sens de rotation non synchronisé sur un bit, ainsi qu'un signal de type flag, pour signaler au CDR quand récupérer cette donnée.



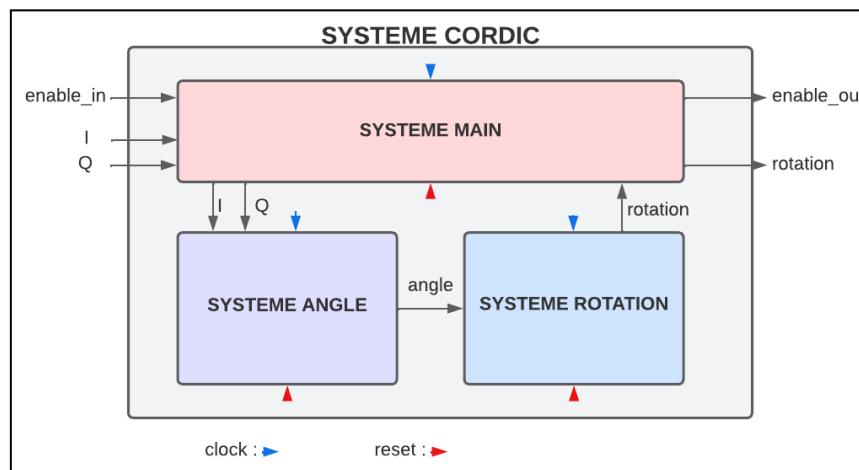
Les blocs à l'interface du Cordic

Objectif:

Fournir un sens de rotation (direction) en fonction des échantillons IQ récupérés en entrée.

Découpage fonctionnel global:

Le système du Cordic est composé de 3 blocs : le bloc principal "Système Main" qui met à jour les signaux I, Q en entrée ainsi que le bit de "rotation" en sortie. Il génère également un signal de sortie "enable_out" qui est activé à chaque mise à jour de la sortie rotation. Le bloc "Système Angle" calcule la phase correspondante pour chaque paire de données (I,Q) en se basant sur l'algorithme de Cordic. Le bloc "Système Rotation" compare la phase actuelle avec la précédente pour déterminer un sens de rotation. Le fonctionnement de chaque bloc est décrit ci-après.

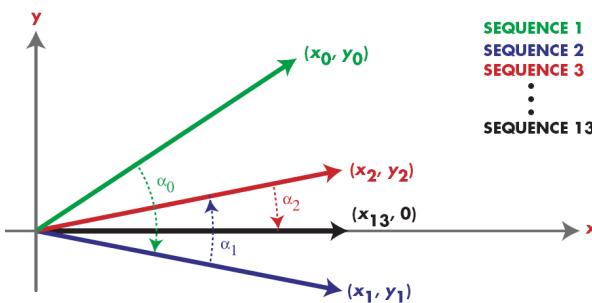


Découpage fonctionnel global du "Système Cordic"

Bloc “SYSTEME ANGLE”:

Algorithme du Cordic:

L'algorithme de Cordic est basé sur la rotation d'un point (X_1, Y_1) à un point (X_2, Y_2) avec un angle standard " Θ " où $\tan\theta = 2^{-i}$ où $i = 0,1,2,3\dots$. Afin de déterminer la phase, on réalise des rotations successives d'angles standards Θ . Une totalité de 6 rotations successives nous permet d'avoir $+/ - 2$ degrés de précision sur le calcul de la phase. Moins de rotations successives résultent en moins de précision sur les phases calculées, donc la possibilité d'envoyer un mauvais sens de rotation au CDR augmente.



Exemple de rotations successives

Dans ce bloc, les signaux qu'on manipule sont I et Q. Les équations de calcul appliquée de manière itératives sur I et Q sont traduites tel que :

$$next_I = I \pm Q \times \tan\theta \Rightarrow next_I = I \pm Q \times 2^{-i}$$

$$next_Q = Q \pm I \times \tan\theta \Rightarrow next_Q = Q \pm I \times 2^{-i}$$

Les équations appliquées de manière itératives sur l'angle sont tel que :

$$next_angle = present_angle \pm \arctan(2^{-i})$$

Les cellules nécessaires à l'implémentation de l'algorithme sont des registres à décalages, des additionneurs, des soustracteurs et des multiplexeurs.

Opération	Implémentation
$/(2^i)$	shift register
+	adder
-	soustractor
If	Mux

Le bloc “SYSTÈME ANGLE” est construit avec une architecture pipeline découpée en 9 étages.

Par conséquent, 9 coups d’horloges sont nécessaires pour le calcul d’un angle à partir des échantillons I et Q appliqués en entrée du bloc.

Le choix d’une architecture pipeline nous offre la possibilité de sortir un angle à chaque coup d’horloge.

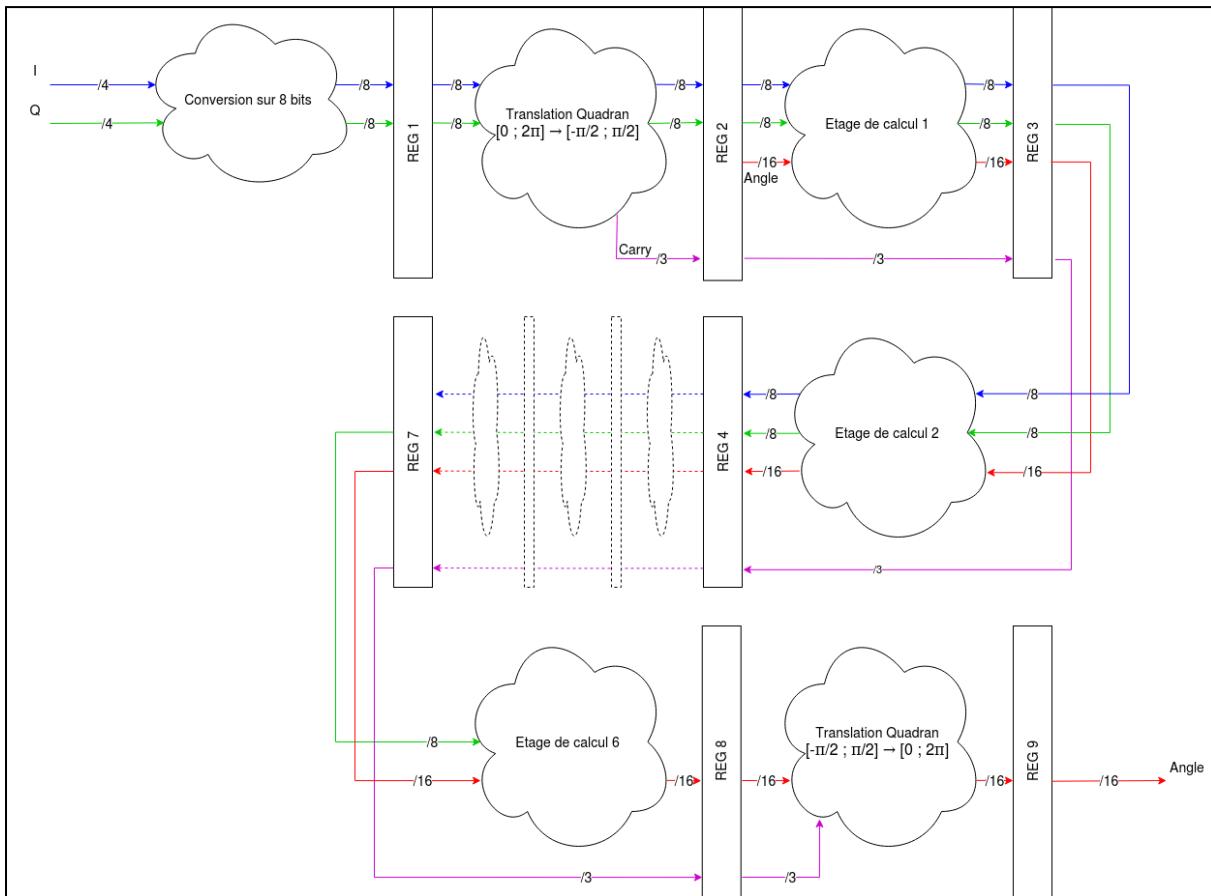
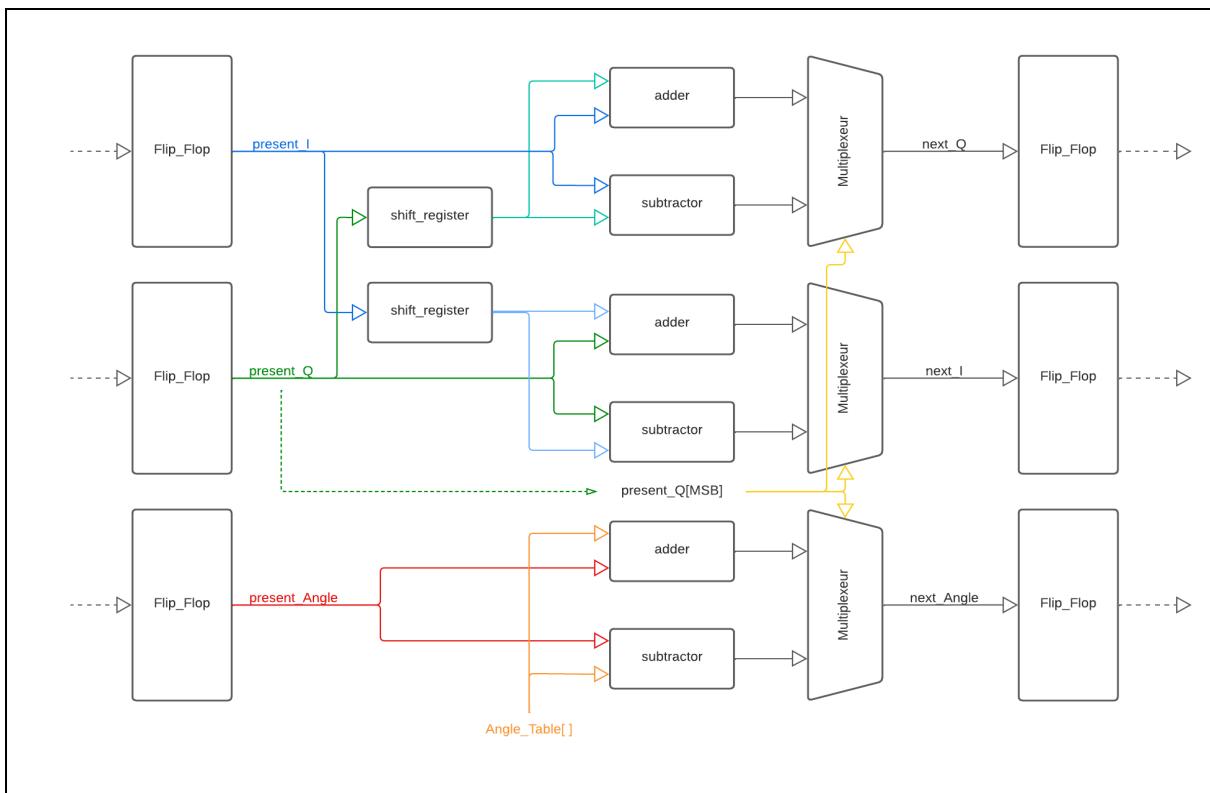


Schéma représentatif des étages du pipeline

Les portes numériques ne permettent pas de faire des opérations de arctan, les valeurs des angles standards associées sont écrites en “dure” :

$\text{atan}(2^0)$	45°	$\text{atan}(2^{-3})$	7°
$\text{atan}(2^{-1})$	27°	$\text{atan}(2^{-4})$	3°
$\text{atan}(2^{-2})$	14°	$\text{atan}(2^{-5})$	2°

Le fonctionnement d’un étage de calcul du pipeline est représenté par schéma ci-après.



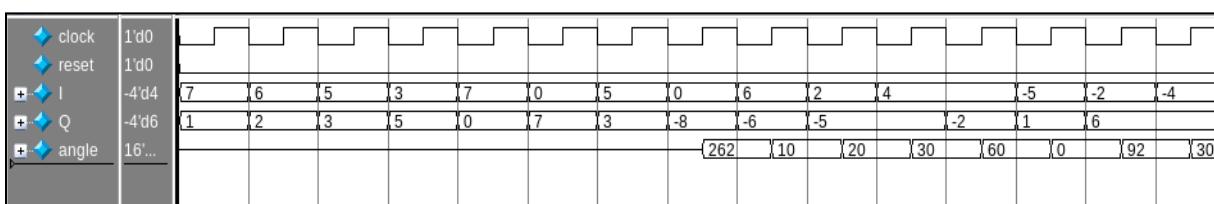
Architecture d'un étage de calcul

Validation de l'implémentation du calcul d'angle:

La validation de l'algorithme passe par le calcul d'angles dans les quatres quadrants c'est à dire $[0 ; \frac{\pi}{2}]$, $[\frac{\pi}{2} ; \pi]$, $[\pi ; \frac{3\pi}{2}]$, et $[-\frac{3\pi}{2} ; 2\pi]$.

```
// Angle + Q1
I = 4'b0111; Q = 4'b0001; reset = 1'b0; #20ns // +8
I = 4'b0110; Q = 4'b0010; reset = 1'b0; #20ns // +18
I = 4'b0101; Q = 4'b0011; reset = 1'b0; #20ns // +30
I = 4'b0011; Q = 4'b0101; reset = 1'b0; #20ns // +59
I = 4'b0111; Q = 4'b0000; reset = 1'b0; #20ns // 0
I = 4'b0000; Q = 4'b0111; reset = 1'b0; #20ns // 90
I = 4'b0101; Q = 4'b0011; reset = 1'b0; #20ns // +30
```

Testbench pour valider le fonctionnement pour le quadrant $[0 ; \frac{\pi}{2}]$

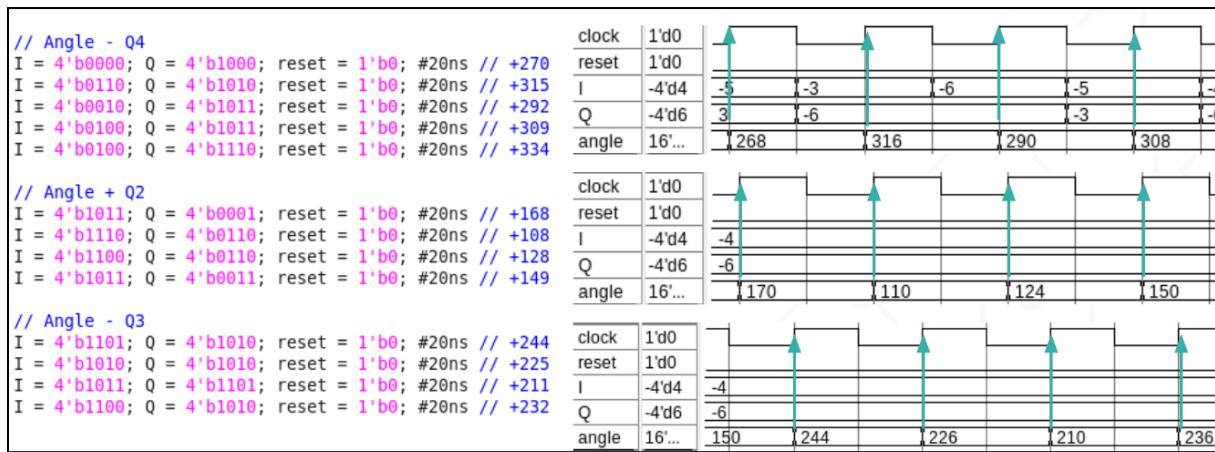


Validation du calcul d'angle pour le quadrant $[0 ; \frac{\pi}{2}]$

Comme mentionné, le calcul d'un angle nécessite 9 coups d'horloge. Une fois le système en routine de fonctionnement, un nouvel angle est disponible en sortie à chaque front d'horloge.

