**Compte rendu de TP Filtre RIF**

Yann DEBAIN Maxime NABON

2A-SICOM

Mardi 13 Mars 2018

**I - Introduction …………………………………………………………………………………...1**

**II - FSM globale et chronogramme** **…………………………………………………………...1**

**III - Simulation …………………………………………………………………………………….5**

**IV - Synthèse sur FPGA ………………………………………………………………………....7**

**V - Placement-Routage sur FPGA …………………………………………………………...10**

**VI - Mesure sur la carte ………………………………………………………………………..12**

**VII - Conclusion …………………..……………………………………………………………..14**

**I - Introduction**

Le but de ce TP est d’implémenter un filtre RIF sur une carte FPGA. Pour cela nous devrons concevoir la FSM qui pilote l’ensemble du système et celle qui pilote l’ADC. Cela implique une phase de simulation puis de test sur carte.

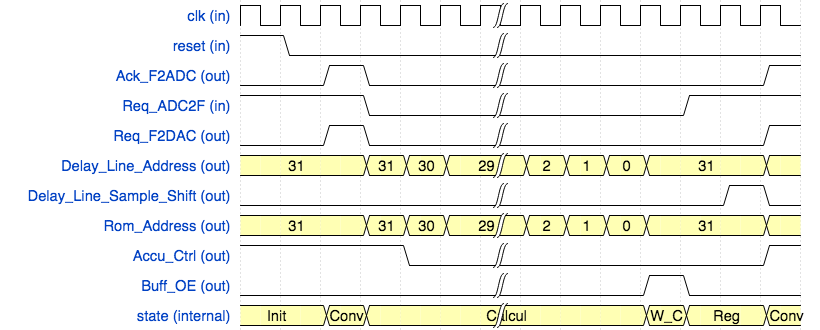
**II - FSM globale et chronogramme**

On a deux choix de conception pour déterminer la machine à états de la FSM :

* On lance la conversion, on attend la fin de conversion et on lance les calculs
* On lance la conversion, on fait les calculs en parallèle et on vérifie que la conversion s’est bien terminé

Notre choix de conception s’est porté sur la deuxième méthode pour réduire le temps de traitement (la période d'échantillonnage).

On commence par définir le chronogramme spécifique à la FSM :

figure 1 : chronogramme de la FSM

Après avoir relâché le reset la machine se trouve dans l’état *Conv*. Dans cet état, on fait une demande de conversion Analogique-Numérique.

Pendant ce temps de conversion, on passe dans l’état *Calcul* qui va décrémenter les adresses des éléments de la multiplication pour ensuite les accumulé et ce jusqu’à l’adresse 0 du tableau.

Une fois le calcul terminé, on rentre dans l’état *W\_C* où on stocke dans le buffer le résultat et on attend la fin de conversion.

Dans l’état *Reg* on décale les coefficients enregistrés, et ensuite on retourne dans l’état de conversion. L’actualisation du DAC est réalisée en même temps que le début de la relance de conversion.

Grâce à ce chronogramme on en déduit la machine à état de la FSM :

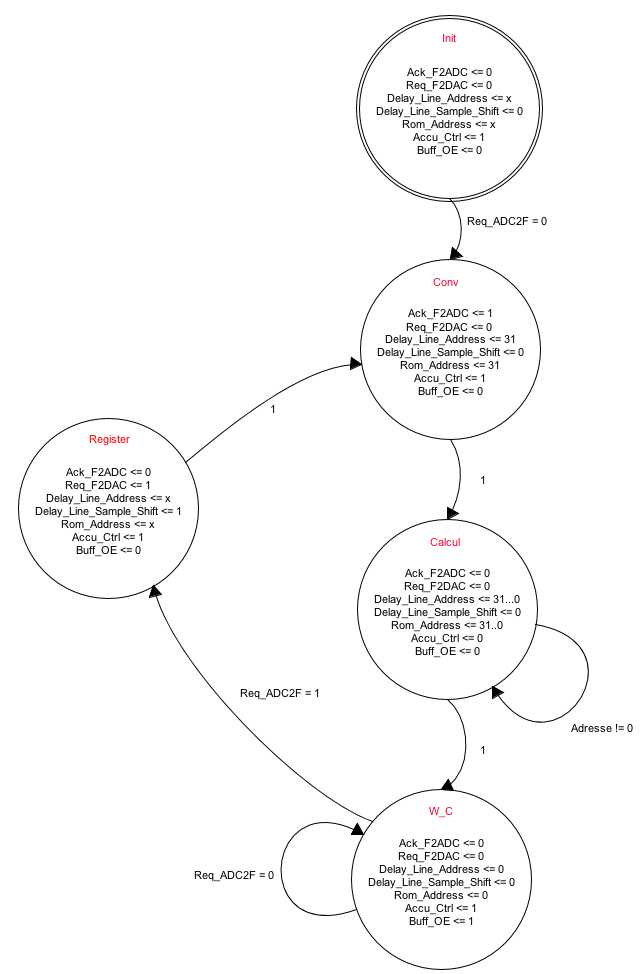


figure 2 : Machine à état de la FSM

Grâce au chronogramme de l’ADC\_FSM fourni dans la sujet on peut en déduire sa machine à état :

