

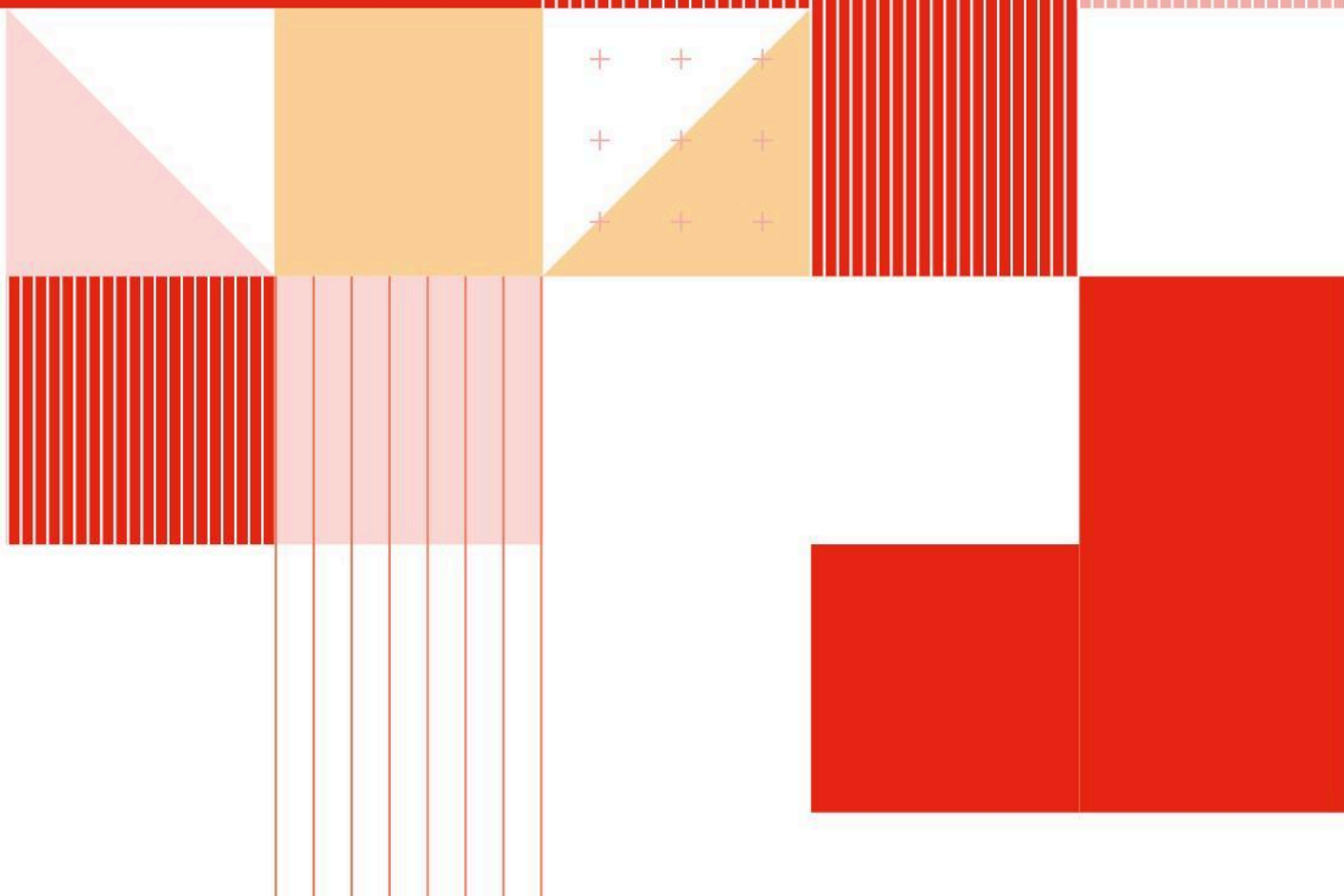
Architecture matérielle des Systèmes de Télécom