Note: N'ayant pas activé le reset au démarrage (reset actif haut), le premier angle affiché (262) est déterminé à partir d'échantillons I et Q inconnues et est donc non significatif.

Validation des quadrants $[\frac{\pi}{2}; \pi]$, $[\pi; \frac{3\pi}{2}]$, et $[\frac{3\pi}{2}; 2\pi]$:



Testbench et résultats de simulations pour les quadrants $[\frac{\pi}{2}; \pi]$, $[\pi; \frac{3\pi}{2}]$, et $[\frac{3\pi}{2}; 2\pi]$

Bloc "SYSTEME ROTATION"

Le bloc "SYSTÈME ROTATION" est construit avec une architecture pipeline 3 étages. La direction est déterminée à l'aide d'un algorithme comparant un angle avec l'angle précédent. L'algorithme calcule la différence entre deux angles et compare cette différence à 180° . Si $\Delta > 180^\circ$ le sens est horaire (`dir=1'b0`), sinon le sens est trigonométrique (`dir=1'b1`).

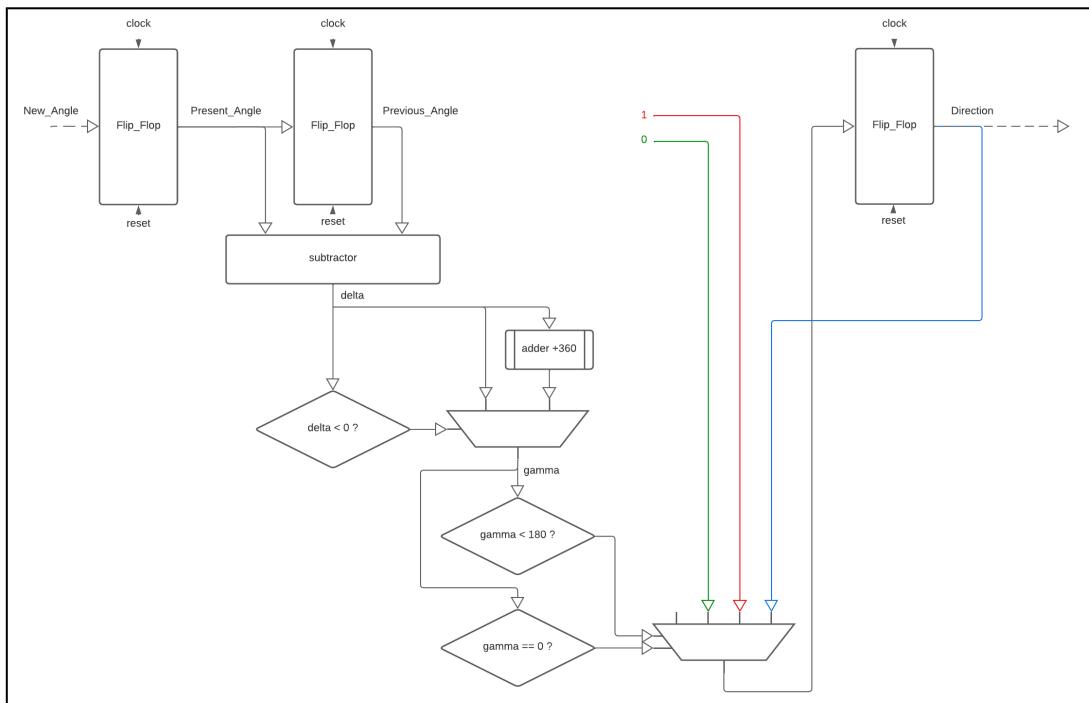
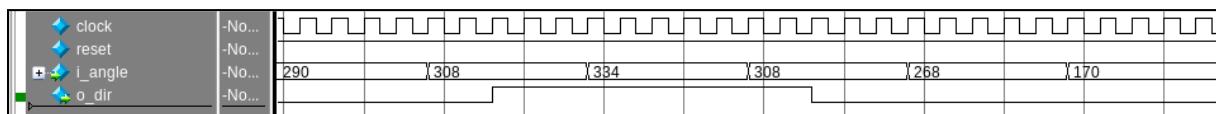


Schéma représentatif du bloc rotation

Les angles sont compris entre 0 et 360°. Si la différence entre deux angles est inférieure à 180° alors le sens rotation est "horaire". Inversement, le sens de rotation est "trigonométrique". Par conséquent, le sens de rotation est exprimé par une valeur binaire soit 0 ou 1.

A chaque front d'horloge, les angles pour le calcul et le sens de rotation sont mis à jour. Trois coups d'horloge sont nécessaires pour déterminer le sens de rotation correspondant à un angle. Par ailleurs, si l'angle récupéré est égal au précédent, le sens de rotation est alors maintenu.

Validation de l'implémentation du calcul de sens de rotation:

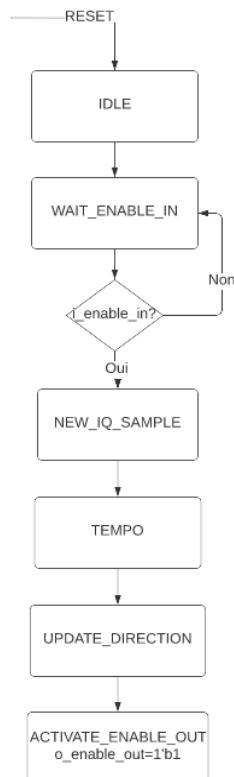


Validation du calcul de sens de rotation

Lorsqu'on compare ici 290° avec 308° et 334°, le sens de rotation est bien trigonométrique donc la sortie reste à 1. Elle passe à 0 lorsqu'on compare 334° avec 308°, 268°, 170° car le sens de rotation est horaire. Le fonctionnement du bloc de rotation est bien validé.

Block "SYSTÈME MAIN" :

Le bloc "SYSTÈME MAIN" est piloté par une MAE de 6 étages. Ce Bloc génère un signal de type flag pour dire au bloc d'après quand récupérer la donnée. C'est lorsqu'on entre dans l'état ACTIVATE_ENABLE_OUT qu'on active le flag de sortie enable_out.



Machine à états du bloc "Système Main"

Synthèse sur Spyglass

Nous avons vérifié sur **Spyglass** le niveau de synthétisation de notre circuit. Nous avons corrigé quelques erreurs liées au fonctionnement de la machine à états. Ensuite nous sommes passés à la synthèse du code, la traduction de notre code en Verilog en équations logiques. Nous avons imposé à l'outil de synthèse des contraintes pour obtenir un résultat optimisé en **surface** et **vitesse**.

Le résultat de synthèse a été conforme à notre attente. Dans la vue RTL on a bien retrouvé la totalité des blocs de notre Cordic. Le chemin critique du Cordic correspond au bloc de rotation.

Synthèse logique sur cible ASIC:

Le but d'une synthèse logique sur circuit numérique est de fournir une solution répondant au cahier des charges et en faisant également un compromis entre consommation, performance et surface. La priorité est le temps de propagation, ensuite la surface et enfin la consommation. Néanmoins la consommation n'est pas à négliger surtout depuis ces dernières années.

Une première synthèse de notre Cordic a été faite avec les réglages suivantes :

- Période d'horloge clk égale à 20 ns
- Aucune contraintes de surfaces

Le “path slack” en conditions worst nous indique que la data arrive au minimum 2,54 ns avant le front montant d'horloge.

Le chemin critique provient du bloc rotation.

La marge de sécurité étant à 5ns de slack, nous avons essayé d'optimiser le bloc rotation afin d'avoir un meilleur slack en condition worst.

La synthèse finale après optimisation du bloc de rotation (ajout de bascules), nous donne un slack de 6.96ns en conditions worst.

U556/Q (INV3)	0.33	13.03 r
mycordic/present ANGLE table req[5][15]/D (DF3)	0.00	13.03 r
data arrival time		13.03

clock clock (rise edge)	20.00	20.00
clock network delay (ideal)	0.00	20.00
mycordic/present ANGLE table req[5][15]/C (DF3)	0.00	20.00 r
library setup time	-0.01	19.99
data required time		19.99

data required time		19.99
data arrival time		-13.03

slack (MET)		6.96

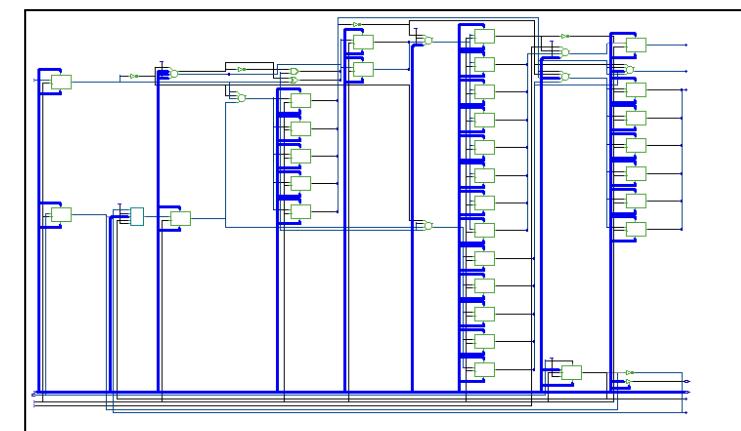
Path slack (WC) du bloc CORDIC

L'outil ‘Area Report’ nous a précisé le taux d'utilisation des LUTs, 126/15360 LUTs qui correspond à un taux d'utilisation de 0.82%. On a utilisé très peu de surface, ce qui est positif d'un point de vue *surface et consommation*.

Resource	Used	Avail	Utilization
IOS	27	173	15.61%
Global Buffers	1	8	12.50%
LUTs	126	15360	0.82%
CLB Slices	83	7680	1.08%
Dffs or Latches	165	15879	1.04%
Block RAMs	0	24	0.00%
Block Multipliers	0	24	0.00%
Block Multiplier Dffs	0	864	0.00%

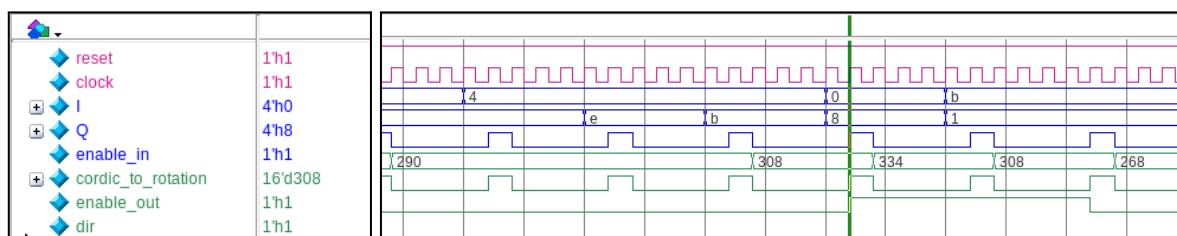
Report Area

La vue RTL du bloc “Système Angle” seul permet de distinguer les 9 étages de l’architecture pipeline.



RTL Schematic du bloc “Système Angle”

Simulation post synthèse du bloc complet :



Nous pouvons constater sur les figures que nous obtenons les mêmes résultats que lors des simulations pré-synthèse.

2.5 CDR (Alexandre Scouarnec, Elisabeth Porret)

2.5.1 Spécification et implémentation RTL

2.5.1.1 Contexte

Lors de la réception des données, les bits peuvent être reçus avec du retard ou de l'avance. Ainsi, le rôle du CDR (Clock Data Recovery) est de synchroniser les données entrantes avec l'horloge, afin de retranscrire correctement les bits de données reçus. Le bloc du CDR reçoit les données de direction du CORDIC, synchronise ces données avec le signal d'horloge, puis envoie les données synchronisées à la FIFO. Les signaux "i_flag" et "o_flag", lorsqu'ils valent 1, indiquent qu'une donnée est transmise.

Pour réaliser le CDR, nous avons choisi d'implémenter une structure à asservissement, composée d'un détecteur de phase de type Alexandre, d'un diviseur de fréquence et d'un bloc de décision. Enfin, la fréquence de l'horloge de travail a été fixée pour tous les blocs numériques à 50MHz et les données nous sont envoyées à 10MHz.

Afin d'augmenter la précision sur le bloc du CDR, nous avons choisi d'utiliser une fréquence de traitement des données à 2MHz : il nous a donc fallu ajouter un sous-bloc effectuant l'interface entre 10 et 2 MHz.