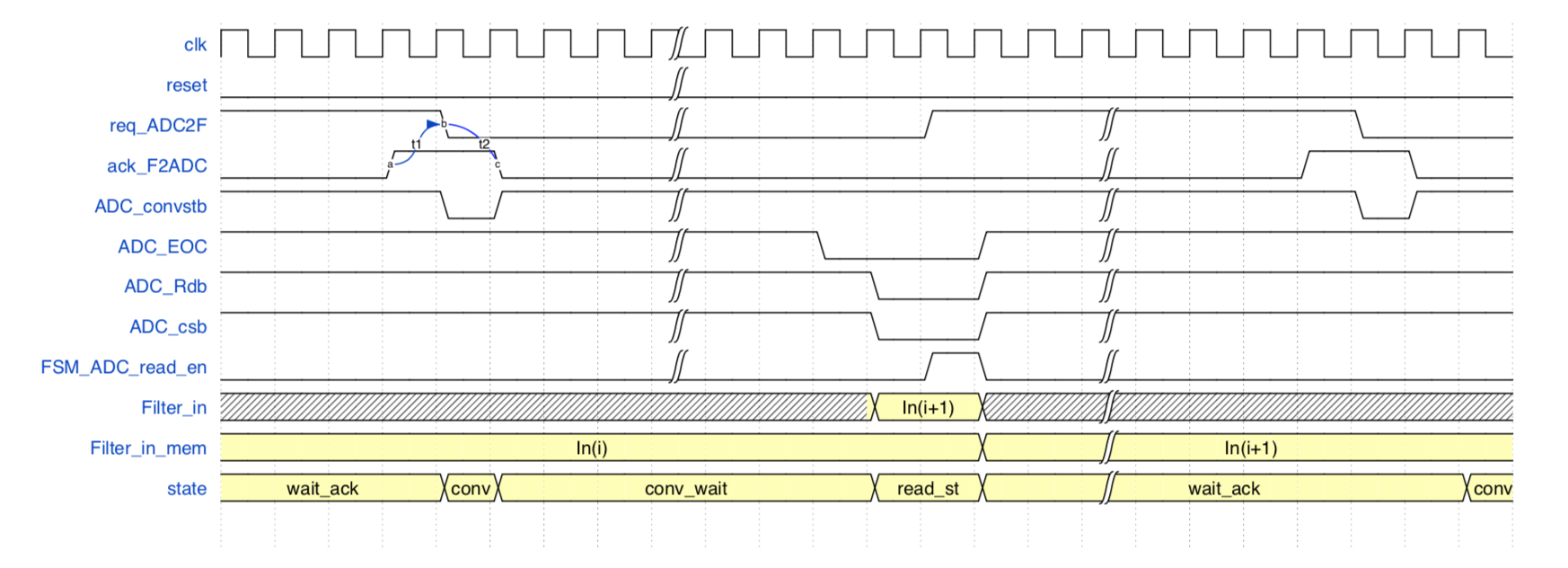
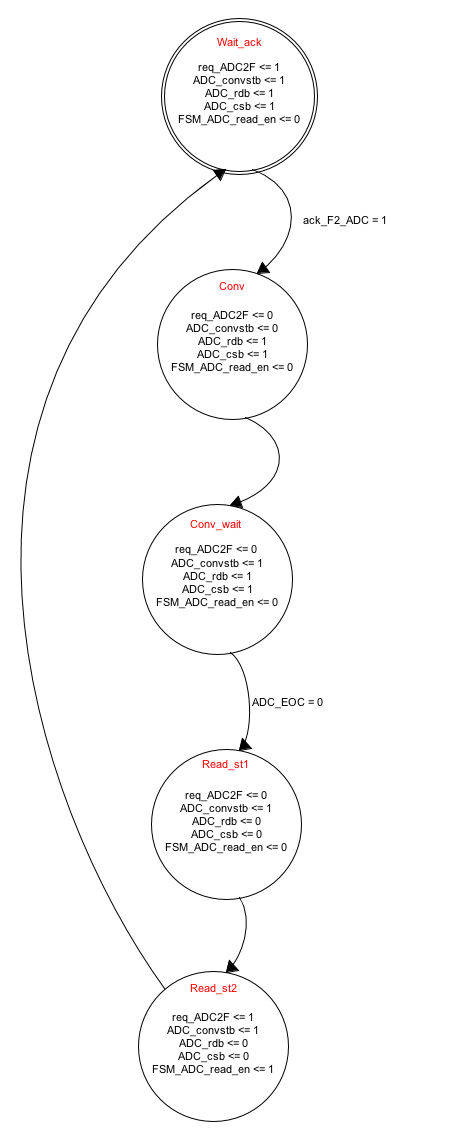
figure 3 : Protocole de communication entre la FSM contrôlant le calcul et la FSM contrôlant le CAN 

figure 4 : Machine à état de l’ADC\_FSM

A partir de ces machines à états, on peut donc commencer à écrire le code VHDL qui déterminera le comportements de chaque entité.