BE Controle d'accès au médium

Daniel Hennig

Sacha Bonicatto

4 IR-SC 2024



Architecture matérielle des Systèmes de Télécom

BE Contrôle d'accès au médium

Daniel Hennig

Sacha Bonicatto

4 IR-SC 2024

Table des matières

I- Introduction	1
II- Le receiver	1
II-a) format d'une trame	1
II-b) Adaptation de la période de réception	2
II-C) La simulation	2
II- Le transmitter	3
II-a) La simulation	3
III -La gestion des collisions	4
III-a) La fusion du transmitter et du receiver	5
III-b) La simulation	5
IV- Les caractéristiques de l'architecture	5
IV-a) Les bascules flip-flop	5
IV-b) La fréquence du contrôleur	5
V- Conclusion	6

I- Introduction

Dans le cadre de notre cursus en 4ème année à l'INSA en spécialisation informatique et réseaux et Système Communicants, nous avons implémenté une partie d'un contrôleur ethernet en vhdl. Ce rapport présente notre travail et les choix d'implémentation. Ci-joint le code dont sont tirés les résultats.

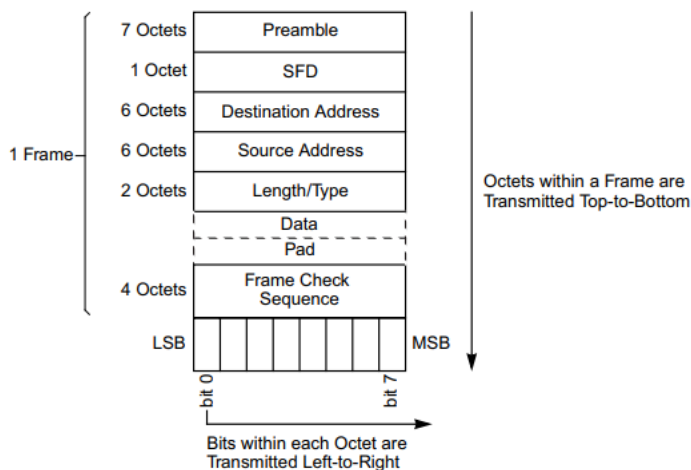
Lien vers le GitHub avec le code: <https://github.com/danielhng/ControleurEthernet.git>

II- Le receiver

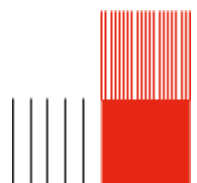
Nous avons premièrement implémenté la partie receiver. L'objectif de cette partie était de comparer une adresse connue avec une adresse reçue par le contrôleur et d'autoriser la réception des données si les adresses correspondent. Pour cela, il faut analyser le format d'une trame.

II-a) format d'une trame

Figure 3.1 MAC Frame Format



Les premiers bits reçus après le Preamble correspondent à un octet SDF de début de trame. C'est à la suite de cet octet que la comparaison des adresses se fait.



II-b) Adaptation de la période de réception

Les données sont reçues par paquets de 8 bits, or la fréquence f est de 10 MHz, ce qui nous donne une période de 100ns pour le pulse d'une période. Nous souhaitons analyser 8 bits à la suite, donc 8 périodes, il nous faudra pour cela une période de 800ns.

Pour cela nous avons mis un compteur dans un rang de 0 à 7 :

```
signal s_count : integer range 0 to 7 := 0;
```

