

**Projet VHDL**

**Polar Encoder**

M1. Master en Systèmes Embarqués et Systèmes Intégrés

Étudiant 2: Gaetan Michaud

Étudiant 2: JAVIER LÓPEZ José Antonio

No. Etudiant : 21804920

Date de livraison : 5 novembre 2019

Professeur : Emmanuel BOUTILLON

Sommaire

[1. Introduction 1](#_Toc23863325)

[1.1. Préambule 1](#_Toc23863326)

[1.2. Objectif 2](#_Toc23863327)

[1.3. Description du projet 2](#_Toc23863328)

[2. Conception / Développement 3](#_Toc23863329)

[2.1. Système globale 3](#_Toc23863330)

[2.2. Interface 3](#_Toc23863331)

[2.3. Les divers composants de l’encodeur 4](#_Toc23863332)

[a. Composant : Réception 4](#_Toc23863333)

[b. Composant : Contrôle 6](#_Toc23863334)

[c. Composant : FrozenBits 7](#_Toc23863335)

[d. Composant : Hadamard 8](#_Toc23863336)

[e. Composant : Mémoire ROM 12](#_Toc23863337)

[f. Composant : Mémoire RAM pour la réception 12](#_Toc23863338)

[g. Composant : Mémoire RAM de l’encodeur 13](#_Toc23863339)

[h. Composant : Transmission 13](#_Toc23863340)

[2.4. Les environnements de test 15](#_Toc23863341)

[3. Résultats 16](#_Toc23863342)

[4. Conclusion 17](#_Toc23863343)

[5. Feedback personnels 18](#_Toc23863344)

[5.1. José JAVIER : 18](#_Toc23863345)

[5.2. Gaëtan MICHAUD : 18](#_Toc23863346)

# Introduction

Nous sommes actuellement en Master 1 Systèmes Embarqués / Systèmes Intégrés, cette mention vise à former des spécialistes capables d’appréhender, c'est-à-dire de modéliser, de concevoir, de gérer ou d’optimiser des applications mettant en œuvre des systèmes embarqués.

Ce parcours vise à former des ingénieurs spécialisés dans la conception de systèmes électroniques à dominante numérique. Dans ce contexte, le diplômé sera un expert dans les domaines permettant de concevoir des architectures de système mixte, matériel-logiciel, complexe et communicant.

Durant cette formation, nous devons travailler sur un projet en VHDL (Very high speed integrated circuit Hardware Description Language).

## Préambule

Le but premier de ce projet, est de nous permettre d’acquérir des compétences.

* Compétences scientifiques :
  + Avoir une attitude de rigueur et une logique scientifique ;
  + Constituer des dossiers scientifiques et techniques ;
  + Appréhender de nouvelles connaissances techniques.
* Compétences techniques :
  + Concevoir et optimiser des systèmes numériques performants et sûrs.
* Compétences organisationnelles :
  + Travailler en autonomie, en responsabilité ;
  + Travailler en temps limité (respect des délais).
* Compétences relationnelles :
  + Communiquer avec clarté et précision dans un langage adapté ;
  + Travailler en équipe autour d’un projet – capacité à s’intégrer.
* Capacités personnelles :
  + Esprit d’initiative ;
  + Volonté à accéder à un résultat visé.

## **Objectif**

Ce projet a pour objectif de nous faire travailler sur une carte FPGA, un système qui requiert une communication PC – FPGA, un calcul sur la carte et le retour du résultat de la carte FPGA au PC.

Le projet est suffisamment complexe pour que le travail en groupe soit nécessaire pour arriver à finaliser un circuit.

Le projet est suffisamment ouvert pour nous confronter à une situation d’autonomie.

## Description du projet

Dans le cadre de la 5G, les codes polaires ont été normalisés. Le but de ce projet est de réaliser un encodeur de code polaire sur une carte FPGA afin de garantir un débit en sortie de 250 Mbit/s.

Le principe de l’encodeur est très simple : on encode un message *M* de *k* bits sur un mot *Y* de *n* = bits, avec *n* > *k*.