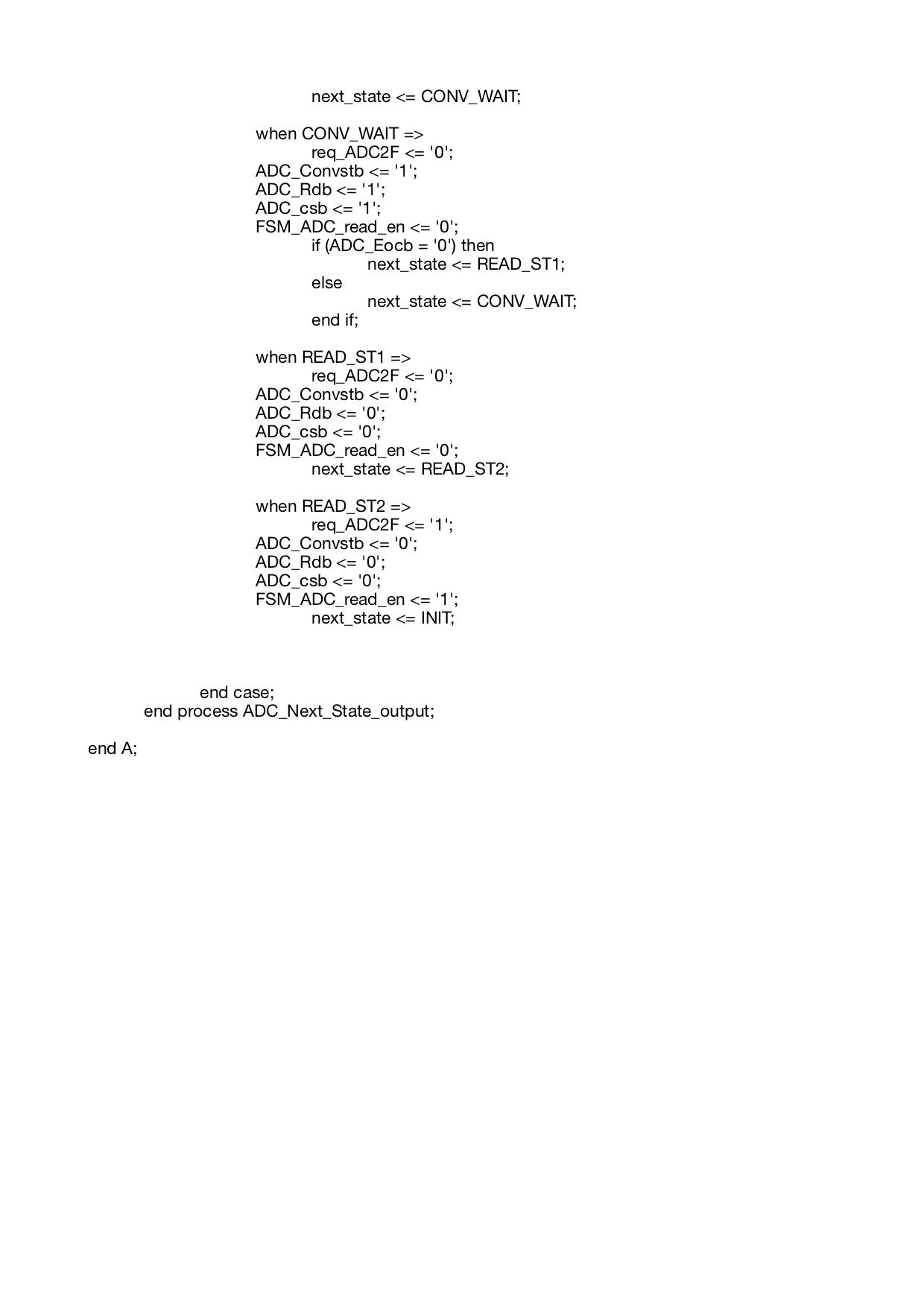
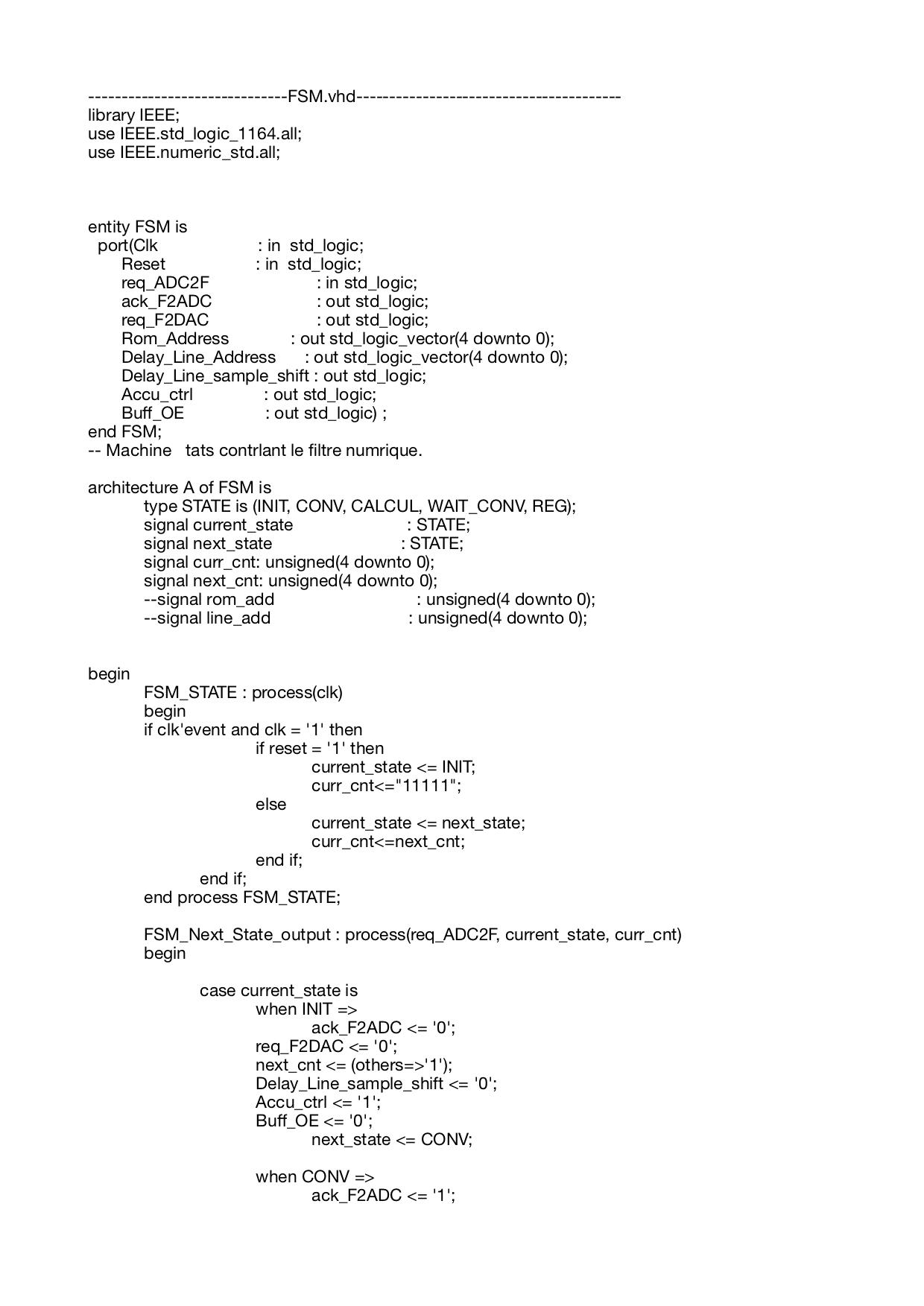