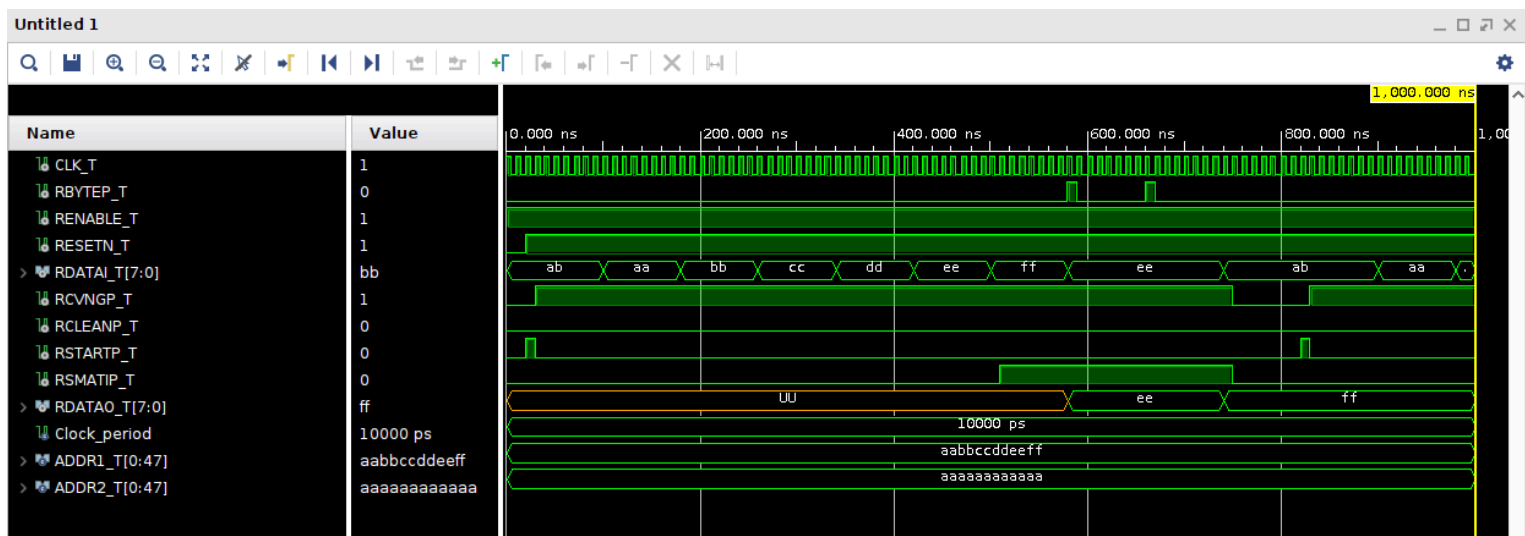
qui nous permet d'exécuter le code seulement après 8 clock grâce à un "if" qui vérifie si la variable est à 0 et une incrémentation sinon.

II-C) La simulation

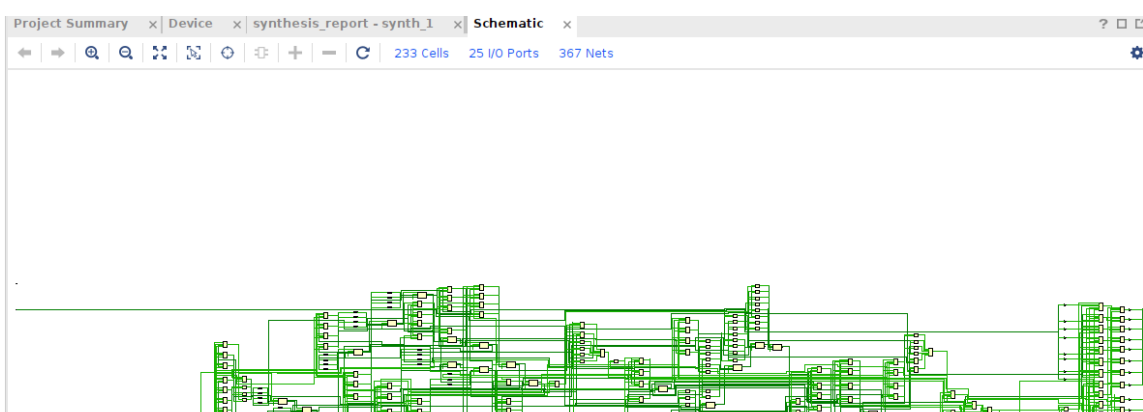
Sur L'image ci-dessous nous pouvons voir que la fonctionnalité de receiver fonctionne :

Après un pulse de RSTARTP nous avons RCVNGP qui passe à 1. Nous testons la concordance avec l'adresse **aa:bb:cc:dd:ee:ff**, à la fin de la comparaison, les adresses étant les mêmes, nous avons une transmission de la donnée envoyée sur RDATAO jusqu'à la prochaine réception d'un signal de fin de frame.