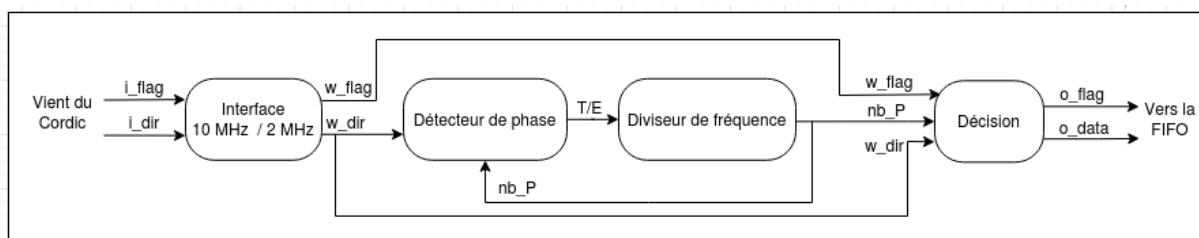


Figure 1 : Schéma de principe du CDR

2.5.1.2 Interface 10MHz/2MHz

Afin que le CDR puisse fonctionner à 2MHz et non à 10MHz comme les autres blocs numériques, nous avons eu besoin d'implémenter ce sous-bloc d'interface pour s'assurer de la valeur de direction du CORDIC. D'autre part, on sait que le CORDIC envoie 5 directions identiques à une fréquence de 10MHz.

Pour ce faire, nous avons choisi d'échantillonner la valeur de chaque direction "**i_dir**", grâce à 3 mesures effectuées à partir du signal "**i_flag**". Sur 5 coups de "**i_flag**", on échantillonne "**i_dir**" sur les 2e, 3e et 4e coups de "**i_flag**", pour se placer au centre des 5 coups. Puis, à l'aide d'une équation logique que nous avons déterminée, la valeur d'entrée "**w_dir**" prend la valeur (1 ou 0) la plus présente, comme le montre la table de vérité ci-dessous.

dir_d	dir_m	dir_f	w_dir
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

Figure 2 : Table de vérité pour “w_dir”

Sur le schéma de principe ci-dessous, les signaux “w_flag” et “w_dir” représentent les signaux “i_flag” et “i_dir”, échantillonnés à 2 MHz.

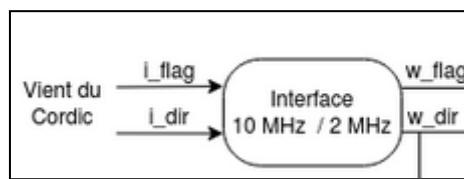


Figure 3 : Schéma de principe de l'interface 10 MHz / 2 MHz

2.5.1.3 DéTECTEUR DE PHASE

Le détecteur de phase permet de déterminer si la donnée est reçue avec du “retard” ou de “l'avance” par rapport à la fréquence d'envoi des données. Après avoir effectué une recherche documentaire sur le sujet, nous avons choisi d'implémenter une structure de type Alexandre, composée de 4 bascules D et de 2 XOR. En effet, il s'agit d'une structure souvent utilisée en numérique et simple à implémenter.

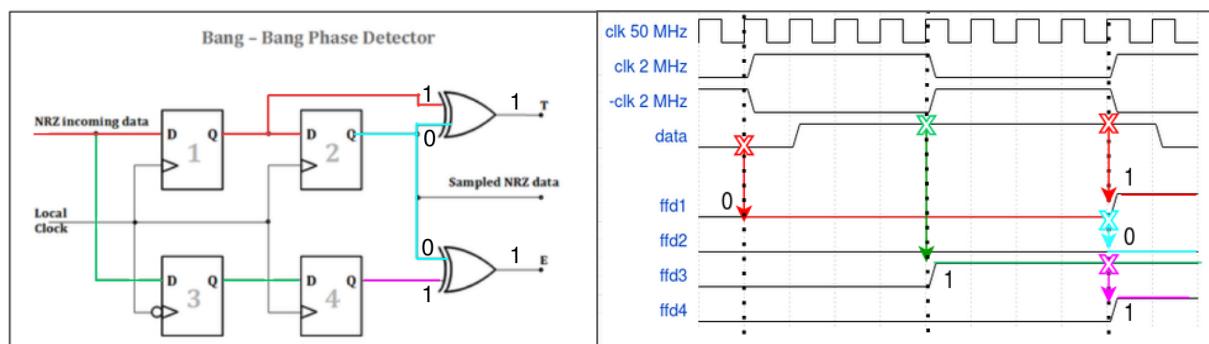


Figure 4 : DÉTECTEUR DE PHASE DE TYPE ALEXANDRE

Pour mieux expliquer le fonctionnement de ce sous-bloc, prenons l'exemple d'un retard. Sur la figure 4 ci-dessus, est représentée la clock à 50 MHz, le signal d'horloge à 2 MHz, la donnée entrante ainsi que les sorties de chaque bascule (appelées ffd) composant le détecteur de phase.

Sur le premier front montant du signal à 2 MHz, la donnée 0 est écrite sur la sortie de ffd1 (en rouge). Ensuite, sur le front montant d'après, cette même donnée est écrite en sortie de ffd2 (en bleu). Au même moment, une nouvelle donnée à 1 est écrite sur la sortie de ffd1 (en rouge). L'opération effectuée est alors " $T = 1 \text{ XOR } 0$ ", soit " $T = 1$: un décalage entre la donnée et la fréquence d'envoi des données est détecté.

Ensuite, sur le premier front montant du signal d'horloge à 2 MHz inversé, la donnée 1 est recopiée sur la sortie de ffd3 (en vert). Sur le front montant suivant du signal à 2 MHz, cette donnée est écrite en sortie de ffd4 (en rose). L'opération effectuée est alors " $E = 0 \text{ XOR } 1$ ", soit " $E = 1$: ici un retard est détecté.

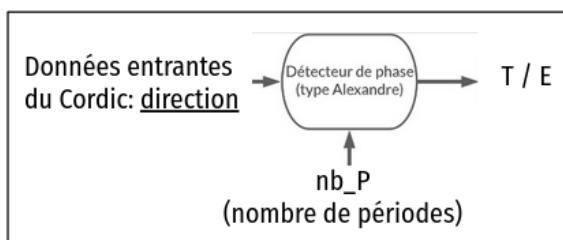


Figure 5 : Schéma de principe du détecteur de phase de type Alexandre

Le détecteur de phase reçoit en entrée la direction donnée par le CORDIC, ainsi que le nombre de périodes d'horloge (50MHz) contenues dans une période d'envoi des données (2MHz). Le nombre de périodes vaut donc initialement 25. Les sorties de ce sous-blocs sont les signaux "T" (Transition) et "E" (Early). "T" permet ainsi d'indiquer si le signal est décalé par rapport à l'horloge, et "E" précise s'il s'agit d'un retard ou d'une avance.

Enfin, comme montré sur la figure 6 ci-dessous, nous avons ajouté un multiplexeur ainsi qu'un compteur à l'entrée de chacune des bascules composant le détecteur de phase, afin d'échantillonner la direction à 2 MHz.

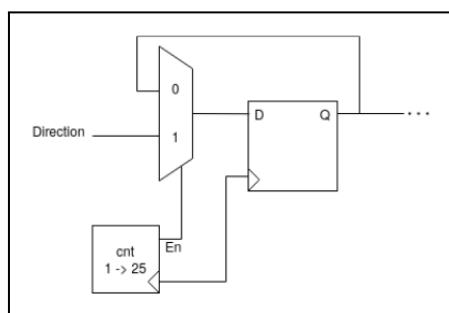


Figure 6 : Schéma du système mis en place pour échantillonner les données à 2 MHz

2.5.1.4 Diviseur de fréquence

Le diviseur de fréquence est situé derrière le détecteur de phase. Il reçoit donc en entrée les signaux “T” et “E”, et retourne en sortie le nombre de périodes (“nb_P”).

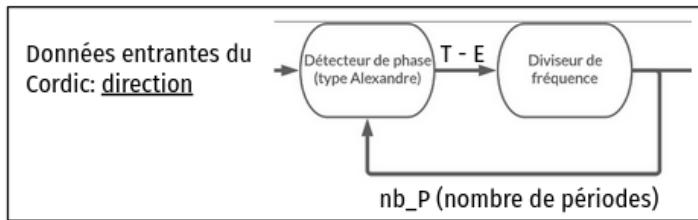


Figure 7 : Détecteur de phase et diviseur de fréquence

Le rôle de ce bloc est d'ajuster le nombre de périodes (“nb_P”) en fonction du retard ou de l'avance détectée. Par exemple, si un retard est détecté, le nombre de périodes va augmenter, ce qui se traduit par une diminution de la fréquence d'échantillonnage.

2.5.1.5 Décision

Le bloc de décision est situé après le diviseur de fréquence et se trouve en sortie de l'asservissement. Ce bloc permet de déterminer à quel moment les bits de direction vont être échantillonnés, à l'aide du nombre de périodes “nb_P” déterminé par le diviseur de fréquence.

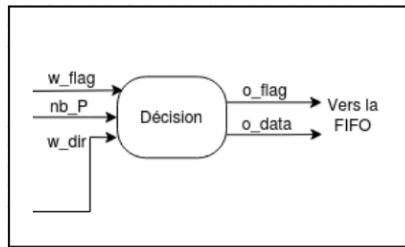


Figure 8 : Bloc de décision

2.5.2 Testbench, plan de validation (spécifications système)

Dans un premier temps, nous avons cherché à valider chacun des sous-blocs composant le CDR de manière séparée. Pour le sous-bloc du détecteur de phase, nous avons notamment simulé les éventuels retards/avances dans le testbench, et vérifié que les signaux “T” et “E” étaient corrects. Une fois tous les sous-blocs validés, nous les avons assemblés, puis testé le système du CDR global.

Ensuite, afin de valider le fonctionnement global du CDR, nous avons pensé à tester différents paramètres, en prenant en compte des paramètres extérieurs, mais également les spécifications des blocs rattachés au CDR. Ainsi, nous avons pu construire le cahier des charges suivant :

- Envoi d'un signal indiquant lorsqu'une direction est envoyée ("i_flag" en entrée et "o_flag" en sortie) aux blocs rattachés au CDR.
- Potentielle présence de retards/avance pouvant survenir lors de l'envoi des données.
- Le binôme chargé d'implémenter la FIFO nous a indiqué qu'il ne souhaitait pas conserver la première donnée ainsi que le flag de sortie ("o_flag") soit envoyé un coup d'horloge après la donnée ("o_data").
- Le CORDIC envoie les données sur une durée qui peut être variable comme 400, 500 (valeur de base) ou 600 ns. De plus, on peut avoir des glitches d'une durée de 100 ns, alors que le fonctionnement du CDR est basé sur la fréquence d'envoi des données à 2 MHz (soit 500 ns).

2.5.3 Résultats de simulations

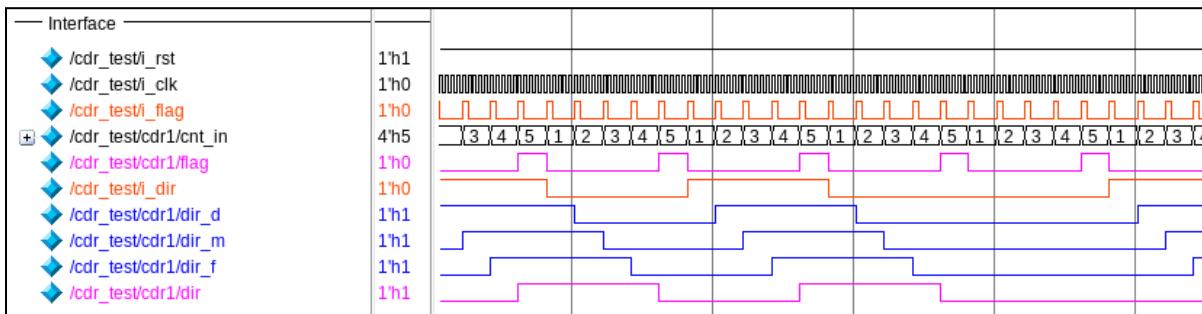


Figure 9 : Simulation pré-synthèse de l'interface 10MHz / 2MHz

Sur la figure 9 ci-dessus, est représenté le résultat de simulation pour le sous-bloc de l'interface. Les signaux "dir_d" (début), "dir_m" (milieu) et "dir_f" (fin) permettent d'échantillonner le signal "i_dir". Enfin, on vient actualiser le signal "dir" avec la bonne valeur de direction.

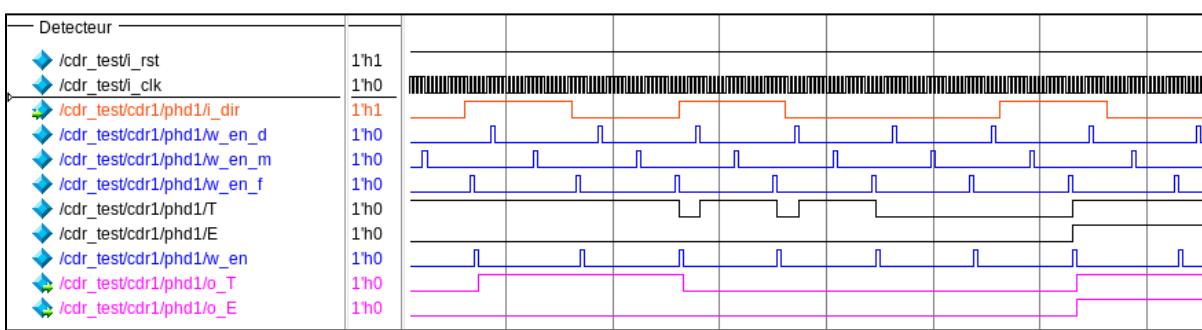


Figure 10 : Simulation pré-synthèse du détecteur de phase

La figure 10 ci-dessus présente le résultat de simulation pré-synthèse du détecteur de phase. Les signaux "w_en_d" (début), "w_en_m" (milieu) et "w_en_f" (fin) permettent d'échantillonner le signal "i_dir". On constate que lorsqu'un retard est détecté, les signaux "T" et

“E” varient. Enfin, les signaux “o_T” et “o_E” représentent l’échantillonnage de “T” et “E”, après que le signal “w_en_f” soit retombé à 0 (c'est-à-dire quand l'échantillonnage d'une donnée “i_dir” est terminé). Pour échantillonner “T” et “E” au bon moment, le signal “w_en” est utilisé.