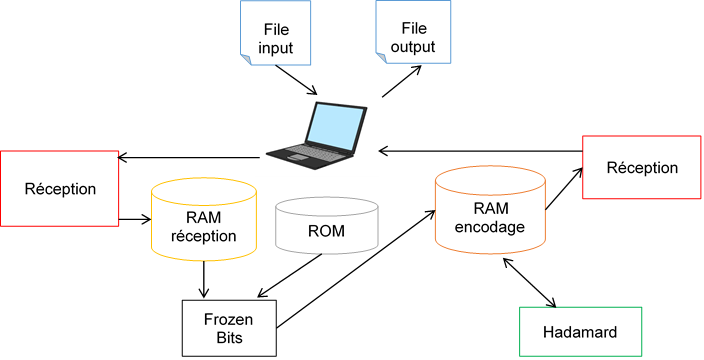
Ces codes sont les premiers codes pouvant atteindre, de façon prouvée, la capacité du canal pour les canaux sans mémoires symétriques. En plus, ces codes possèdent une faible complexité d'encodage et de décodage, soit de ***O(n log n)***. Ces particularités rendent donc les codes polaires très attrayants pour la recherche et les applications industrielles.

# Conception / Développement

## Système globale

Pour utiliser le système, il doit fonctionner de manière dynamique, c'est-à-dire qu'il n'est pas possible d'utiliser des enregistrements prédéterminés, puisque le message d'entrée peut être de taille différente et que la même chose se produit avec le message de sortie. Dans ce document, nous montrons comment ce problème a été résolu, grâce à l'utilisation de mémoires RAM.

Le système se compose de 5 composants principaux, deux mémoires RAM et une mémoire ROM.



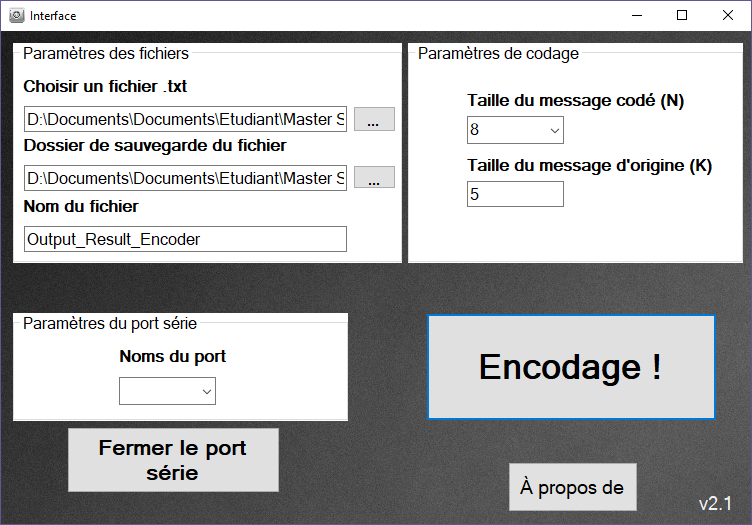
*Figure 2 : Schéma général du système*

## Interface

Comme dit précédemment, le rôle de l’interface est de prendre le message de l’utilisateur, l’envoyer, via une connexion série, à la carte FPGA. Et enfin, de récupérer le message encodé.

L’utilisateur écrit le message à encoder dans un fichier .txt, puis il paramètre l’interface et enfin, l’utilisateur donne l’ordre à l’interface d’envoyer le message à la carte. La réponse de la carte est écrite dans un second fichier.txt.

L’interface se présente comme ceci :



*Figure 3 : Interface du projet*

L’utilisateur choisi le port série correspondant à la carte FPGA, puis paramètre la taille de *N* et de *K*.

Entrées / Sorties

Entrées :

* Fichier .txt ;
* Les divers paramètres.

Sorties :

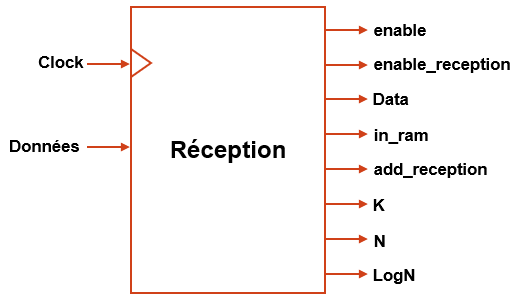
* La taille de N et de K ;
* Log n en base 2 ;
* Le message.

L’interface a été développé en C#. En effet, ce langage offre une programmation intuitive pour les applications en fenêtre.

## Les divers composants de l’encodeur

### Composant : Réception

Ce composant nous permettra de stocker toutes les informations envoyées par l'utilisateur.