figure 5 : Code VHDL de l’ADC\_FSM



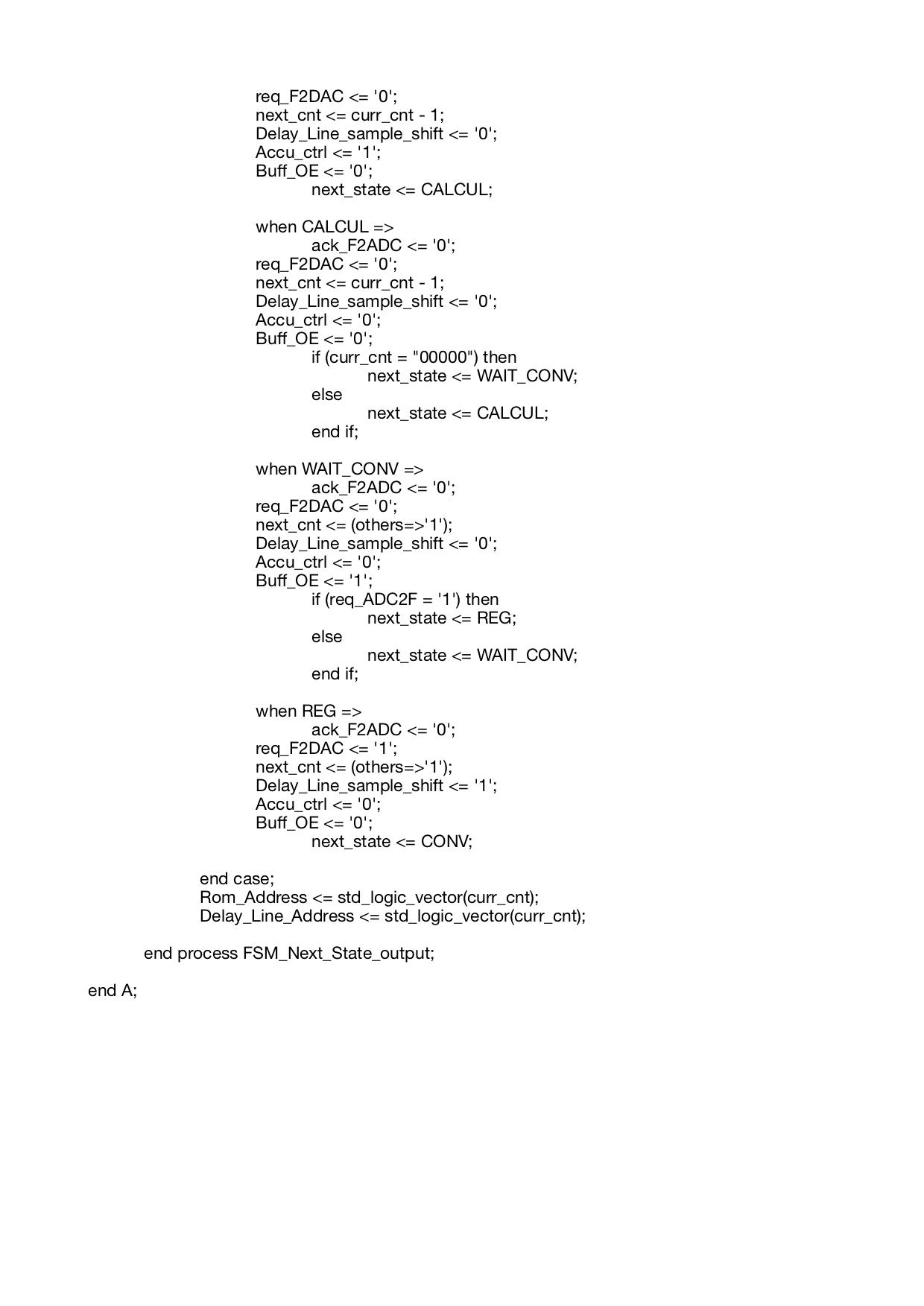


figure 6 : Code VHDL de la FSM

Pour chaque code écrit on vérifie bien que les règles de synthèse sont respectées pour la partie combinatoire, c’est-à-dire :

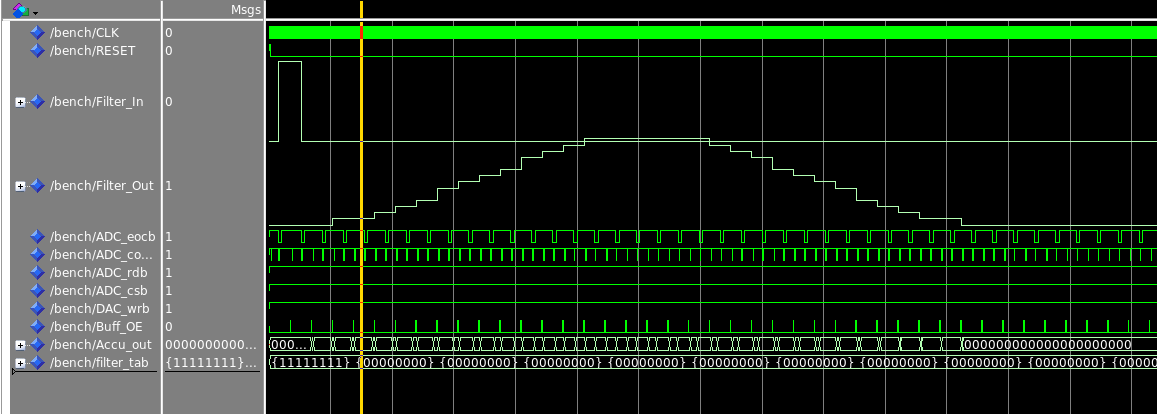
* Toutes les variables/signaux lu(e)s sont dans la liste de sensibilité du process
* Toutes les sorties sont affectées à chaque état

**III - Simulations**

Il faut valider le fonctionnement du code avant de passer aux étapes suivantes.

Pour cela nous allons observer le comportement de la réponse impulsionnelle, puis indicielle et enfin un sweep à l’aide des benchs fournis et complétés.

Réponse à une impulsion.

figure 7 : Réponse impulsionnelle de la simulation

On peut bien mesurer que la réponse impulsionnelle a bien une longueur de 32 périodes d'échantillonnage.

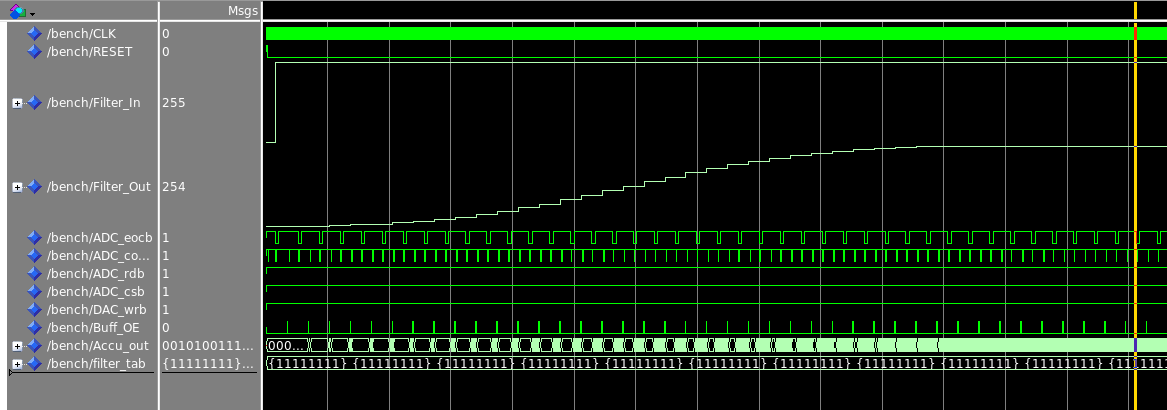
On peut aussi vérifier les valeurs de la sortie. Elle est symétrique, ce qui est normal car les coefficients le sont eux aussi.

Explications sur la valeur du premier échantillon en sortie:

On prend en sortie de l’accumulateur: le coefficient 0 de la delay line vaut 1111 1111 et le coefficient 0 de la rom vaut 0000 1101, ce qui donne sur 0 0000 1100 1111 0011 sur 21 bits

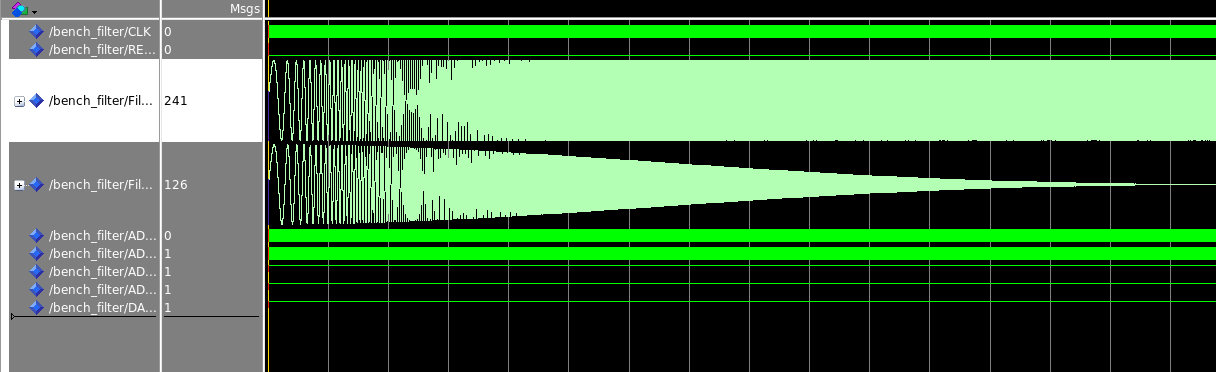
mais en sortie on garde entre les 12èmes et 19èmes bits. Ce qui donne 0000 0001. Et l’on retrouve cette valeur en sortie.