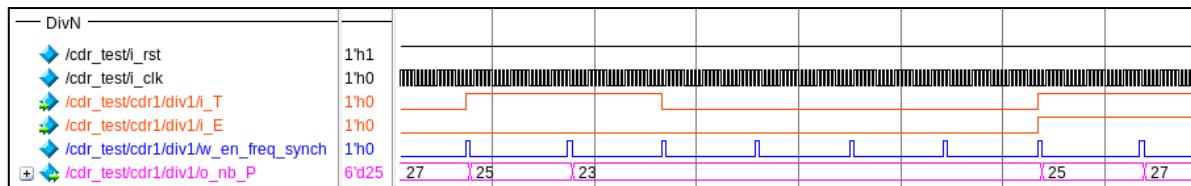


Figure 11 : Simulation pré-synthèse du diviseur de fréquence

La figure 11 ci-dessus présente le résultat de simulation pré-synthèse du diviseur de fréquence. Le signal “w_en_freq_synch” permet de modifier le nombre de période “o_nb_P”, et varie bien en fonction du retard ou de l'avance détectée.

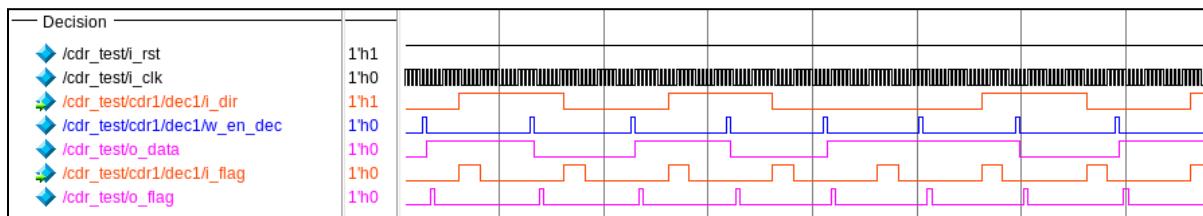


Figure 12 : Simulation pré-synthèse du bloc de décision

Grâce au résultat de simulation du sous-bloc de décision (figure 12 ci-dessus), le signal “o_data” est bien généré avec le signal i_dir par l'intermédiaire de “w_en_dec”. Ce dernier est actif en fonction du nombre de périodes, afin de se trouver le plus au centre de la direction possible, et de permettre la recopie de “i_dir” sur “o_data”. Pour les flags on a bien “o_flag” qui est actif une période d'horloge plus tard que l'actualisation de la donnée.

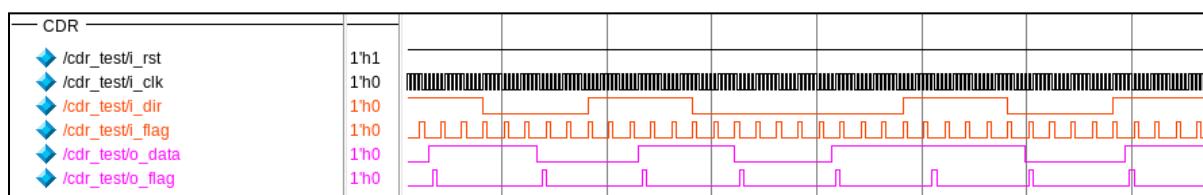


Figure 13 : Simulation pré-synthèse du CDR global

Comme on peut le constater sur la figure 13 ci-dessus, la donnée en sortie “o_data” est correcte. Nous l'avons simplement inversée par rapport à la donnée d'entrée venant du CORDIC, pour être cohérente avec la donnée envoyée en transmission (ceci vient du sens d'interprétation de la phase déterminé par le CORDIC, qui était simplement “inversé” par rapport à la donnée transmise).

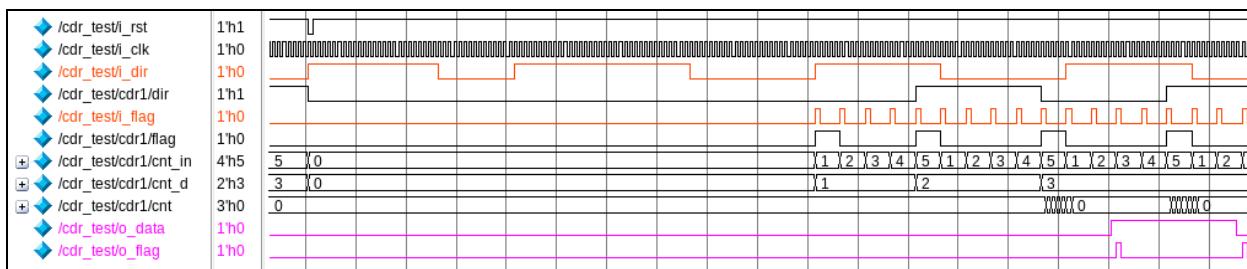


Figure 14 : Simulation pré-synthèse du CDR global, montrant le cas avec un reset et la non prise en compte du “premier” échantillon

Sur la figure 14 ci-dessus, montrant le cas du fonctionnement du CDR avec un reset et la non prise en compte du “premier” échantillon, on constate que lorsque le reset est à 0, les signaux internes reprennent leur état initial. De plus, tant qu'il n'y a pas de flag d'entrée, on n'observe pas de changement par rapport au moment du reset. Ce n'est qu'une fois le premier flag arrivé que les compteurs se mettent en marche ainsi que les autres signaux internes, ce qui permet une bonne synchronisation.

Quant à elle, la première direction n'est pas prise en compte, comme demandé par le groupe de la FIFO, ce qui est également montré sur la figure 14 ci-dessus.

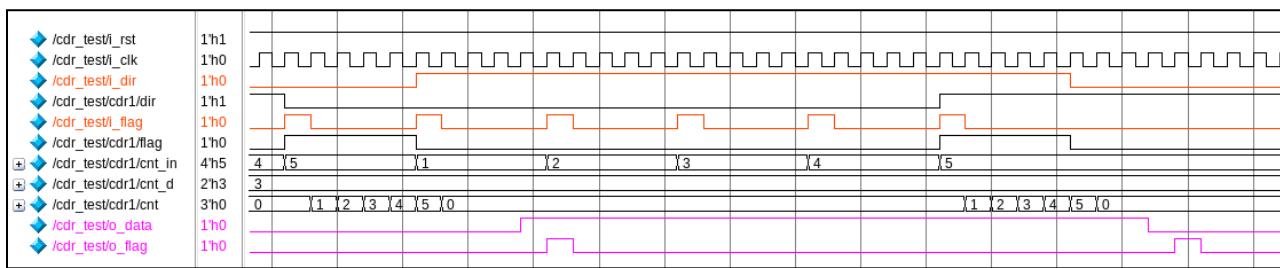


Figure 15 : Simulation pré-synthèse du CDR global, montrant le cas du “dernier” échantillon

Enfin, la figure 15 ci-dessus présente les résultats de simulation pré-synthèse du CDR, mais cette fois montrant la manière de traiter le “dernier” échantillon d'une trame. Dans le sous-bloc de l'interface, les “i_flag” sont utilisés pour incrémenter “cnt_in”. La variable “flag” est celle qui est utilisée dans le reste du CDR: il est donc primordial de la faire redescendre à 0 au bout de 100 ns. Dans le cas contraire, la sortie continuerait à mettre des flag de sortie (“o_flag”) alors qu'il n'en faut pas. Pour ce faire, grâce à “cnt”, on va compter le nombre de période d'horloge jusqu'à 5 (5 * 20ns = 100 ns). Ensuite, la valeur de “flag” est forcée à 0: s'il n'y a plus de “i_flag”, la valeur de direction n'est donc plus prise en compte.

2.5.4 Synthèse du CDR

Afin de valider le CDR mis en place, nous avons tout d'abord vérifié que le code rtl était synthétisable avec Spyglass. Ensuite, nous sommes passés à la synthèse du CDR avec Design Vision.

Dans un premier temps, nous avions choisi une fréquence de 10MHz pour traiter les données. La fréquence de travail étant de 50 MHz, nous disposions donc de 5 coups d'horloge pour échantillonner les données dans le détecteur de phase. Afin d'avoir davantage de précision sur l'opération de traitement de ce bloc, nous avions décidé de doubler ces 5 coups d'horloge, en faisant fonctionner les bascules à la fois sur front montant et front descendant. Avec Design Vision, nous nous sommes rendus compte que la synthèse comportait des bascules latchs (actives sur un état et non sur un front). En effet, il n'existe pas de composant pouvant être actif à la fois sur les 2 fronts d'horloge. C'est pourquoi nous avons modifié notre code en travaillant à une fréquence de 2MHz, à la fois pour avoir davantage de fronts d'horloge, et pouvoir travailler sur front montant OU front descendant.

clock i clk (rise edge)	20.00	20.00	Combinational area:	27991.600121
clock network delay (ideal)	0.00	20.00	Buf/Inv area:	3658.200150
phd1/cnt phd/cnt req[0]/C (DF3)	0.00	20.00 r	Noncombinational area:	16871.400085
library setup time	-0.01	19.99	Macro/Black Box area:	0.000000
data required time		19.99	Net Interconnect area:	8946.000000
-----			Total cell area:	44863.000206
data required time		19.99	Total area:	53809.000206
data arrival time		-10.83		

slack (MET)		9.16		

Figure 16 : Chemin critique (slack) du CDR et surface du circuit après synthèse

Ensuite, lors de la synthèse, nous avons vérifié que la synthèse ne générât plus de latchs, ainsi que le chemin critique (slack). Dans le pire des cas, nous avions un slack de 0.05 ns. Nous avons donc optimisé notre code, de manière à ce que toutes les parties séquentielles fonctionnent sur front positif. Ainsi, après optimisation, nous avons obtenu un slack de 9 ns dans le pire des cas, ce qui est largement suffisant. La surface totale occupée par le bloc du CDR est de $53809 \mu\text{m}^2$.

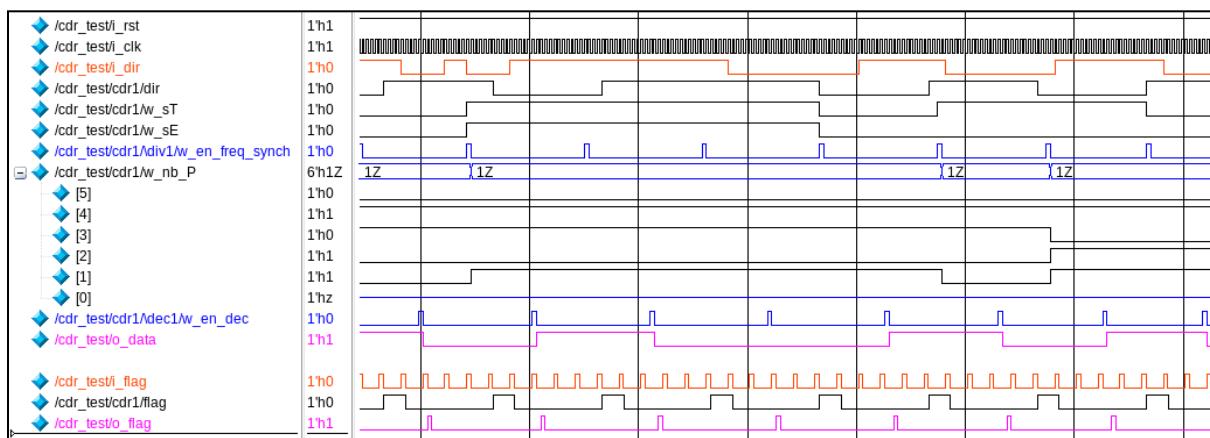


Figure 17 : Simulation post-synthèse du système global

Sur la figure 17 ci-dessus présentant nos résultats post-synthèse, on peut constater que la sortie o_data est bien échantillonnée, malgré le retard induit. Le signal w_nb_P[0] est sur Z car lors de la synthèse une optimisation a été faite et un nom différent lui a été donné. A cause du grand nombre de signaux générés, nous n'avons simplement pas pu retrouver le nom réel de ce signal.

2.6. Wrappers (Axel Baldacchino, Mamadou Hawa Diallo)

2.6.1. Émetteur Zigbee - Partie Numérique

Comme nous l'avons décrit précédemment, pour la partie numérique, la chaîne d'émission est composée de l'interface externe, modélisée par la FIFO, ainsi que du CODEUR IQ (Figure ci-dessous).

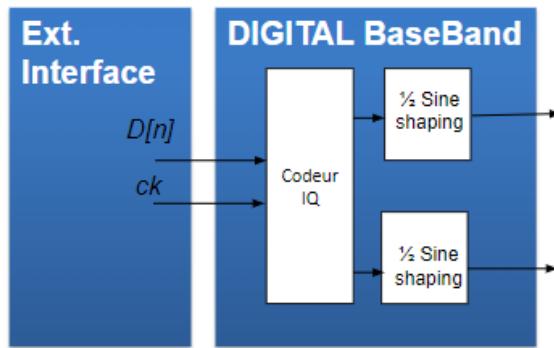


Figure 1: Schématisation de la chaîne de transmission

Pour valider le fonctionnement de la chaîne d'émission numérique, nous avons construit un wrapper (concaténation de plusieurs descriptions RTL). Le but est de simuler les interactions entre les deux blocs. Dans notre cas, nous voulons vérifier que lorsque la FIFO contient à minima une donnée, elle la transmet bien bit par bit au CODER sur une demande de celui-ci.