Nous avons fait le choix arbitraire de transmettre le signal "ff" lorsque nous sommes en train d'attendre une autre frame.



Le schéma et la taille de l'architecture du receiver est le suivant :



II- Le transmitter

La logique du transmitter est similaire à celle du récepteur mais dans le sens inverse. Il s'agit de transmettre le paquet tant qu'il n'est pas fini.

Afin de bien envoyer les 3 octets lors de l'abort d'une transmission, nous avons utilisé un compteur qui permet de rester dans la boucle jusqu'à la fin de la transmission.

Lorsque le compteur arrive à 3, on asserte abortP et on réinitialise le transmitter afin d'être prêt pour une nouvelle transmission.

Comme l'adresse de l'émetteur est l'adresse constante ici, nous l'avons implémenté comme une constante du programme qu'il envoie automatiquement après la réception de l'adresse destinataire.

Pour gérer l'attente de la transmission de l'adresse destinataire de l'adresse source nous avons aussi utilisé un compteur qui permet de suivre l'avancée dans le code. Lorsque celui-ci est au-dessus de 6, la transmission de l'adresse destinataire est en cours et lorsque celui-ci est entre 0 et 6 la transmission de l'adresse source est en cours. Lorsque le compteur arrive à 0, il est réinitialisé et le signal endadresse est mis à 1 pour indiquer la fin de la gestion des adresses et la possibilité de passer à la transmission des données. Le signal EndAdresse n'est pas réinitialisé avant le début d'une nouvelle transmission pour assurer l'envoi de toutes les données.

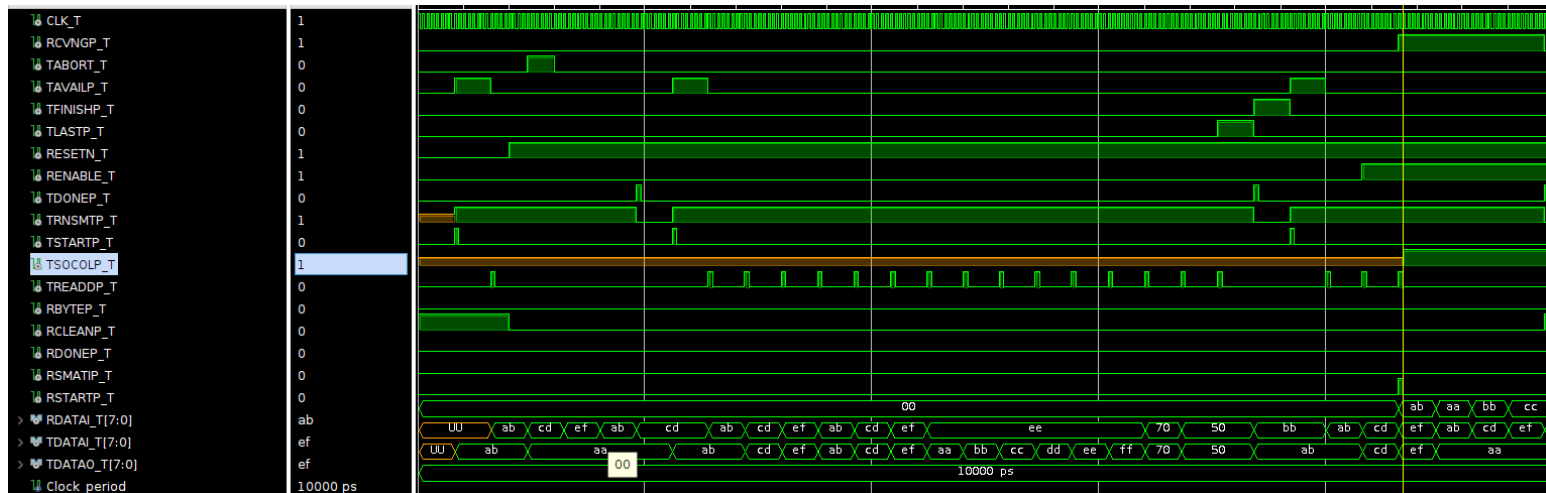
Ensuite pour gérer la réception de TLASTP et de TFINISHP, nous avons utilisé une variable qui enregistre le fait que le dernier byte arrive pour permettre l'attente de la réception de TFINISHP. Enfin la réception de TFINISHP déclenche l'envoi de l'EFD et la terminaison de la transmission.