Réponse à un échelon :

figure 8 : Réponse à un échelon de la simulation

On observe bien la réponse d’un filtre passe bas a une impulsion.

Réponse à un sweep de fréquence:

figure 9 : Réponse de la simulation avec un entrée un sweep entre 127Hz et 1MHz

Le sweep permet d’obtenir une allure du diagramme de Bode. Le balayage des fréquences est ici linéaire. L'atténuation du signal de étant croissante avec la fréquence,nous avons donc bien un filtre passe bas.

**IV - Synthèse sur FPGA**

Avant d’implémenter le filtre sur FPGA on vérifie que le code est bien synthétisable et le circuit numérique ainsi créé est bien validé.

On réalise le flot de synthèse à l’aide de l’outil ***Precision de Mentor Graphics***

Une fois que la synthèse a été validé, on obtient un certain nombre de résultats qui vous nous permettre de paramétrer la carte FPGA et de connaître dans une moindre mesure le nombre d’éléments utilisé par le filtre.

Le premier paramètre que l’on cherche à connaître est le chemin critique et sa durée.

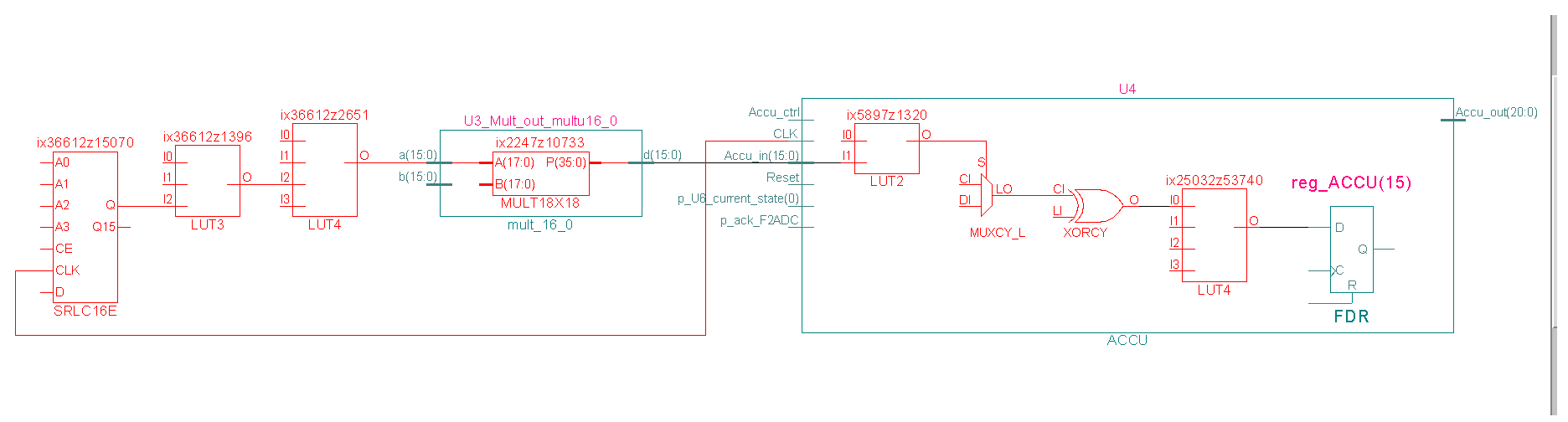
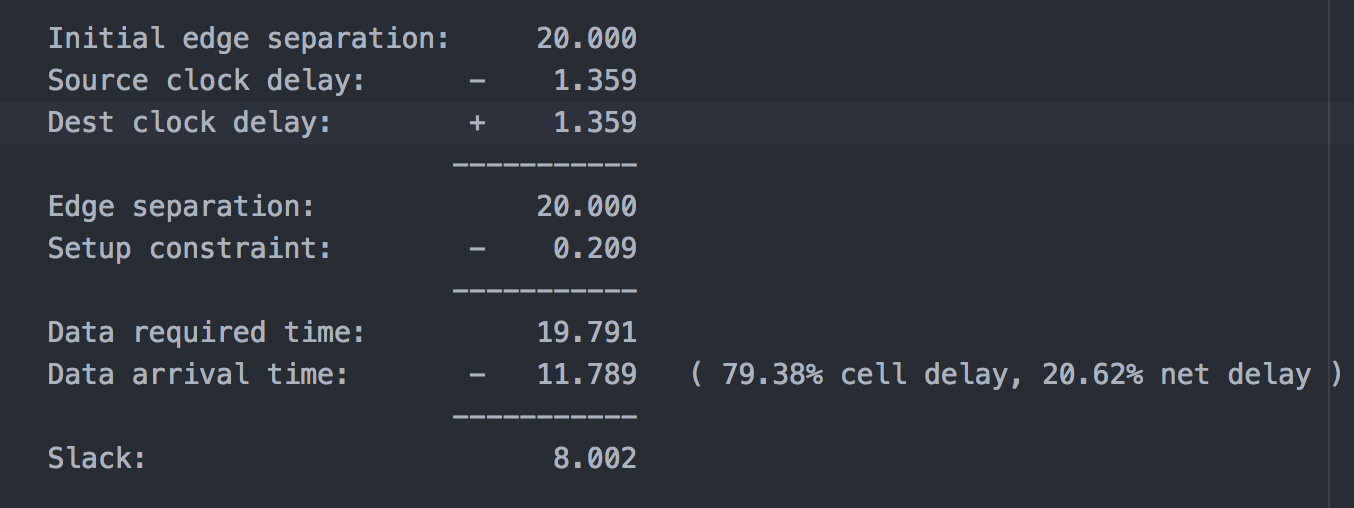
figure 10 : Chemin critique à travers les différents éléments de la carte

figure 11 : Résultat du rapport de synthèse sur le chemin critique

En connaissant le chemin critique, on peut déterminer la fréquence maximale de travail du FPGA.

Fmax = 1 / 8.002e-9 = 124 968 757,810547 Hz = 125 MHz

Si on voulait dépasser cette fréquence il faudrait mettre un buffer entre le multiplicateur et l’accumulateur.

Après avoir vérifié que le circuit était bien synthétisable et que le temps de travail du FPGA était une fréquence de travail inférieur à Fmax, on peut vérifier que les composants proposé la synthèse sont bien ce qu’on ce qu’on attend.