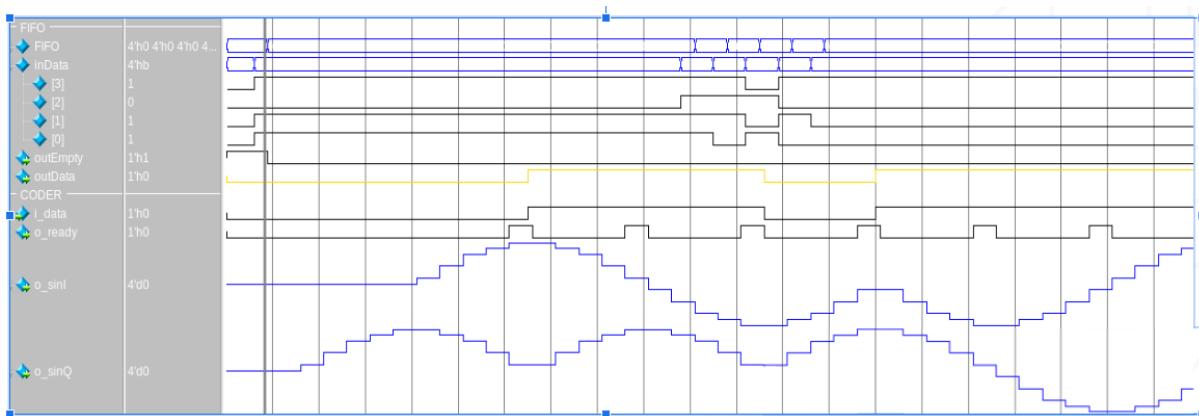


Figure 2: Simulation du wrapper

La Figure ci-dessus correspond à la simulation du wrapper de la chaîne d'émission. Nous pouvons voir que dès lors que le signal *outEmpty* est à 0, la FIFO contient au moins une valeur de 4 bits, et sur une requête de lecture dans la FIFO du CODER via le signal *o_ready*, le signal de sortie *o_data* transmet le premier bit de la donnée 4 bit au CODER.

Dès lors que le CODER reçoit une première donnée, il commence son traitement pour construire les sinus I et Q.

2.6.2. Récepteur Zigbee - Partie Numérique

Une fois chaque bloc validé par le biais de simulations Pré et Post-Synthèse, nous les regroupons afin de constituer la partie numérique de la chaîne de réception du Zigbee. De ce fait, tel que sur la *Figure* ci-dessous, elle comprend : le démodulateur IQ incluant la fonctionnalité de réjection de fréquence image, le CORDIC, le CDR et l'interface externe Rx.

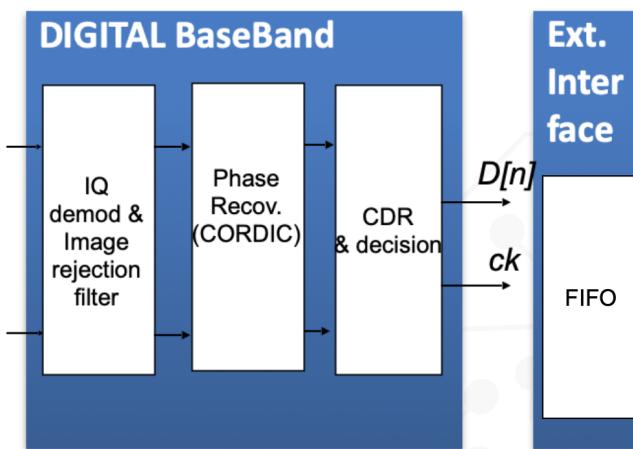


Figure 3 : Partie numérique de la chaîne de réception du zigbee

Pour valider le fonctionnement de cette chaîne de réception numérique, nous partons du testbench du démodulateur IQ avec, en entrées, les signaux I et Q à la fréquence intermédiaire extraits de Matlab.

Ainsi, nous nous sommes vite aperçus de l'existence d'erreurs générées liées à l'imprécision des signaux I_BB_postfilter et Q_BB_postfilter ramenés sur 4 bits avant d'être transmis au CORDIC. En conséquence, tel que sur la *Figure* ci-dessous, le CORDIC envoie les données à une fréquence variable (période de 100, 400 ou 600 ns), alors que le fonctionnement du CDR est basé sur la fréquence d'envoi des données à 2 MHz (soit 500 ns).

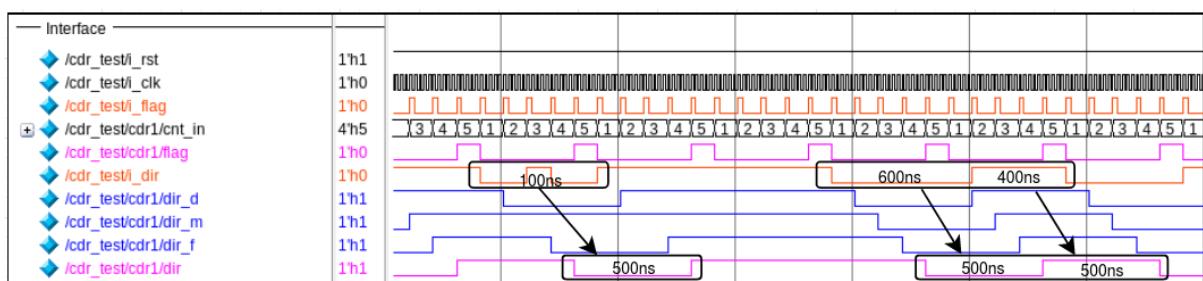


Figure 4: Simulation pré-synthèse du CDR, prenant en compte ces erreurs de précision

De ce fait, le binôme travaillant sur le CDR a dû modifier son code rtl, afin de prendre en compte ces erreurs. La *Figure* ci-dessus présente une simulation de ces erreurs, lors de l'envoi des données, corrigées par le CDR. On voit que le "glitch" de 100ns n'est pas pris en compte tandis que les données de 400 et 600 ns sont modifiées pour durer 500 ns.

Une fois ces corrections effectuées, nous obtenons alors les résultats suivants :

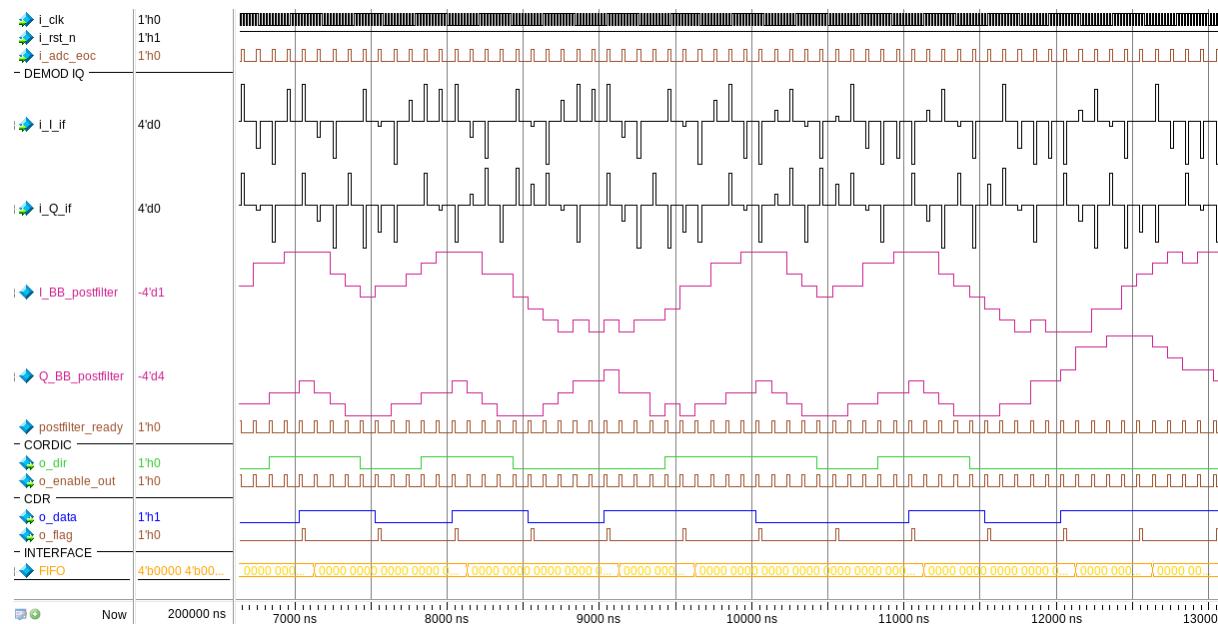


Figure 5 : Validation de la chaîne de réception numérique

Ainsi, la validation de la chaîne de réception numérique s'effectue à 3 niveaux :

- Chaque binôme s'assure que son bloc se comporte bien comme il se doit. Par exemple, nous constatons que les signaux **I_BB_postfilter** et **Q_BB_postfilter** sont bien ramenés en bande de base à 500 KHz et que l'influence de la fréquence image a été rejetée ;
- Nous vérifions que les signaux d'interaction entre blocs, en **marron**, sont synchronisés et que les blocs communiquent bien entre eux ;
- Enfin, nous comparons le contenu de la FIFO ci-dessous à la trame envoyée depuis Matlab et nous constatons que les deux sont semblables.

```
sim:/tbench/decoder_cordic_cdr_ei/u_fifo/FIFO @ 278815380 ps
127 : 0101 0101 0101 0111 1010 1111 1101 0110
119 : 0011 1110 1011 1011 1010 0000 0111 1101
111 : 0110 1010 1000 0000 1001 0110 1110 0101
103 : 0110 0101 0101 0101 1101 1111 0110
95 : 1011 0001 1111 1101 0101 0101 1000 1011
87 : 0110 0011 0101 0100 1000 0100 0011 1111
79 : 0010 1011 1010 1010 0011 1101 1111
71 : 0110 1011 0001 1111 1101 0101 0101 1000
63 : 1011 0110 0011 0101 0100 1000 0100 0011
55 : 1111 0010 1011 1010 1010 1010 1000 1110
47 : 0111 1011 1101 1000 1111 1110 1010 0010
39 : 1100 0101 1011 1001 0010 0010 0100 1010
31 : 1001 0111 1001 0101 0101 1101 1001
23 : 1110 0111 1011 1001 1000 1111 1110 1010
15 : 0010 1100 0101 1011 1001 0010 0010 0100
7 : 1010 1001 0111 1001 0101 0101 0101 0110
```

Figure 6: Contenu de la FIFO

Également, nous comparons la séquence binaire envoyée depuis Matlab à la séquence binaire reçue, en sortie du CDR, après la reconstruction des bits en sortie.

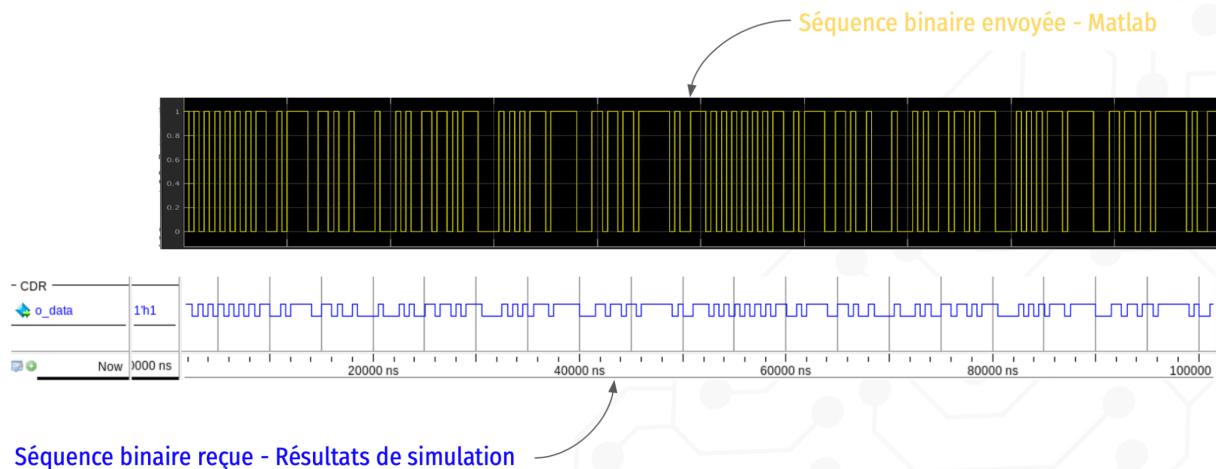


Figure 7 : Comparaison de la séquence binaire envoyée à celle reçue

Nous pouvons alors constater que les deux sont identiques. Ainsi, cela nous permet de valider le fonctionnement de la partie numérique de la chaîne de réception du Zigbee.

2.7 TOP (Tom Désesquelle)

2.7.1 Spécification et implémentation RTL

2.7.1.1 Contexte

La puce numérique doit respecter certaines contraintes. Parmi ces contraintes, nous avons affaire à un nombre limité de *pins* (48 disponibles). Sur les 48 *pins* disponibles, il nous est imposé de disposer d'un *pin* pour l'horloge, d'un *pin* pour la remise à zéro (reset) et d'un minimum de quatre *pins* pour les alimentations (2 VDD et 2 GND).

Nous nous sommes rajoutés des contraintes supplémentaires : à savoir que nous voulons tester toute la chaîne de transmission assemblées et la chaîne de réception assemblées. A cela, nous rajoutons la volonté de tester chaque bloc numérique unitairement. Enfin, nous voulons avoir la possibilité de connecter les sorties du CODER sur les entrées du CORDIC, pour quand même tester la chaîne de réception si le DECODER est amené à ne pas fonctionner.

2.7.1.2 Solution d'assemblage du TOP

Pour satisfaire ces contraintes, nous avons utilisé des multiplexeurs et des démultiplexeurs pour augmenter significativement le nombre de signaux que nous pouvons piloter. Ces multiplexeurs et démultiplexeurs nous permettent, dans un même temps, d'isoler chacun des blocs mais aussi de tester les chaînes de transmission et de réception complètes.

Une première solution a été menée. Celle-ci comportait 4 démultiplexeurs et 14 multiplexeurs, de taille variable (1:2, 1:4, 1:8). L'ensemble de ces multiplexeurs et démultiplexeurs étaient pilotés par 9 sélecteurs, pour un total de 16 *pins* parmi les 48 disponibles.

Cette première solution nous a conduit à un circuit de 48 *pins* avec le minimum requis pour les alimentations et embarqués le minimum requis pour chacun des blocs. En plus de cela, le test du circuit, aussi bien au complet qu'unitairement, était fastidieux.

C'est pourquoi, nous avons réalisé une deuxième version du TOP. Cette deuxième version utilise toujours 4 démultiplexeurs et 14 multiplexeurs, mais la taille a été fixée (1:8). Fixer la taille des démultiplexeurs et des multiplexeurs nous a permis de rassembler les sélecteurs de ces derniers pour rendre le test bien plus simple et, par la même occasion, d'économiser des *pins*. Nous sommes passés de 48 *pins* essentiels à 38 *pins* essentiels. Nous avons donc ajouté aux entrées/sorties du circuit 6 bits provenant des registres de status et d'erreurs des deux interfaces FIFO. Nous avons aussi ajouté deux alimentations supplémentaires, pour un total final de 4 VDD et 4 GND.