II-a) La simulation

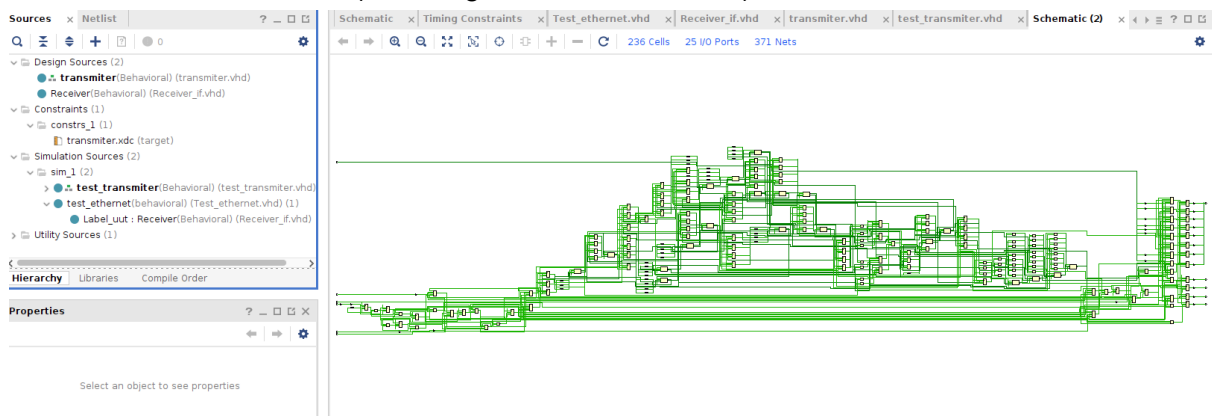
La simulation suivante représente tout d'abord une itération de transmission avec une assertion de TABORTP qui interrompt bien la transmission. Ensuite on voit une transmission réussie après la première assertion de TDONEP_T. Enfin nous avons testé une transmission en même temps qu'une réception et nous voyons bien l'assertion de TSOCOLP qui signifie que la collision a été détectée et nous avons bien TDATAO qui envoie les suites de "10" pour indiquer l'interruption de la transmission.

Ces simulations nous ont été très utiles pour tester l'efficacité de notre code au fur et à mesure de l'implémentation.

simulation du transmitter :



architecture du transmitter (sans la gestion des collisions):



III -La gestion des collisions

Pour la gestion des collision, il a fallu ajouter une variable qui se met à 1 lorsque nous avons l'émission et la réception en même temps. Si cette variable passe à 1 il faut alors arrêter la transmission, de plus, il faut gérer la situation de la même manière que lors d'un TABORTP, c'est-à-dire un envoie sur TDATAL d'un nombre déterminé de "10".

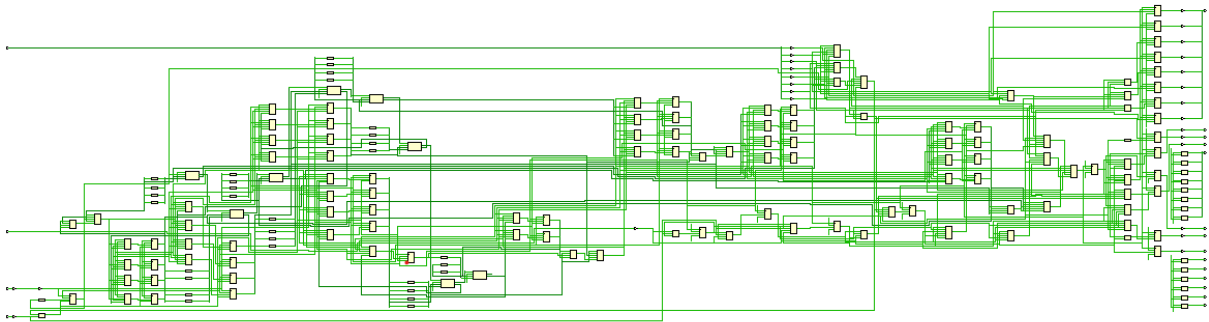
III-a) La fusion du transmitter et du receiver

Pour cette partie de l'implémentation, nous avons fusionné notre récepteur et notre transmitter dans un seul fichier, TransReceivCollision.vhd. Nous avons gardé les fichiers de Receiver et de Transmitter à côté pour pouvoir effectuer des tests sur ceux-ci sans perturbations externes.