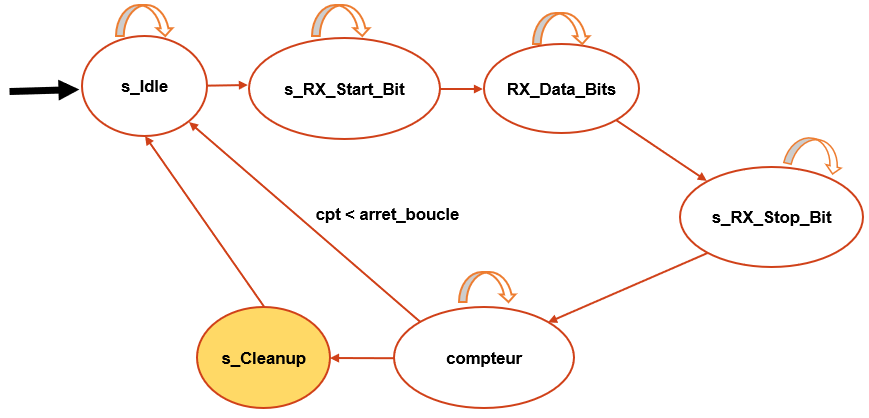


*Figure 5 : Composant VHDL : Réception*

La partie réception comporte 6 états principaux :

* L'état s\_Idle, nous permet de réinitialiser les variables et d'attendre la variable qui nous permettra de passer à l'état suivant ;
* S\_RX\_Start\_bit : Cette état permet d'initialiser le processus de réception des données, dès qu’il reçoit le bit '0'. Le processus est entièrement synchronisé avec la vitesse de 9600 bauds, puisqu'il existe un processus externe qui compte le système général et les conditions ne sont exécutées que lorsque la vitesse de transmission est atteinte.
* S\_RX\_Dat\_Bit : Une fois que le composant a été initialisé, il commence à récupérer les données reçues et à les sauvegarder dans une variable. Afin d'effectuer ce processus, il est nécessaire d'avoir un compteur interne qui déplace l'index où nous voulons sauvegarder la valeur reçue.
* S\_RX\_Stop\_Bit : Une fois les données reçues enregistrées, le processus de réception peut être considéré comme terminé.
* Compteur : Cet état nous permettra de récupérer les données nécessaires à l'exécution du programme, deux compteurs sont nécessaires, le compteur général nous permet d'effectuer le processus suivant :
* Premier cycle d'itération : Récupérer la valeur de K
* Deuxième cycle d'itération : Récupérer la valeur de N
* Troisième cycle d'itération : Récupérer la valeur de Log M
* Le reste des cycles : Récupérer le message

Le deuxième compteur, nous permet de savoir combien de valeurs seront stockées dans la réception RAM, ceci se fait en faisant l'équivalent en VHDL de la fonction "ceil" (division et utilisation du module de commande). Lorsque le nombre de valeurs a été correctement enregistré, le cycle se termine.

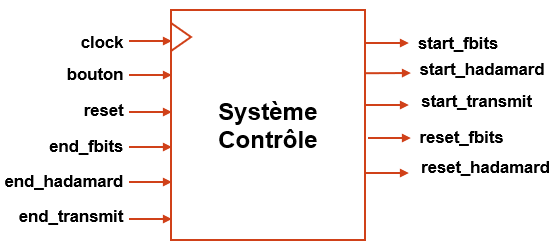
* Enfin, l'état s\_cleanup nous permet de réinitialiser toutes nos variables utilisées et de revenir ensuite à l'état initial.
* 
* *Figure 5 : Machine d’états du composant de Réception*

### Composant : Contrôle

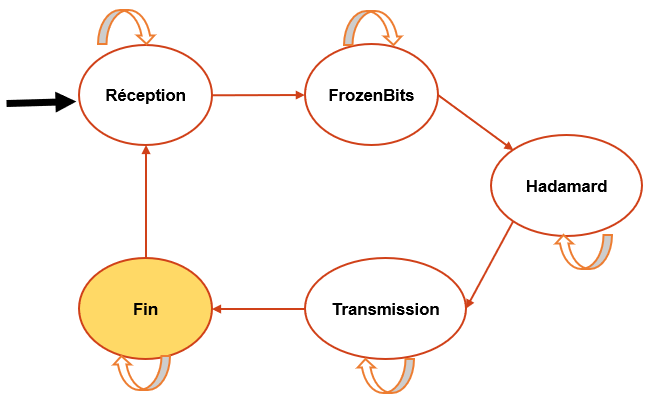
C'est le composant principal de notre système, car il nous permet de définir quand chacun des composants doit être exécuté. Le composant utilise une machine d'état dans l'ordre suivant :

* Réception, FrozenBits, Hadamard, Transmission et Fin

Chacune de ces étapes est exécutée de manière séquentielle, la dernière étape « *FIN* », indique que l'ensemble du processus est terminé et qu'il est possible de revenir à la réception des données.