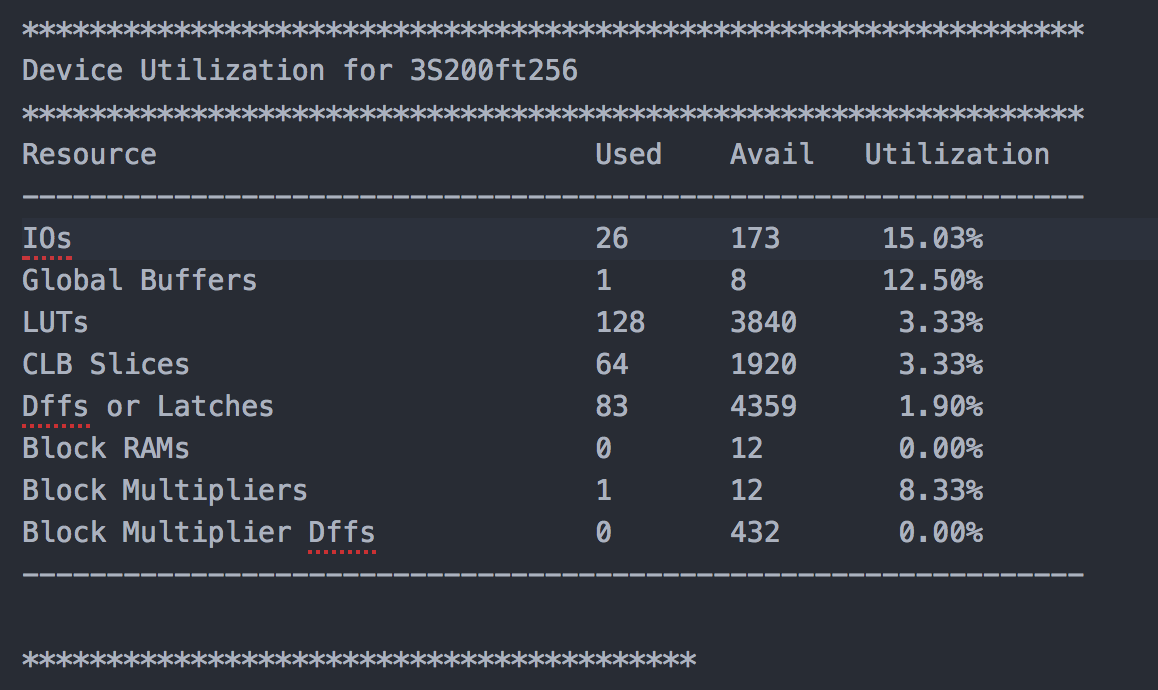


figure 12 : Résultat du rapport de synthèse sur les composants générés

On peut comparer le résultat avec le schéma de l’architecture pour vérifier que celui-ci est cohérent.

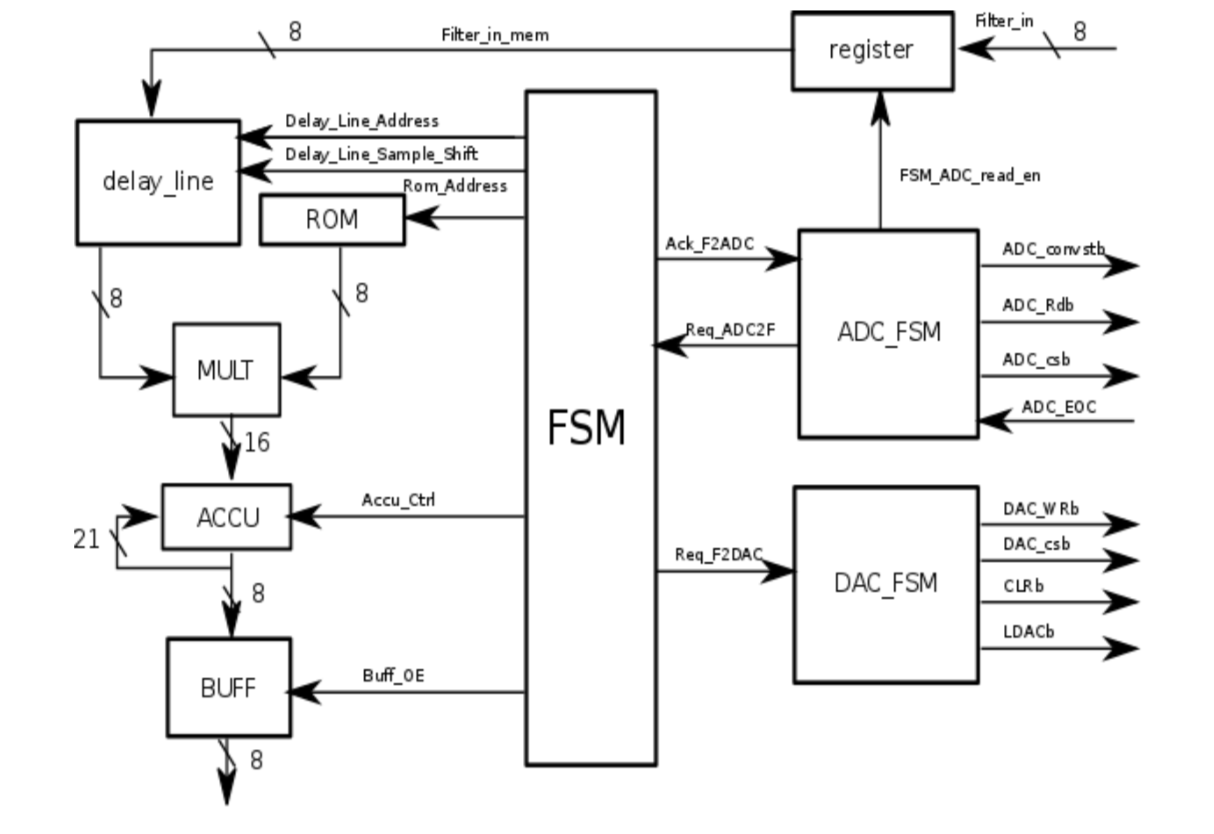


figure 13 : Architecture séquentielle détaillée

Sur le schéma de l’architecture, on compte 24 inputs/outputs + l’horloge et le reset, ce qui nous donne 26 inputs/outputs.

On a bien un buffer et un bloc multiplieur.

On s’attend à 8 + 8 + 8 + 8 + 16 + 8 + 8 + 32 + 32 = 128 LUTs, ce qui correspond à ce que nous dit la synthèse.

L’outil à choisi un code de gray pour coder les états.

Le paquetage VITAL sert à compiler le code pour le FPGA (traduire le code en portes logiques présentes sur le FPGA).

Après ces différentes vérifications, on peut dire que la synthèse du code VHDL est correct et correspond à l’architecture demandée.

On peut alors passer au placement-routage sur FPGA.

**V - Placement-Routage sur FPGA**

On vérifie les mêmes paramètres qu’après la synthèse, à savoir le temps du chemin et le nombre de composants.