Notre méthode a été d'utiliser plusieurs processus indépendants afin de pouvoir gérer les différentes possibilités. Nous avons donc un processus de transmission, un de réception et nous avons implémenté celui de collision.

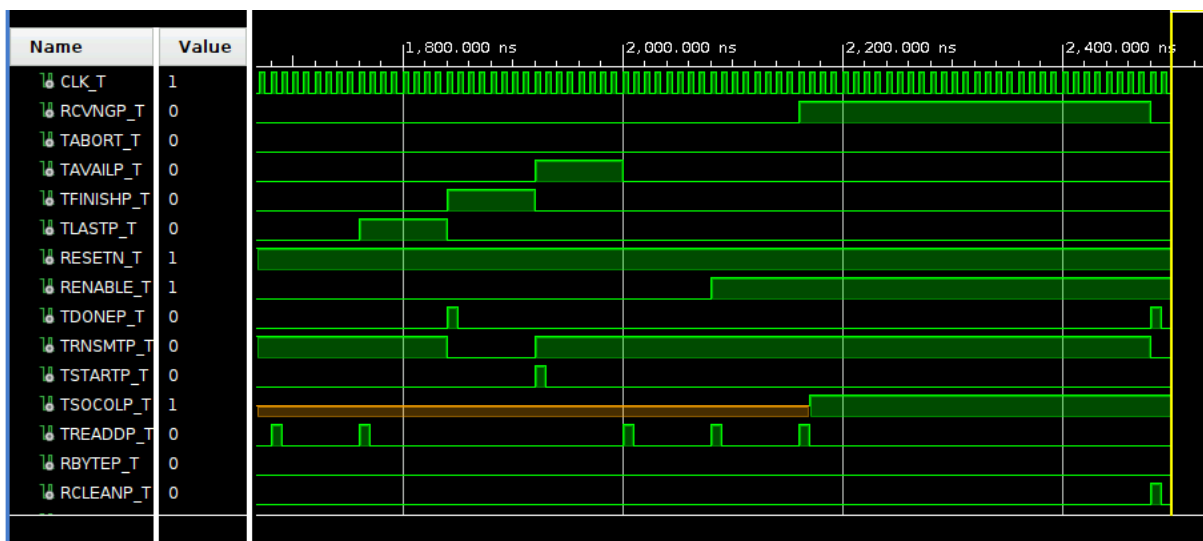
Nous avons ajouté un processus de collision dans notre fichier fusionné qui met la variable d'output TSOCOLP à 1 lors d'une détection d'envoi et de réception simultanée. De plus, nous avons ajouté une condition dans notre process de Transmitter pour que lors de la détection d'un signal TSOCOLP à 1 celui ci effectue les mêmes démarche que lors d'un TABORTP.

Architecture du modèle fusionné:



III-b) La simulation

Sur la simulation suivante, nous voyons bien le scénario où il y a une tentative de réception pendant une transmission. TSOCOLP passe bien à 1.