*Figure 6 : Composant VHDL : Contrôle*



*Figure 7 : Machine d’états du composant de contrôle*

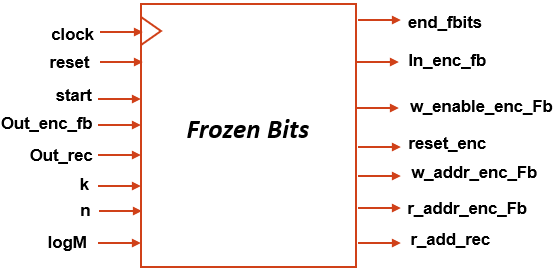
### Composant : FrozenBits

C'est le composant chargé d'insérer les zéros dans le vecteur qui sera utilisé pour déterminer le message crypté. Pour que le composant soit géré dynamiquement, nous avons besoin des signaux suivants :

* start : Ceci permet au composant de fonctionner ;
* reset : Met toutes les variables dans leurs états d'origine ;
* out\_enc\_fb : Valeur de l'encodage de la RAM ;
* out\_rec : Valeur de la RAM de réception ;
* k : Taille du message d'entrée ;
* n : Taille du message de sortie ;
* logM : Logarithme de la taille du message de sortie.

Les signaux de sortie sont :

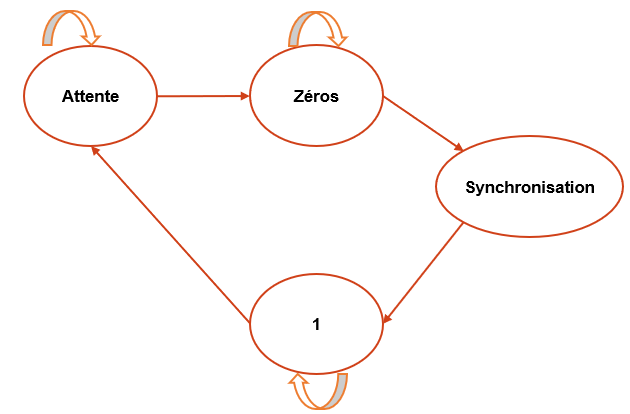
* end\_fb : Flag indiquant que le composant a terminé son exécution ;
* in\_enc\_fb : Le résultat est transmis petit à petit dans la mémoire vive ;
* w\_enable\_enc\_fb : Permet de sauvegarder les valeurs dans la RAM ;
* reset\_enc : Permet de remettre la RAM à ses valeurs initiales ;
* w\_add\_enc\_fb : Il s'agit de l'adresse à laquelle les données seront stockées ;
* r\_add\_enc\_fb : C'est l'adresse de lecture de la RAM d'encodage ;
* r\_add\_rec : C'est l'adresse de lecture de la RAM de réception.



*Figure 8 : Composant VHDL : FrozenBits*

Afin de mieux comprendre le fonctionnement du système, la machine d'état suivante est utilisée :

* L'état de "veille" : C'est l'état initial du composant, seulement si le signal "start=1" est reçu, il sera possible d'avancer à l'état suivant, sinon il reste indéfiniment dans cet état.
* L'état "nulle" : Cet état est chargé de positionner les zéros dans le vecteur de sortie, pour cela il est nécessaire de savoir dans quelle position les zéros doivent être placés. Pour cette raison, la mémoire ROM est utilisée, dans cette mémoire la position des zéros est récupérée et ceux-ci sont assignés dans le vecteur de sortie. Pour accéder à la ROM, vous devez définir la ligne et la colonne : La ligne correspond au logarithme du message et la colonne au nombre de bits gelés que vous voulez ajouter. La nouvelle valeur est stockée dans le cryptage ROM, pour cette raison, vous n'avez qu'à indiquer l'adresse où vous voulez stocker les informations et un drapeau d'activation qui vous permet de stocker les informations en mémoire.
* L'état de synchronisation permet au système de récupérer un cycle de temps, car l'accès à la RAM provoque un léger retard dans l'analyse de l'information.
* L'état "1" : Effectue une comparaison, si dans le vecteur de sortie le nombre est différent de zéro, cela signifie que le message d'entrée peut être attribué. Pour cette raison, la mémoire RAM de réception est accédée et enregistrée dans la mémoire RAM de cryptage. Ce processus est répété jusqu'à ce que toutes les valeurs du message d'entrée aient été enregistrées dans le vecteur de sortie.