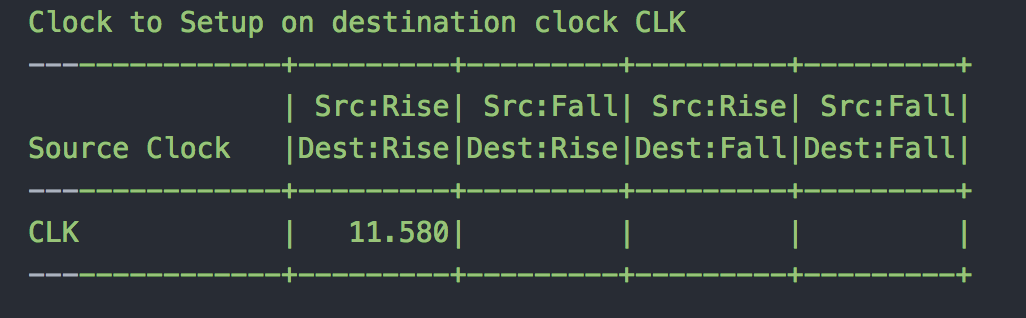


figure 14 : Résultat du rapport de placement-routage sur le chemin critique

On trouve un temps de chemin critique plus grand que pour la synthèse, ceci est normal car la synthèse fait ses calculs en prenant des hypothèses de modélisation du temps de propagation et du temps de bascule. Dans la réalité, le temps du chemin critique est toujours plus grand de celui donné par la synthèse, aux alentours de 25% en plus ( on retrouve bien cet ordre de grandeur : le temps après placement routage est 27% plus grand que celui donné par la synthèse).

Cela a pour conséquence de baisser la fréquence maximum de travail du FPGA. On avait Fmax = 125 MHz, maintenant la fréquence maximale de travail du FPGA est Fmax = 91 MHz.

Si on espérait faire travailler le FPGA à par exemple F = 100 MHz, cela est impossible avec le code VHDL proposé et le placement-routage effectué. Il faudrait alors changer soit le code VHDL, soit la façon dont le placement-routage est effectué.

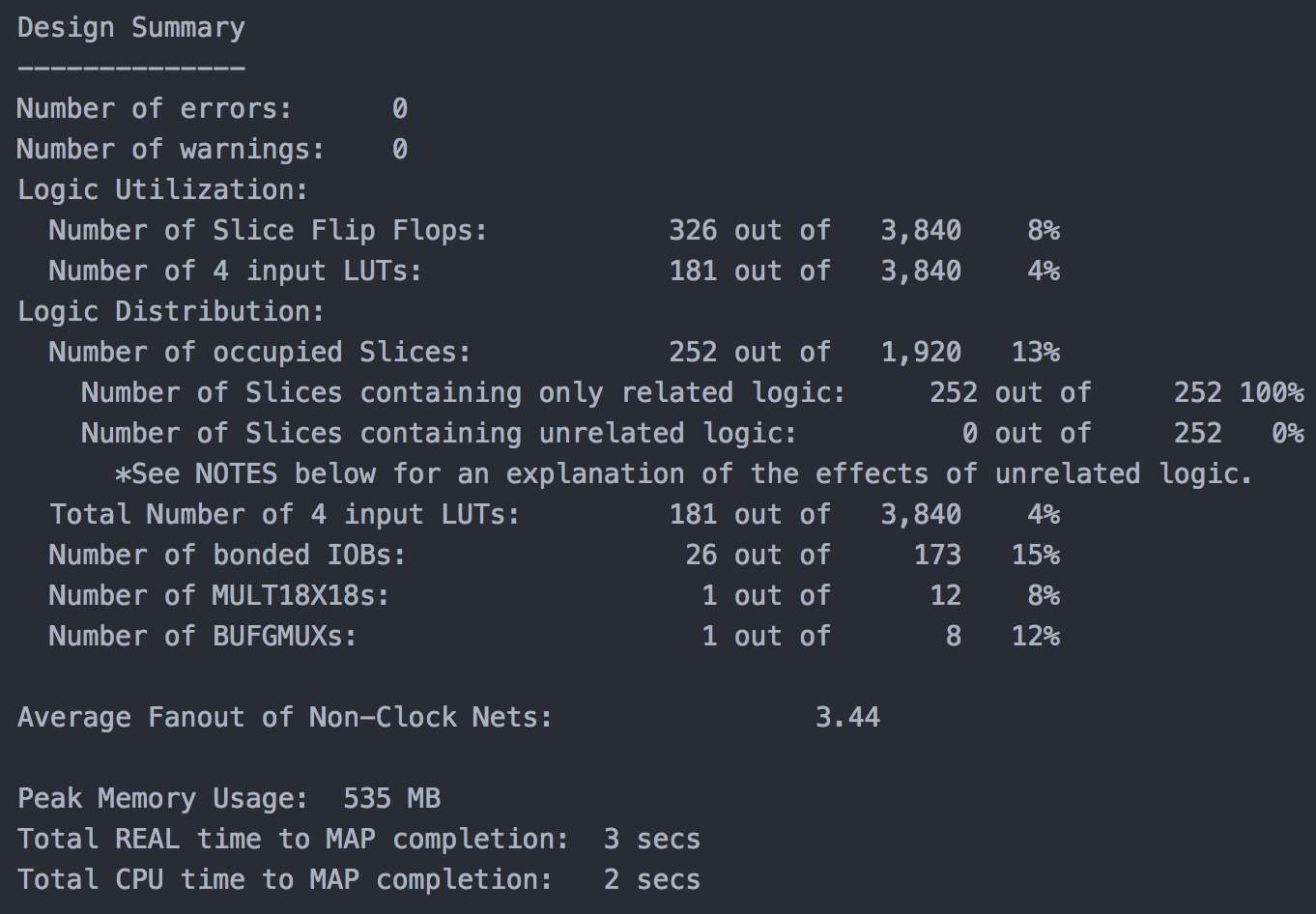


figure 15 : Résultat du rapport de placement-routage sur les composants

Si on fait la comparaison avec l'architecture comme avec pour la synthèse, on retrouve des différences, surtout dans le nombre de LUTs, de Slices et de registres car ces éléments sont dépendant du FPGA et donc peut augmenter le nombre de composants.

La synthèse optimise le circuit et donc, va choisir une architecture RTL optimale qui va remplir la fonction voulue.

Un FPGA est déjà un circuit réalisé que l’on peut juste “reprogrammer”, donc le programme de placement-routage va réaliser le circuit avec les composants déjà présent sur le FPGA. Ici on est amené à utiliser des Slices, or les Slices sont constitués de différents composants dont 2 LUTs, ce qui explique que le nombre de LUTs, de Slices et de registres soient différents.