L'architecture retenue (deuxième solution) est présentée en figure 1. Les tableaux décrivent les blocs testés en fonction des sélecteurs.

2.7.2 Testbench, plan de validation (spécifications système)

L'objectif de la simulation du TOP est de vérifier que chaque bloc puisse être testé singulièrement, mais aussi que chaque bloc puisse être interconnecté avec les blocs annexes. Dans cette partie de simulation, il n'est pas question de valider ni le fonctionnement de chaque bloc, ni le fonctionnement entre les blocs. Ces validations ont déjà été menées lors de la conception RTL et de la mise en commun des blocs dans des wrappers. En d'autres termes, l'objectif de la simulation TOP est de vérifier que chacun des *pins* qui va composer la puce

numérique est capable de piloter tous les signaux des blocs composant notre émission (inFIFO, CODER) et notre transmission (DECODER, CORDIC, CDR, outFIFO).

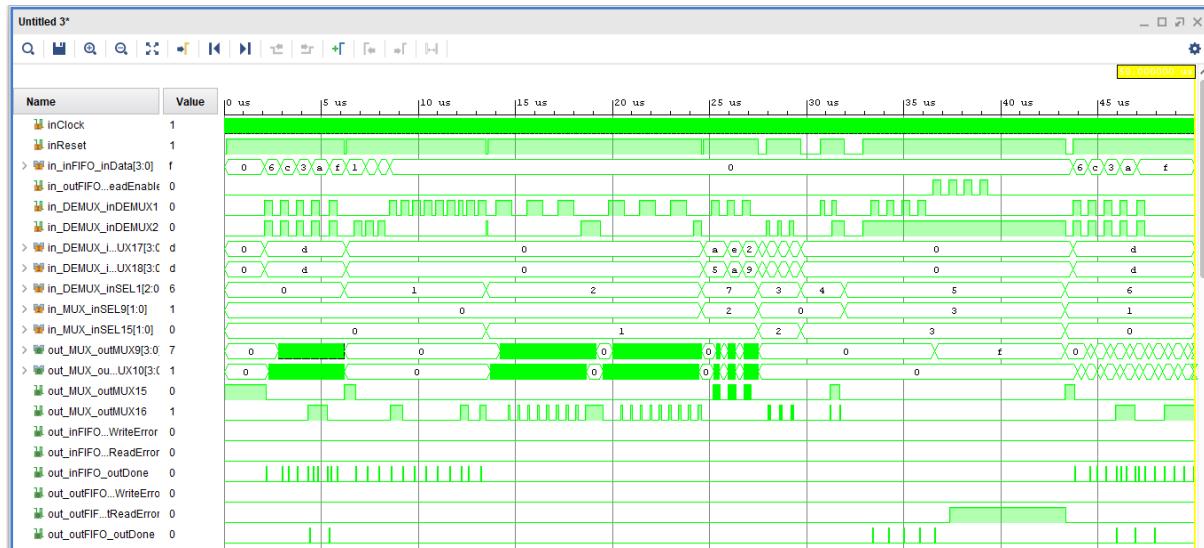


Figure 1 : Simulation des pins d'entrées/sorties du TOP visant à tester toutes les combinaisons possibles

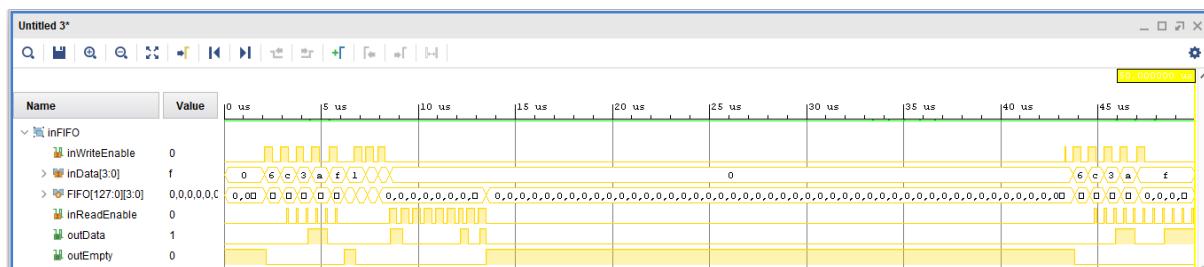


Figure 2 : Simulation de la chaîne de transmission, une fois le TOP assemblé avec MUX et DEMUX

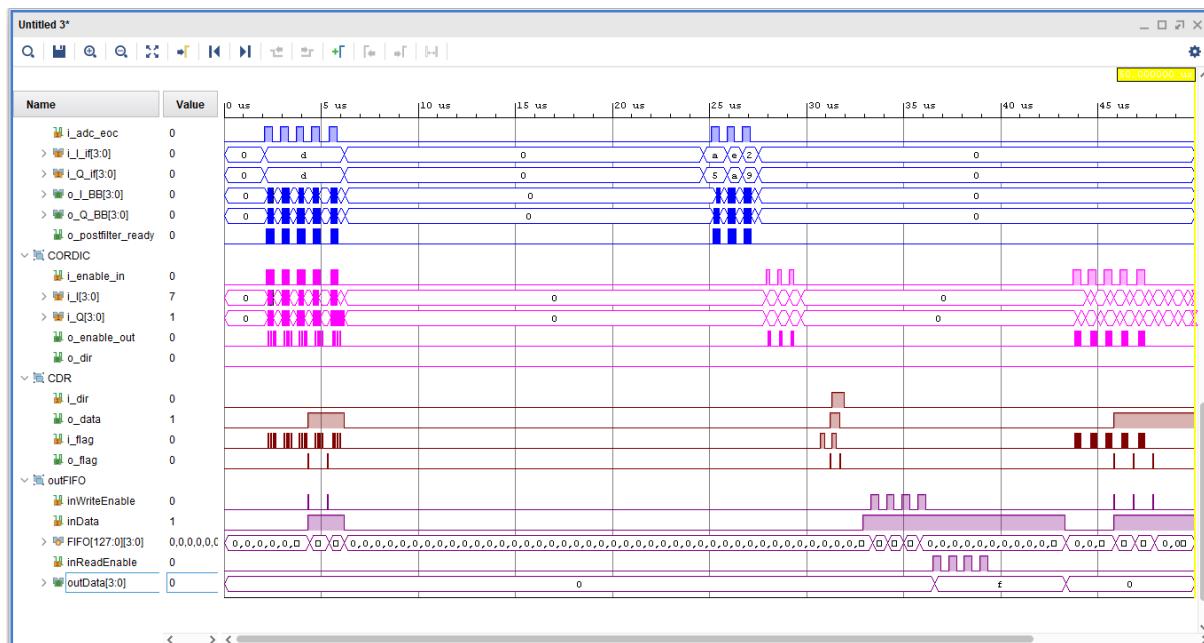


Figure 3 : Simulation de la chaîne de réception, une fois le TOP assemblé avec MUX et DEMUX

Les figures 1, 2 et 3 valident le pilotage des données depuis les entrées de la puce numérique jusque dans les blocs qui la composent. Le pilotage des données vérifient les contraintes que nous nous sommes fixées : test des chaînes de transmission/réception, test unitaire de chaque bloc et test de la chaîne de transmission + réception avec bouclage du CODER sur le CORDIC.

2.7.3 Synthèse

A la suite de l'étape de synthèse, nous obtenons exactement le même résultat de simulation qu'en pré-synthèse (figure 4).

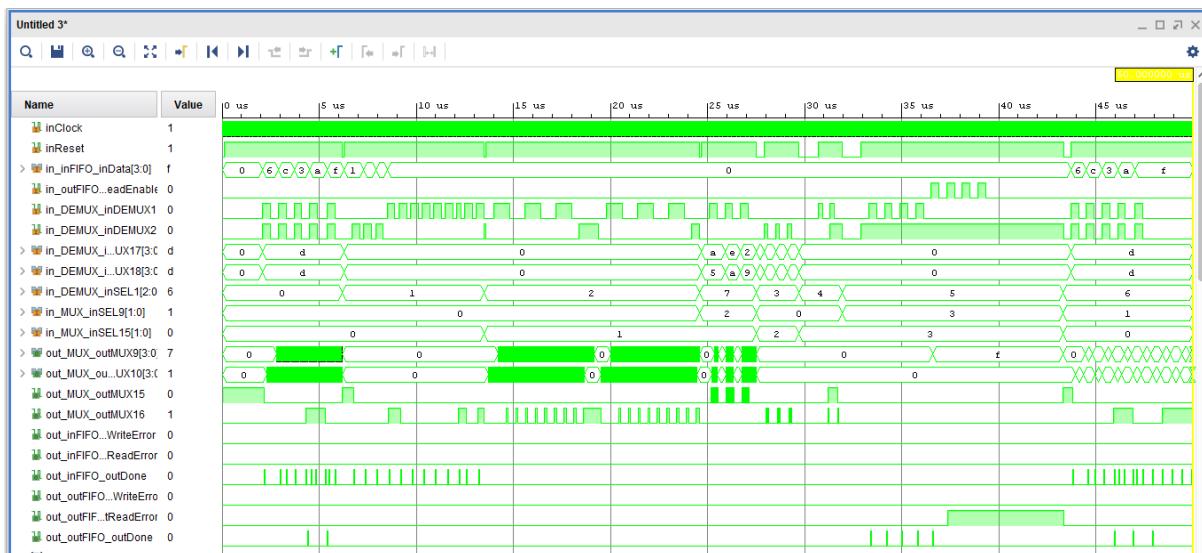


Figure 4 : Simulation post-synthèse de l'assemblage TOP

Nous avons ainsi pu générer les rapports de surface et de *timing* pour chacun de nos deux designs.

La place occupée par le design est d'environ 1 mm^2 et nous avons un *slack* de 5 ns lorsque nous réalisons une synthèse dans le pire des cas (WORST). Nous avons donc bon espoir que nos blocs soient suffisamment bien conçus pour ne pas avoir à les modifier/optimiser lors de l'étape de placement/routage qui va engendrer une dégradation du *slack*.

Number of ports:	43
Number of nets:	8288
Number of cells:	7363
Number of combinational cells:	6050
Number of sequential cells:	1313
Number of macros/black boxes:	0
Number of buf/inv:	1735
Number of references:	37
Combinational area:	510983.202934
Buf/Inv area:	66557.402138
Noncombinational area:	366675.400879
Macro/Black Box area:	0.000000
Net Interconnect area:	150219.000000
Total cell area:	877658.603813
Total area:	1027877.603813

clock inClock (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
u_coder/j_reg[0]/C (DF3)	0.00 #	0.00 r
u_coder/j_reg[0]/Q (DF3)	1.01	1.01 f
U1589/Q (BUF2)	1.03	2.03 f
u_coder/add_282/U1_1/1/C0 (ADD22)	0.94	2.97 f
u_coder/add_282/U1_1/2/C0 (ADD22)	0.63	3.60 f
u_coder/add_282/U1_1/3/C0 (ADD22)	0.63	4.23 f
u_coder/add_282/U1_1/4/C0 (ADD22)	0.63	4.86 f
u_coder/add_282/U1_1/5/C0 (ADD22)	0.63	5.49 f
u_coder/add_282/U1_1/6/C0 (ADD22)	0.63	6.12 f
u_coder/add_282/U1_1/7/C0 (ADD22)	0.63	6.75 f
u_coder/add_282/U1_1/8/C0 (ADD22)	0.63	7.38 f
u_coder/add_282/U1_1/9/C0 (ADD22)	0.63	8.01 f
u_coder/add_282/U1_1/10/C0 (ADD22)	0.63	8.64 f
u_coder/add_282/U1_1/11/C0 (ADD22)	0.63	9.26 f
u_coder/add_282/U1_1/12/C0 (ADD22)	0.63	9.89 f
u_coder/add_282/U1_1/13/C0 (ADD22)	0.63	10.52 f
u_coder/add_282/U1_1/14/C0 (ADD22)	0.63	11.15 f
u_coder/add_282/U1_1/15/C0 (ADD22)	0.63	11.78 f
u_coder/add_282/U1_1/16/C0 (ADD22)	0.63	12.41 f
u_coder/add_282/U1_1/17/C0 (ADD22)	0.63	13.04 f
u_coder/add_282/U1_1/18/C0 (ADD22)	0.61	13.65 f
U1588/Q (XNR21)	0.65	14.30 f
u_coder/U155/Q (OAI222)	0.38	14.68 r
u_coder/j_reg[19]/D (DF3)	0.00	14.68 r
data arrival time		14.68

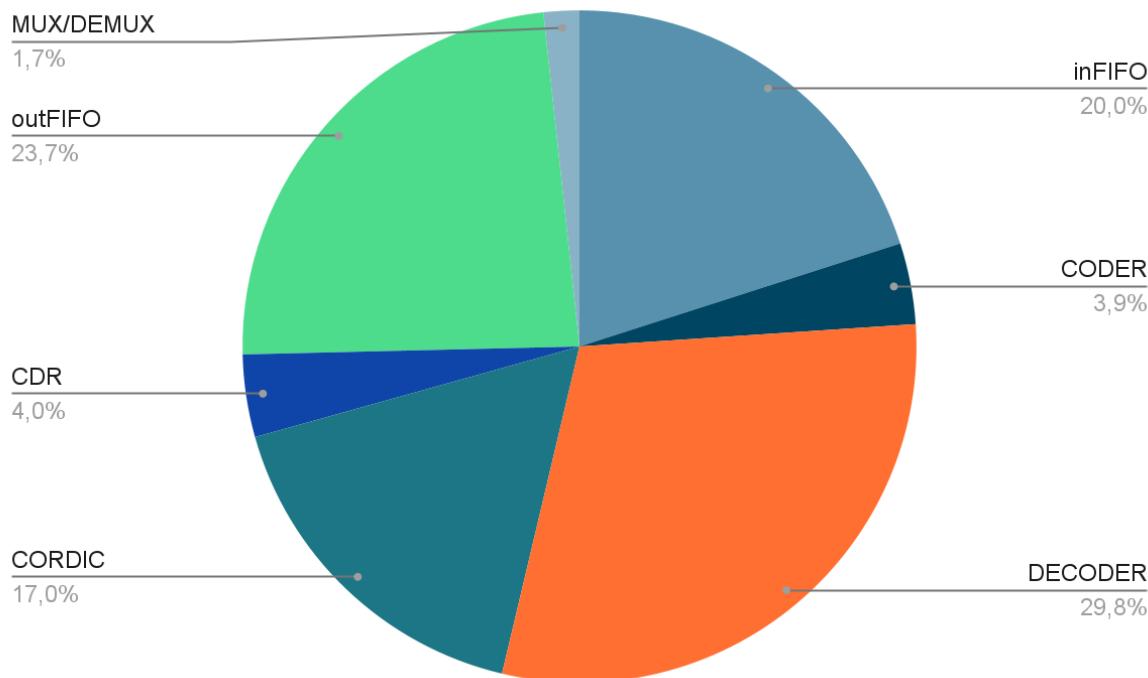
clock inClock (rise edge)	20.00	20.00
clock network delay (ideal)	0.00	20.00
u_coder/j_reg[19]/C (DF3)	0.00	20.00 r
library setup time	-0.01	19.99
data required time		19.99

data required time		19.99
data arrival time		-14.68

slack (MET)		5.31

Figure 5 et Figure 6: Rapport de surface et rapport de timing de l'interface outFIFO

A titre d'information, le graphique 1 recense le pourcentage de place occupée par chacun des blocs de la puce numérique.