*Figure 9 : Machine d’état du composant FrozenBits*

### Composant : Hadamard

L'objectif de ce composant est de multiplier le message modifié par Frozen Bits, avec la matrice d'Hadamard. Cette multiplication est la dernière étape du processus avant l'envoie du message.

* La matrice Hadamard. La matrice Hadamard est une matrice pouvant augmenter de taille dynamiquement. Pour cela, elle part d'une base 2\*2 : 1 1

1 0

Ensuite, elle utilise cette base pour en construire de plus grande (8\*8 dans cet exemple) :

H4 H4

H4 0

où H4 représente la matrice Hadamard 4\*4.

Puis,

H2 H2 H2 H2

H2 0 H2 0

H2 H2 0 0

H2 0 0 0

où H2 représente la matrice Hadamard 2\*2

Pour au final ;

1 1 1 1 1 1 1 1

1 0 1 0 1 0 1 0

1 1 0 0 1 1 0 0

1 0 0 0 1 0 0 0

1 1 1 1 0 0 0 0

1 0 1 0 0 0 0 0

1 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

* La multiplication

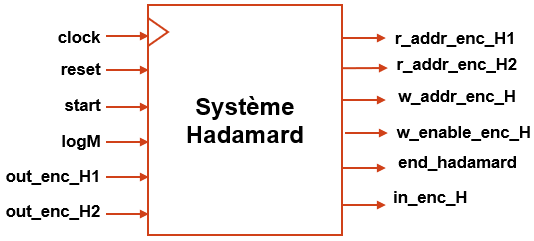
La multiplication se fera comme suit : (message)\*(matrice Hadamard).

* Les contraintes

Afin de respecter les dimensions de la matrice Hadamard, la taille du message doit être de 2^n, où n est un entier entre 1 et 20.

Présentation détaillée :

* Les entrées :
* clock : synchronisation avec l'horloge ;
* reset : remettre les compteurs et la machine d'état à zéro ;
* start : démarre le processus ;
* logM : correspond au logarithme de base 2 de la taille du message de sortie ;
* out\_enc\_H1/2 : correspond aux sorties de la RAM encodage, ces valeurs subiront un XOR.
* Les sorties :
* r\_addr\_enc\_H1/2 : correspond aux adresses envoyées à la RAM encodage pour récupérer les valeurs à ces adresses ;
* w\_addr\_enc\_H : correspond à l'adresse envoyée à la RAM encodage pour écrire le résultat dans la RAM ;
* w\_enable\_enc\_H : active l'écriture de la RAM ;
* end\_hadamard : lève un flag pour prévenir le contrôle que c'est la fin du processus ;
* in\_enc\_H : correspond à la valeur qui va être écrite dans la RAM encodage.



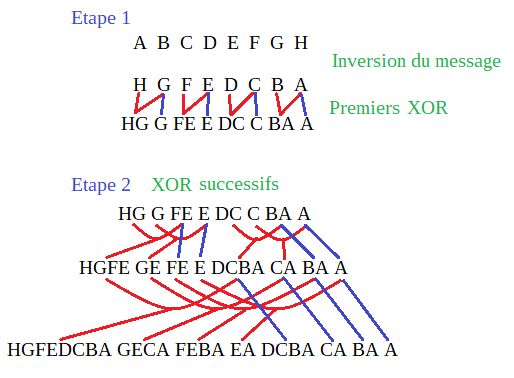
*Figure 10 : Composant VHDL : Hadamard*

Pour l'exemple, le message sera traité avec des lettres, ce qui permet une représentation des messages possibles :

Le message "A,B,C,D" peut représenter "1101", "1111", "0000",.... un XOR entre deux chiffres sera représenté de cette manière :