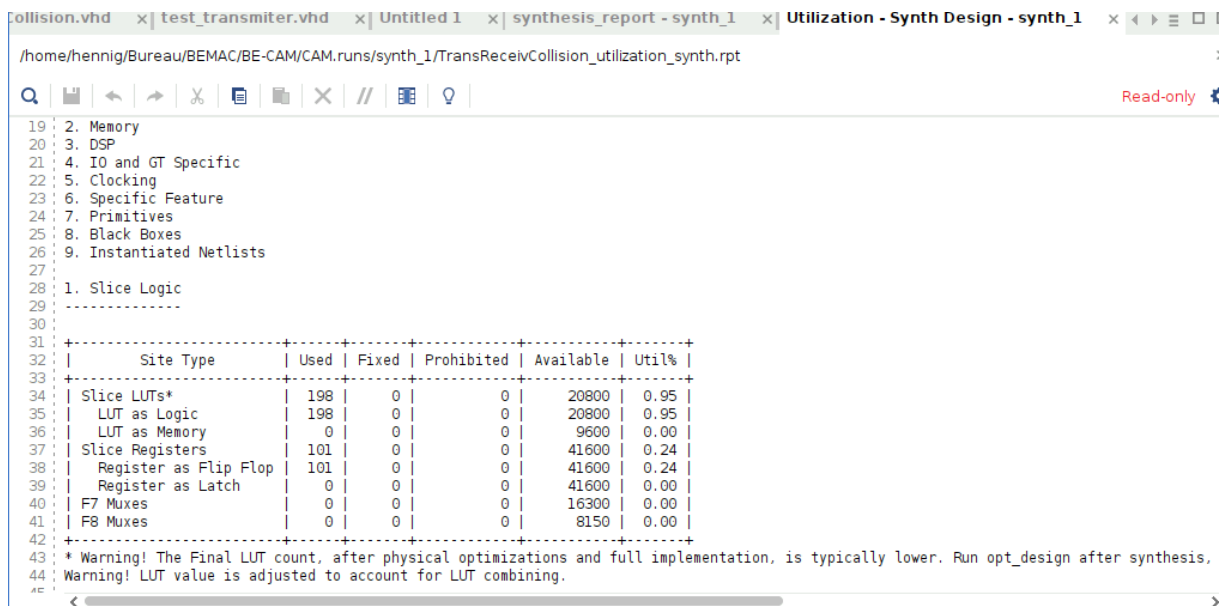


IV- Les caractéristiques de l'architecture

Vivado nous permet de connaître différentes caractéristiques de notre contrôleur grâce aux rapports de synthèses.

IV-a) Les bascules flip-flop

Le synth_synthesis_report nous permet notamment de connaître le nombre de bascules flip flop utilisées :



Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	198	0	0	20800	0.95
LUT as Logic	198	0	0	20800	0.95
LUT as Memory	0	0	0	9600	0.00
Slice Registers	101	0	0	41600	0.24
Register as Flip Flop	101	0	0	41600	0.24
Register as Latch	0	0	0	41600	0.00
F7 Muxes	0	0	0	16300	0.00
F8 Muxes	0	0	0	8150	0.00

Ici nous voyons 101 bascules flip flop soit l'intégralité des bascules.

IV-b) La fréquence du contrôleur

L'analyse synthétique du code nous permet d'accéder au report_timing_summary et de trouver la fréquence par le calcul :

Nous avons $T = 2,155 \text{ ns}$
Or $F = 1/T$
d'où $F = 1/(2,155 * 10^{-9}) = 464 \text{ MHz}$

Extrait du rapport de timing :

Pulse Width Checks									

Clock Name:	CLK								
Waveform(ns):	{ 0.000 5.000 }								
Period(ns):	10.000								
Sources:	{ CLK }								
Check Type	Corner	Lib Pin	Reference Pin	Required(ns)	Actual(ns)	Slack(ns)	Location	Pin	
Min Period	n/a	BUFG/I	n/a	2.155	10.000	7.845		CLK_IBUF_BUGF_inst/I	
Low Pulse Width	Slow	FDRE/C	n/a	0.500	5.000	4.500		EndAddress_reg/C	
High Pulse Width	Slow	FDRE/C	n/a	0.500	5.000	4.500		EndAddress_reg/C	

V- Conclusion

Ce BE nous a permis d'apprendre un nouveau langage de programmation et de comprendre en profondeur le fonctionnement des échanges de trames ethernet.

Il aurait été intéressant dans la continuité que nous codions des fonctionnalités supplémentaires comme un processus de génération aléatoire d'un temps d'attente lors d'une collision avant de renvoyer la trame perdue par exemple.