Graphique 1: Pourcentage de place occupée par chacun des blocs

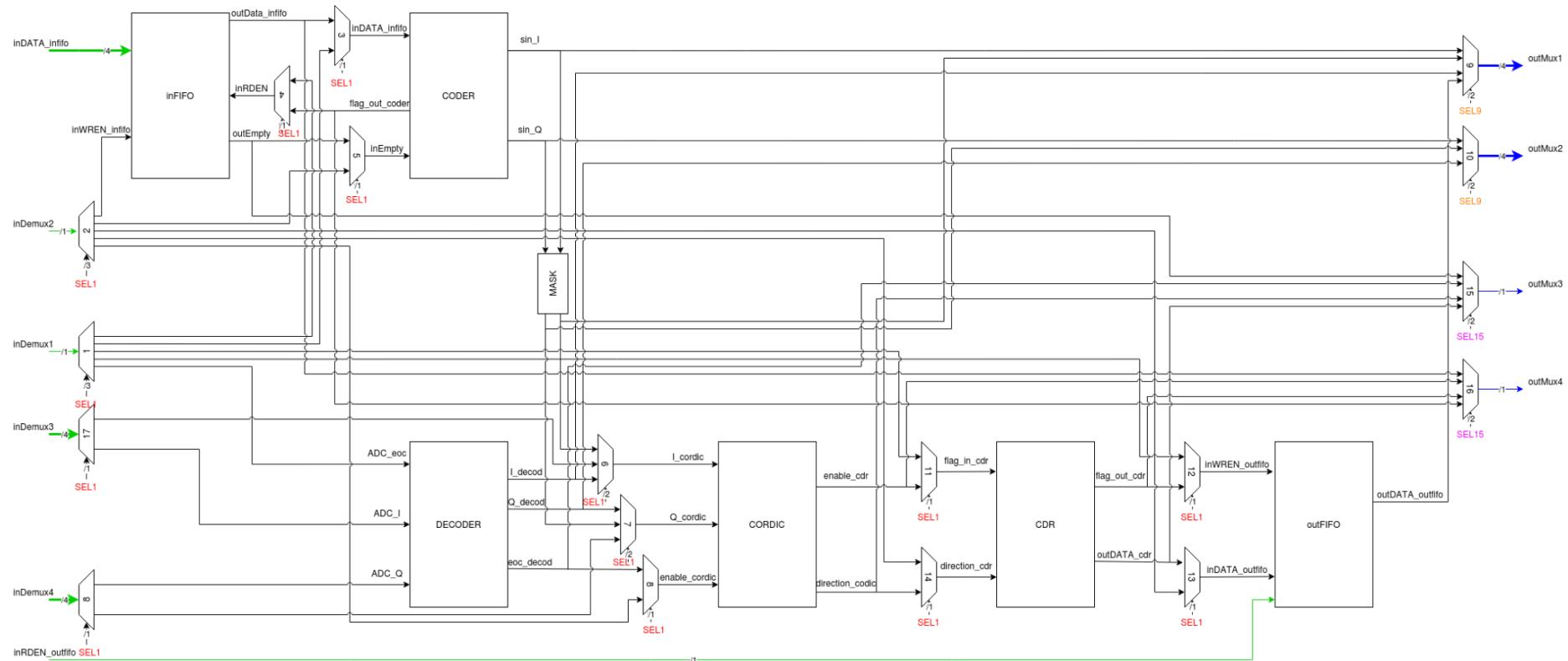


Figure 7 : Schéma Top Zigbee Numérique

SEL1															
CODE	DEMUX1	DEMUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX8	MUX11	MUX12	MUX13	MUX14	DEMUX17	DEMUX18	
0	Decod	inFIFO	Coder inFIFO	inFIFO Coder	inFIFO Coder	Decod Cordic	Decod Cordic	Decod Cordic	Cordic CDR	CDR outFIFO	CDR outFIFO	Cordic CDR	Decod	Decod	
1	inFIFO	0	inFIFO	0	1	0	0	0	0	0	0	0	0	0	
2	Coder	Coder	0	Coder	Coder	0	0	0	0	0	0	0	0	0	
3	0	Cordic	0	0	1	Cordic	Cordic	Cordic	0	0	0	0	Cordic	Cordic	
4	CDR	CDR	0	0	1	0	0	0	CDR	0	0	CDR	0	0	
5	outFiFo	outFiFo	0	0	1	0	0	0	0	outFiFo	outFiFo	0	0	0	
6	Cordic	inFIFO	Coder inFIFO	inFIFO Coder	inFIFO Coder	Mask	Mask	Mask	Cordic CDR	CDR outFIFO	CDR outFIFO	Cordic CDR	0	0	
7	Decod	0	0	0	1	0	0	0	0	0	0	0	Decod	Decod	

Tableau 8 : Setup multiplexeurs en fonction du code appliqué au niveau du sélecteur d'entrée SEL1

SEL9			SEL15		
CODE	MUX9	MUX10	CODE	MUX15	MUX16
0	coder_outSinI	coder_outSinQ	0	inFIFO_outEmpty	inFIFO_outData
1	coder_outSinIMasked	coder_outSinQMasked	1	decod_outEOC	cordic_outReady
2	decod_outI	decod_outQ	2	cordic_outDirection	cordic_outEnable
3	outFiFo_outData		3	cdr_outData	cdr_outFlag

Tableau 9 : Setup multiplexeurs en fonction du code appliqué au niveau des sélecteurs de sortie SEL9 et SEL15

2.8 PLACE & ROUTE TOP (Anthony Teissier, Axel Baldacchino)

Le flow de conception

L'objectif de cette partie est de décrire comment fonctionne le flow de conception de notre module et comment nous avons résolu les problèmes auxquels nous avons pu faire face. Nous avons repris le flow débuté dans la partie codeur puis nous l'avons adapté à la netlist du top avec le système de test. Nous avons choisi de prendre la netlist et de dérouler le flow en Worst Corner afin de rendre le projet plus réel et espérer avoir une puce fonctionnelle lors des tests si nous parvenons à la faire fabriquer.

Le flow de conception se déroule de la façon suivante :

- **Floorplan** : Création des pads, dimensionnement du core, création de la grille d'alimentation + connexion aux pads, connexion des pins d'alimentation.

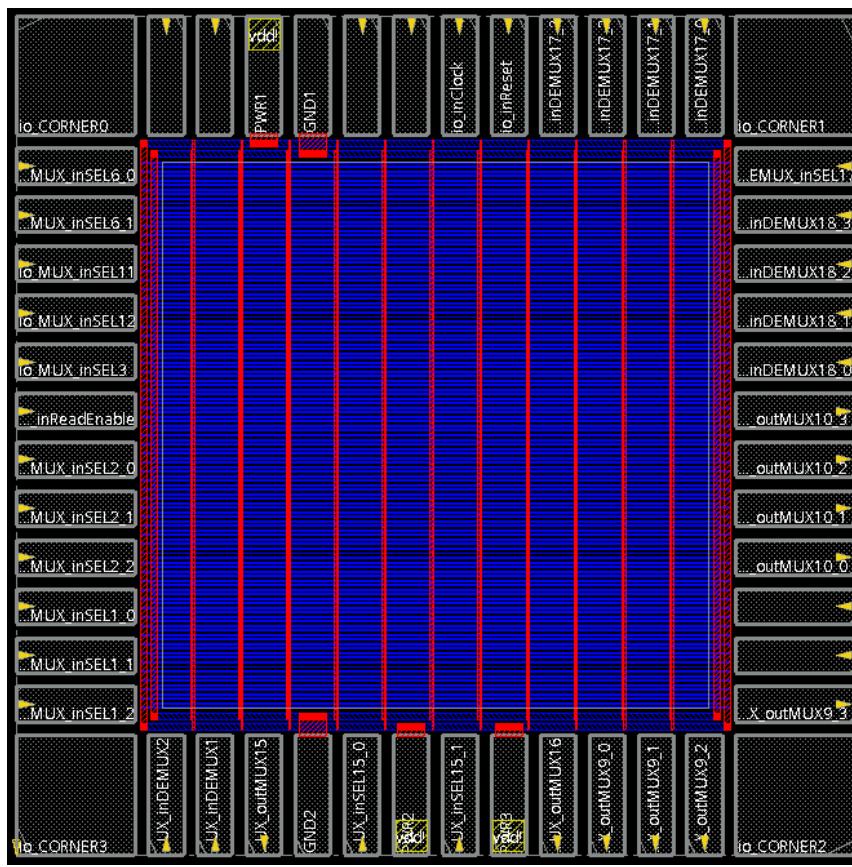


Figure 1 : Floorplan du top

La dimension de la puce est de $2,4 \mu\text{m} \times 2,4 \mu\text{m}$ (nous vous expliquerons pourquoi une telle taille dans la partie DRC), avec une grille d'alimentation dessinée sur les métaux 1 et 2. Les stripes VDD/GND en métal 2 sont espacés de $125 \mu\text{m}$ avec une largeur de $5 \mu\text{m}$ et un espacement de $0,5$

µm entre un stripe de GND et un stripe de VDD. Comme nous l'avons précisé plus haut dans la partie dédiée au codeur, nous générerons la grille d'alimentation de façon automatique en fonction de la taille du core, le nombre de stripe nécessaire est calculé seul.

Cette partie est décrite dans un script appelé design_config.tcl.

- **Placement :** Placement des standard cells

Une fois la partie floorplan terminée, nous pouvons passer au placement des standard cells.

Avant de lancer la commande de placement, nous ajoutons les endcaps sur les côtés gauche et droite du core. Ceux-ci permettent de mettre des cellules de fin permettant de réduire l'effet capacatif des standard cells situées en bout de core.

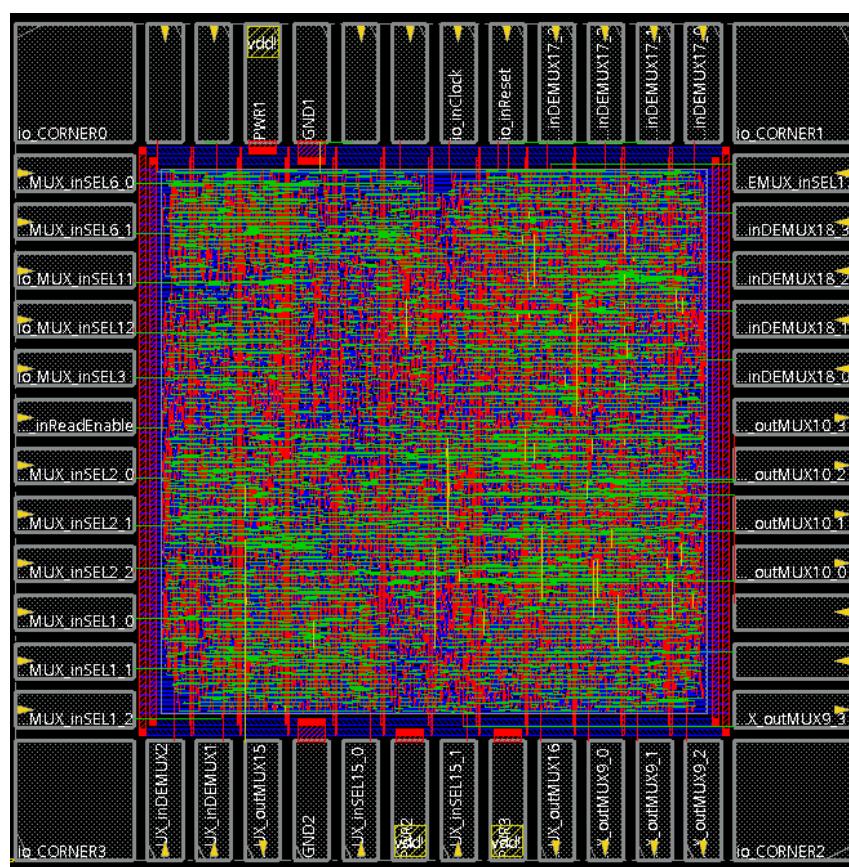


Figure 2 : Placement des standard cells

Les standard cells sont la partie logique de la puce qui a été décrite en system verilog dans les étapes précédentes. Dans cette étape, Innovus cherche à placer les standard cells de manière à violer le moins possible le timing. L'objectif est de ne pas avoir de slack négatif, Innovus va donc faire plusieurs essais de placement afin de trouver le meilleur compromis. Cela dure environ une quinzaine de minutes dans notre cas.

Finalement, nous obtenons un slack de -0,001 ns en setup avec un uncertainty de 10 ns. L'uncertainty contenant le jitter d'horloge et le skew de l'horloge pour la technologie C35, nous

pouvons en déduire que nous sommes MET de 9,999 ns. Ceci représente une moitié de période de marge pour une horloge comme nous à 50 MHz soit une période de 20ns.

Pour en arriver à ce résultat, nous avons créé ce que nous appelons des “group path” entre les chemins qui étaient critiques. Ainsi en spécifiant les chemins qui doivent être routé ensemble, Innovus travail plus dessus et optimise encore le timing.