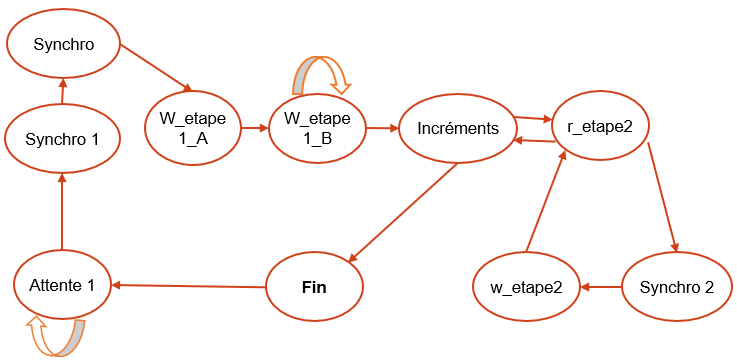
* "AB,C" => "A xor B, C" => "1 xor 1, 0" => "00" => "0 xor 1, 1" => "11"

Soit le message issue de Frozen Bits suivant : "A,B,C,D,E,F,G,H"



*Figure 11 : Les étapes du composant Hadamard*

* La machine d'états

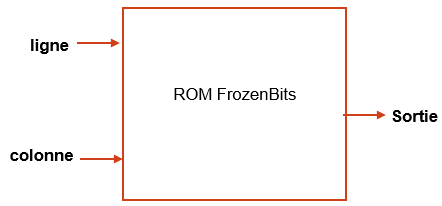


*Figure 12 : Machine d’état du composant Hadamard*

* Attente = St : Attend que les processus avant lui aient tous finis avant de commencer ;
* R\_etape1 = S0lecture : Demande à la ram de lire deux valeurs ;
* Synchro1 = S0att : Laisse passer un coup d'horloge pour avoir le temps de récupérer les deux valeurs de la RAM ;
* W\_etape1\_A = S0ecriture1 : Fait le XOR des valeurs et l'écrit dans la RAM ;
* W\_etape1\_B = S0ecriture2 : replace la deuxième valeur dans la RAM. Selon les compteurs, soit il revient sur R\_etape1, soit il continue sur incréments ;
* Incréments = S1 : modifie les valeurs de i, j et n qui sont utilisés dans l'étape 2 ;
* R\_etape2 = S2lecture : Vérifie le compteur j pour rester sur cet état et demander à la ram de lire deux valeurs, ou revenir à incréments ;
* Synchro2 = S2att : Laisse passer un coup d'horloge ;
* W\_etape2 = S2ecriture : Fait le XOR des valeurs et l'écrit dans la RAM ;
* Fin = Fin : lève le flag de fin pour que contrôle soit averti et active la transmission.

### Composant : Mémoire ROM

La mémoire ROM permet d'obtenir la position où le bit gelé doit être placé. La séquence Q varie en fonction de la taille du message de sortie. Pour cette raison, la ROM doit être affichée comme une matrice. Où le nombre de lignes représente le logN et les colonnes représentent la valeur de 2^n

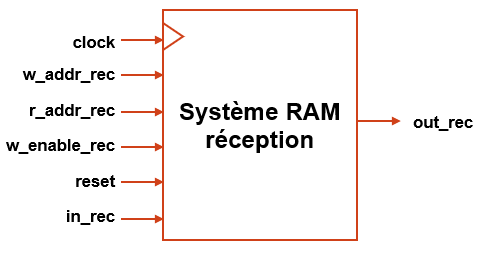


*Figure 13 : Composant VHDL : Mémoire ROM des FrozenBits*

### Composant : Mémoire RAM pour la réception

La mémoire RAM de réception vous permet d'enregistrer le message qui sera crypté. Le processus de sauvegarde des informations s'effectue dans la composante Réception des données. Les signaux suivants commandent le composant :

* w\_enable\_rec : Signal qui permet d'enregistrer des valeurs dans la RAM lorsque celle-ci a la valeur'1' ;
* r\_add\_rec : C'est l'adresse de lecture des données à lire dans la mémoire RAM ;
* w\_add\_rec : C'est l'adresse d'écriture des données qui doit être sauvegardée dans la mémoire RAM ;
* in\_rec : C'est la valeur des données à stocker dans la mémoire ;
* out\_rec : Valeur des données à lire en mémoire.



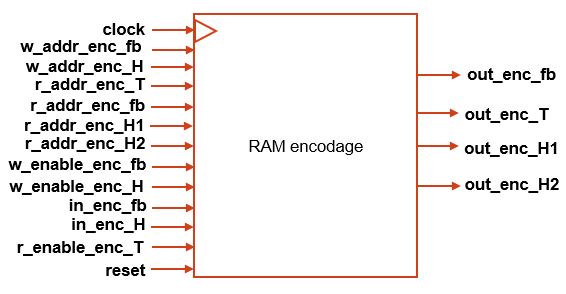
*Figure 14 : Composant VHDL : Mémoire RAM de Réception*

### Composant : Mémoire RAM de l’encodeur

La mémoire de chiffrement est la plus importante, car c’est cette mémoire qui comporte le plus de connexions, puisqu'elle est partagée avec trois composants, FrozenBits, Hadammard et Transmission.