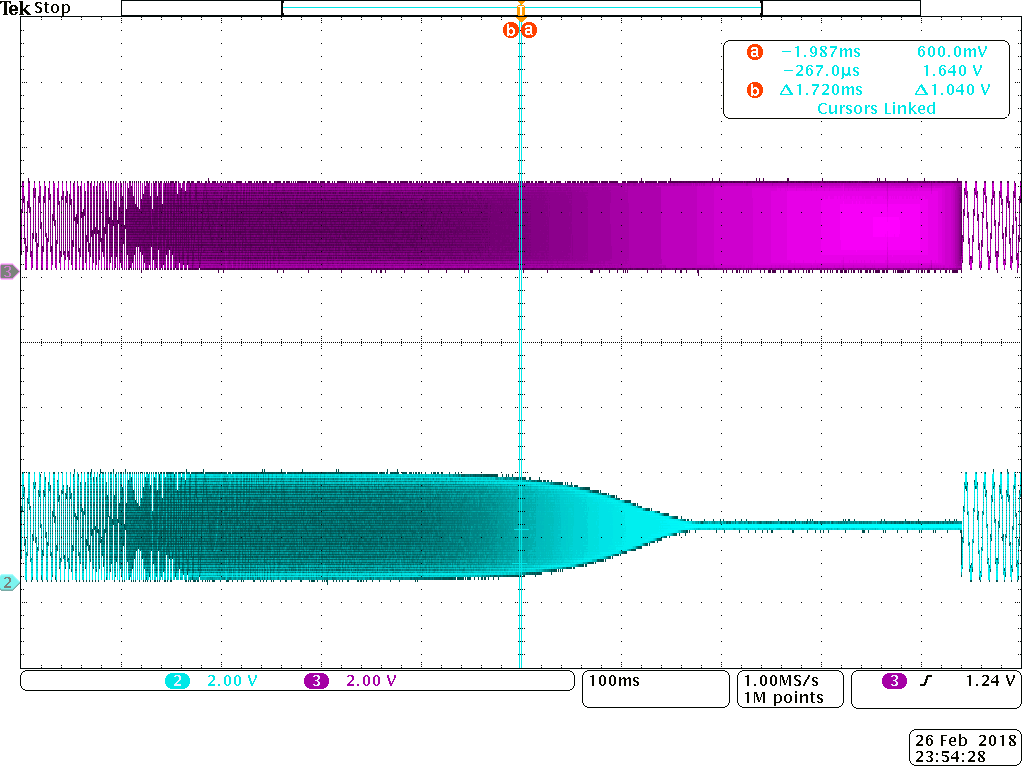
Toutefois on retrouve le bon nombres d’Inputs/Outputs, le bloc multiplieur, le buffer.

On pourrait chercher à vérifier le nombre de LUTs, de Slices et de registres en faisant une étude plus précise de l’architecture du FPGA.

**VI - Mesures sur carte:**

Après avoir mis le fichier .BIT sur la carte nous avons commencé par regarder si il y avait bien quelque chose en sortie. A ce moment nous avons eu un problème, il ne passait rien sur la carte. Pour résoudre ce problème nous avons effectué la compilation et le placement routage avec ISE.

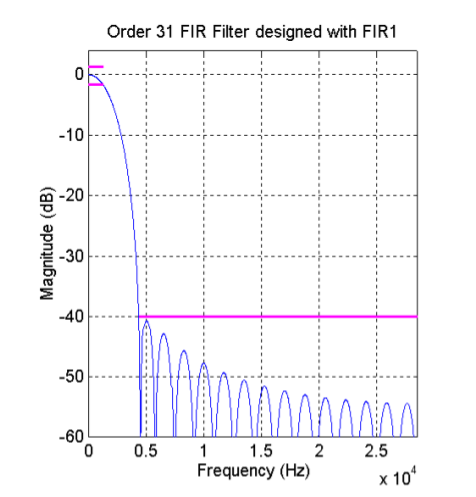
Ensuite nous avon mis un sweep de fréquence en échelle logarithmique de 100 Hz à 1MHz d’une durée de 1s. Nous observons bien la forme attendue.



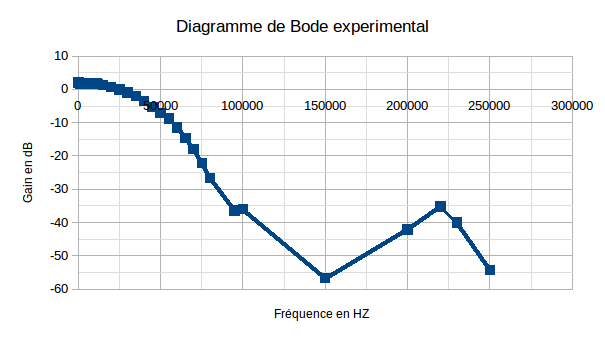
Ensuite en observant les signaux de pilotage du DAC que Te = 680 ns ce qui correspond bien à 34 cycles d’horloge pour un cycle de calcul.

Cela nous donne Fe= 1.47MHz.

Ensuite on veut tracer le diagramme de bode pour voir si on retrouve bien l’allure théorique du sujet:



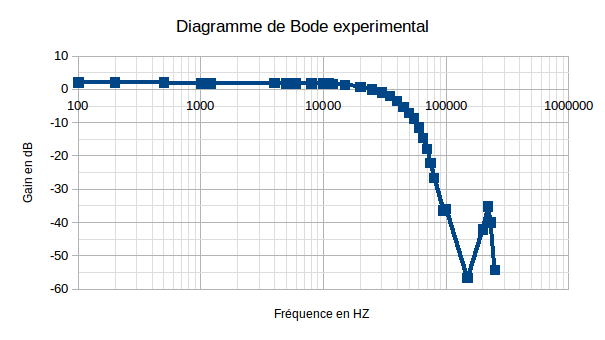
On considère que l’on a Fe/2=3e+4 Hz pour pouvoir comparer notre résultat à celui attendu.



D’après notre mesure précédente nous avons relevé Fe/2=735 kHz.

Pour passer du diagramme du sujet à celui relevé il faut donc multiplier par 735E+3/3E+4.

Le premier passage par 0 sur le diagramme de bode expérimental à 150000 Hz correspondrait à la fréquence 0.61E+4 Hz sur le tracé du sujet. Ce qui correspondrait en fait au deuxième passage par 0. Ce résultat n’est pas aberrant car il n’y a pas assez de points de mesure dans cette région car le diagramme à originalement été tracé sur une échelle logarithmique pour les fréquences.



**VII - Conclusion:**

Nous avons donc bien réussi à un implémenter un filtre RIF sur FPGA en suivant le flot de conception d’un circuit électronique numérique (spécification, code VHDL, simulation, synthèse, placement-routage, tests). Et nous pouvons en retenir les éléments suivants:

-Le code écrit doit pouvoir être synthétisable

-La simulation est indispensable pour pouvoir évaluer le comportement du code

-Il faut toujours avoir conscience de ce que les outils font et il faut vérifier manuellement le résultat.