Pour le placement, nous n'avons d'ailleurs pas utilisé la commande placeDesign -fp comme précisé dans le TP mais place_opt_design avec du useful skew et un OCV (On Chip Variation). Le useful skew permet d'utiliser l'avance sur un chemin de data pour compenser un autre chemin violé en timing en retardant l'horloge sur la flop de capture. L'OCV sert quant à lui à prendre en compte un mode de fonctionnement des transistors différent en fonction de leur emplacement sur la puce et par rapport à la grille d'alimentation.

Nous avons une densité assez faible d'environ 50 %.

Cette étape est réalisée dans le flow par le script placement.tcl.

- **Clock Tree Synthesis** : Création de l'arbre d'horloge

Nous réalisons un circuit synchrone, ce qui signifie que l'horloge doit être routée sur les standard cells. Pour cela, nous générions ce qui s'appelle un arbre d'horloge qui va venir du pad de l'horloge jusqu'aux pins des standard cells. Innovus va ensuite bufferiser certains chemin afin que toutes les flip flop reçoive le signal d'horloge quasiment en même temps.

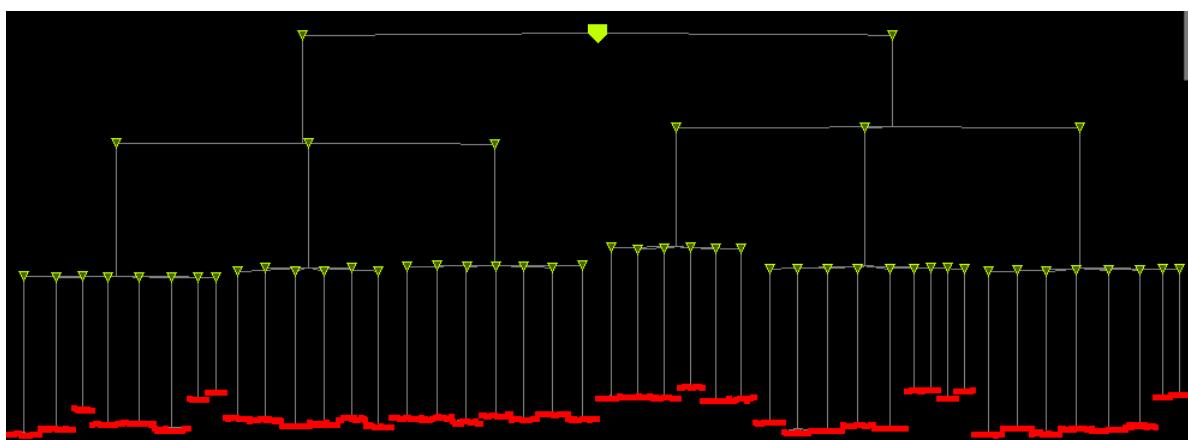


Figure 3 : Arbre d'horloge

Sur la capture ci-dessus, nous observons que la répartition des flip flop est assez uniforme (Les parties rouges (les flip flop) sont presque alignées). L'horloge arrive presque en même temps sur chaque flip flop.

Ainsi, nous obtenons les résultats suivants :

```

Endpoint: t_op/u_decoder/fir_filter/Q_data_add_7_buff_reg[13]/D (^) checked
with leading edge of 'inClock'
Beginpoint: t_op/u_decoder/fir_filter/Q_data_mult_8_buff_reg[0]/QN (^)
triggered by leading edge of 'inClock'
Path Groups: {reg2reg}
Analysis View: setup_func_max
Other End Arrival Time      -0.136
- Setup                      0.025
+ Phase Shift                20.000
- Uncertainty                 10.000
= Required Time               9.839
- Arrival Time                10.485
= Slack Time                  -0.645
    Clock Rise Edge           0.000
    + Clock Network Latency (Prop) 0.079
= Beginpoint Arrival Time     0.079
  
```

Figure 4 : Report timing en setup (MET)

Nous remarquons que nous sommes passés d'un slack de -0,001 ns à un slack de -0,645 ns. Néanmoins, nous avons toujours l'uncertainty de 10 ns, ce qui signifie que nous sommes encore MET de 9,355 ns en setup ce qui est largement suffisant.

Nous pouvons maintenant regarder si nous avons des violations de timing en hold

```

Endpoint: t_op/u_outFIFO/os1/dff1/s_qout_reg/D (^) checked with leading edge
of 'inClock'
Beginpoint: in_outFIFO_inReadEnable          (^) triggered by leading edge
of '@'
Path Groups: {in2reg}
Analysis View: hold_func_min
Other End Arrival Time      0.020
+ Hold                      0.014
+ Phase Shift                0.000
= Required Time               0.033
Arrival Time                 0.188
Slack Time                   0.155
    Clock Rise Edge           0.000
    + Input Delay              0.000
= Beginpoint Arrival Time     0.000
  
```

Figure 5 : Report timing en hold (MET)

Comme nous pouvons le remarquer sur la figure ci-dessus, nous sommes MET de 0,155 ns ce qui est très bien.

La génération du clock tree se déroule dans le script `clock_tree_synthesis.tcl`.

- Ajout des fillers : Ajout des fillers

Le but des fillers est de remplir les espaces qui peuvent se créer entre les standard cells. Les IOFillers servent aussi à faire la continuité des alimentations VDD et GND.

Cette étape se déroule dans le script `add_fillers.tcl`.

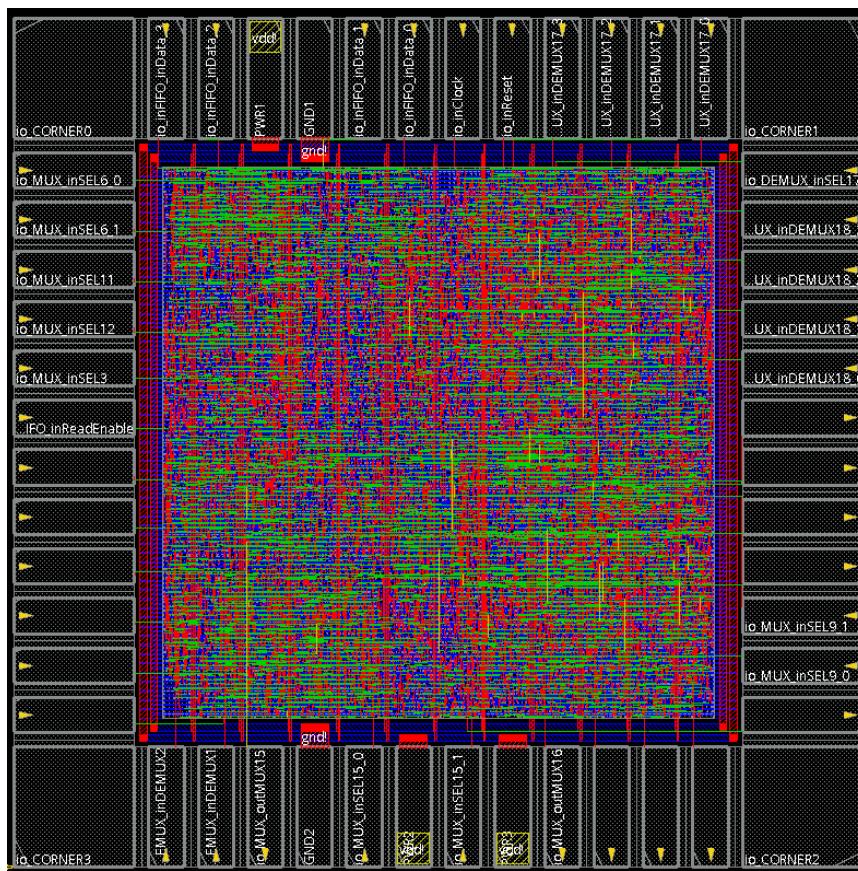


Figure 6 : Ajout des fillers

- **Routage :** routage final

La dernière étape est l'étape de routage. Celle-ci comporte un routage dit global puis un routage plus "détaillé". Ceci se fait avec la commande routeDesign.

Le problème est que lorsque nous arrivons à cette étape, la commande routeDesign ne parvient pas à router toutes les pins des standard cells.

Nous avons alors trouvé la commande pinAnalysis qui fait en sorte de tirer les connexion sur les pins. Seulement, vu que celles-ci tirent tout droit, nous nous retrouvons avec des court-circuits sur les signaux. et les alimentations.



Figure 7 : Exemple de routage avec les commandes routeDesign et pinAnalysis

Sur la figure ci-dessus, on peut distinguer un court-circuit en-bas à droite. De plus, les pistes sont routées dans les deux directions (horizontale et verticale) sur la même couche de métal.

Chaque script est ensuite appelé dans un script global contenant le routage et des optimisation nommé flow.tcl.

Les erreurs de DRC (Design Rules Check)

Lorsque nous cherchions à corriger les erreurs de DRC, nous avions demandé au CMP (maintenant Euronext) sur quelles erreurs fallait-il que nous nous préoccupions en priorité ? Ce à quoi on nous avait répondu les erreurs de spacing, de largeurs de métaux et d'overlap.

Nous avions aussi des problèmes de DRC sur les fillers qui n'était pas assez grand ou inversement, pas assez petit, pour faire la connexion entre tous les pads de la die. C'est-à-dire qu'il y a plusieurs tailles prédéfinies de fillers et que si l'espace entre les pads n'est pas un multiple de ces tailles, alors le filler ne pourra pas remplir tout l'espace. Nous avons donc modifié les dimensions du core de façon à pouvoir faire rentrer un filler de plus entre chaque pad.

Pour les erreurs de DRC liées au spacing, nous avons tenté beaucoup de commandes pour modifier les largeurs de piste et de vias seulement sans jamais vraiment de succès. Nous avons aussi tenté de modifier la grille de track qui permet de guider le routage des pistes. Seulement, les pins des standard cells ne tombaient plus sur les points d'intersection de la track. De plus même si nous avions corrigé quelques erreurs de spacing, nous avions toujours le problème du routeDesign qui ne s'effectuait pas correctement.

Sur la figure ci-dessous, vous pouvez voir la grille de track en jaune.

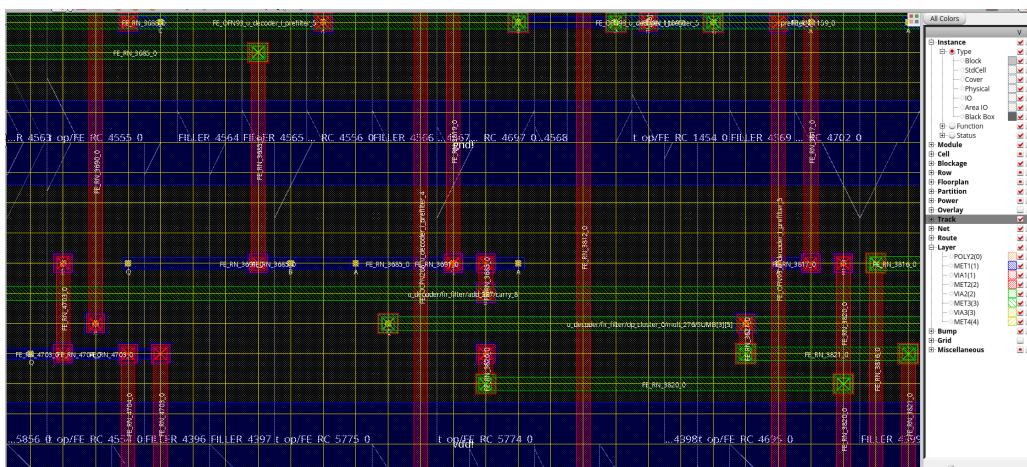


Figure 8 : Représentation de la grille de track en jaune

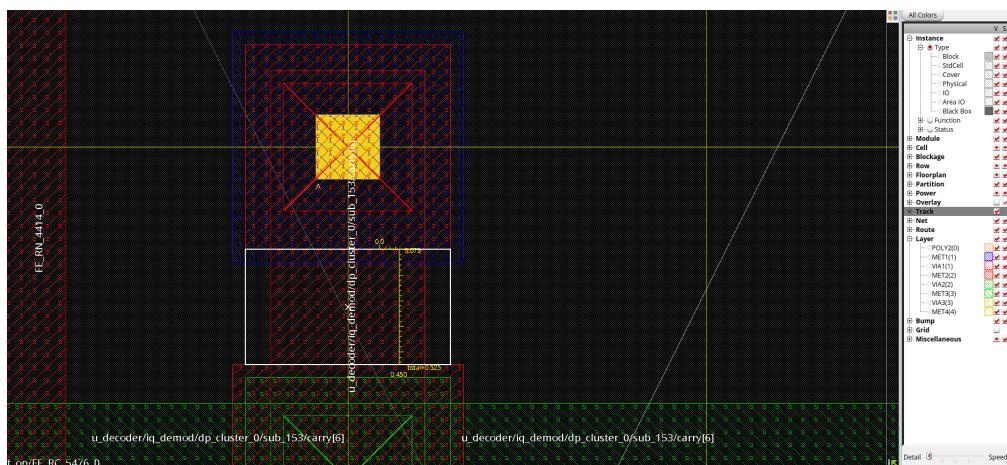


Figure 9 : Erreur de DRC liée au spacing entre 2 vias du métal 2

Sur la figure ci-dessus, vous pouvez voir une erreur de DRC liée au spacing entre 2 vias du métal 2. En effet, pour le métal 2, il faut un espacement de minimum 0,45, hors ici nous sommes légèrement en-dessous.

Récapitulatif des règles d'espacement à respecter :

-Spacing entre les pistes (μm) : METAL 1 > 0,45

METAL 2 > 0,5

METAL 3 > 0,6

METAL 4 > 0,6

-Spacing entre les vias (μm) : VIA 1 > 0,45

VIA 2 > 0,45

VIA 3 > 0,45

Pour conclure, malgré nos longues nuits passées au CIME, nous n'avons pas réussi à résoudre toutes les erreurs de DRC qui se comptent à environ 5747. Mais nous avons appris énormément de choses en l'espace de ce très court intervalle de temps et sommes prêts à revenir la dernière semaine avec de nouvelles idées pour, nous l'espérons, finir le projet.