* Les FrozenBits utilisent la mémoire RAM pour enregistrer et lire les valeurs mais sur un seul bit ;
* Hadammard : L'algorithme utilisé dans le composant, doit lire deux valeurs de RAM et à son tour n'a besoin que d'enregistrer un bit en mémoire ;
* Transmission : Comme les messages seront envoyés en une série d'octets, le composant de transmission lira 8 bits de données en même temps.

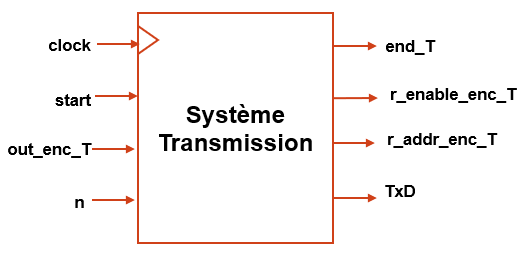
Pour sauvegarder chacune des valeurs il faut compter sur son adresse correspondante, de même pour toute lecture ou écriture de données il faut indiquer l'adresse de la mémoire.



*Figure 15 : Composant VHDL : Mémoire RAM de l’encodeur*

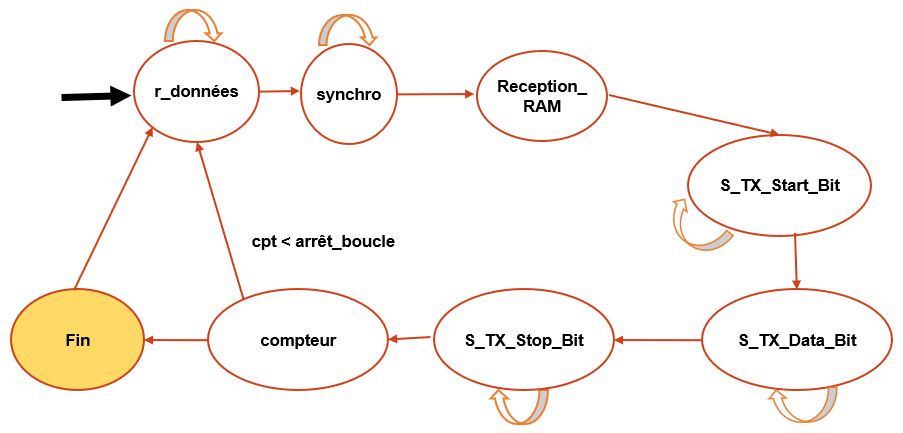
### Composant : Transmission

Ce composant nous permettra d'envoyer le message stocké dans le RAM encodage.



*Figure 16 : Composant VHDL : Transmission*

Le composant suit une machine d'état, qui est composée par :



*Figure 17 : Machine d’état du composant Transmission*

* R\_donnes : Lorsque le composant Hadamard a terminé le traitement des informations, le composant accède à la mémoire RAM pour récupérer les données du signal crypté. Pour cette raison, l'option de lecture 8 bits doit être activée, ainsi que l'adresse du point de départ de la RAM. Il est important de noter qu'en raison de l'algorithme utilisé, les données stockées dans le vecteur sont inversées, c'est pourquoi la première série d'octets à envoyer doit commencer par la taille finale du message et diminuer ensuite jusqu'à la position 7 ;
* Synchronisation : En raison de l'accès à la RAM, un état de synchronisation est défini pour une récupération réussie des données ;
* Recuperaton\_RAM : A ce stade, la valeur lue de la RAM peut être enregistrée dans un signal pour être utilisée. Il arrête également la lecture du composant ;
* s\_TX\_Start\_Bit : Cet état nous permet de traiter l'information en fonction de la vitesse de transfert de données souhaitée. En d'autres termes, l'information sera envoyée jusqu'à ce que le compteur ait atteint la valeur 10416, ce qui correspond à une vitesse de 9600 bauds ;
* S\_TX\_Data\_Bits : Ce processus nous permet d'envoyer chacune des valeurs de notre message (8 bits). Il y a un compteur qui nous permettra d'accéder à chacune des positions du message à envoyer et ces valeurs seront affectées à la variable TxD ;
* S\_TX\_Stop\_bits : Lorsque le processus d'envoi des informations est terminé, selon le protocole de communication UART, la fin de la chaîne doit être spécifiée, c'est pourquoi la variable TxD est mise dans un ;
* Compteur : Cet état définit le nombre de fois que l'information sera envoyée, ceci est fait par un compteur, tant qu'il est inférieur à (n/8), le processus sera cyclique à partir de la récupération des données ;
* Et enfin, l'état final nous permet de réinitialiser les variables et aussi d'indiquer que toutes les données ont été envoyées correctement.

## Les environnements de test

Dans la phase de construction du projet, nous avons construit votre environnement de test. Ce processus identifie le matériel nécessaire, les logiciels, les données de test initiaux, les outils et les connexions réseaux nécessaires pour exécuter les cas de test dans la Phase de Test.

Pour recueillir des données précises et précieuses sur les résultats du test, nous devons maintenir correctement notre environnement de test et sa configuration tout au long de la Phase de Test. Le maintien et l'utilisation efficace des environnements de test tout au long du projet comprennent les éléments suivants :

* Utiliser des outils de gestion d'actifs et des processus pour marquer et suivre tout le matériel et tous les logiciels de test. Nous nous sommes assuré que toutes les modifications sur le matériel et les logiciels soient notées. Certains problèmes qui sont très difficiles à traquer finissent par être le résultat de cas de test fonctionnant sous un autre matériel ou logiciel. Il est important que les résultats des tests qui utilisent le même matériel et les logiciels en tant que tests préalables soient validés. Sinon, ces résultats seront sujets à des erreurs dans l'environnement, pas nécessairement dans la solution elle-même.
* En plus de la gestion des actifs et des configurations, l'intégrité de l'environnement de test complet devrait être maintenue. Dans de nombreuses organisations, l'environnement de test est en fait divisé et isolé de l'environnement de développement afin que les travaux de développement en cours n'aient pas d’impact négatif sur l'environnement de test. L'avantage de cette approche consiste à ce que l'environnement de test puisse mieux refléter l'environnement utilisé par le client. Néanmoins, si les environnements de développement et de test ne sont pas exactement les mêmes, il peut être difficile de reproduire quelques-uns des problèmes rencontrés dans l'environnement de test. Dans certains cas, un test peut très bien fonctionner dans l'environnement de développement, mais échouer dans l'environnement de test, et il peut être difficile d’en retrouver la raison.

# Résultats

Actuellement, nous avons trois « parties » opérationnelles :

* La première correspond à k=5 et n=8. L'utilisateur peut entrer le message dans l’encodeur via une interface et visualiser instantanément le résultat (Dossier : 2\_R\_FB\_H\_T\_k5\_n8).
* La deuxième partie coïncide à l'encodeur dynamique, sans tenir compte de la réception. Autrement dit, la mémoire RAM est initialisée avec plusieurs valeurs et l'utilisateur peut sélectionner la taille du message désirée en utilisant une interface. Le résultat est reçu par cette même interface, cependant, comme la RAM est initialisée seule la valeur initiale et une valeur de RESET peuvent être chiffrée. Cela s'est fait de cette façon, car, en essayant d'intégrer le volet réception, on s'est heurté à des difficultés techniques (Dossier : 3\_FB\_H\_T\_dynamique\_test2).
* Enfin la dernière partie, c’est le système intégral, cependant, le composant de réception ne fonctionne pas intégralement, donc, ce système ne fonctionne qu'avec 8 bits. Si l’on essaye avec des messages plus volumineux, la taille du message souhaité est la même que celle de la sortie, mais le cryptage n'est pas correct. C'est pourquoi l'intégration totale du projet est limitée à une taille de k=7. Si ce système est limité c’est parce que les valeurs reçues par l'utilisateur ne sont pas enregistrées correctement. Pour cette raison, le calcul effectué par nos composants n'est pas adéquat (Dossier : 4\_R\_FB\_H\_T\_dynamique).

# Conclusion

La partie « encodage » a été réalisée de façon satisfaisante, cependant, il y a eu quelques problèmes dans la réception des données, avec les résultats supérieurs à 8 bits.

Le projet nous a permis de renforcer notre capacité d'analyse pour la résolution de problèmes, en nous permettant d'abord de trouver des algorithmes pour le fonctionnement du programme, puis la conception de structures de contrôle pour finalement réaliser l'étape de programmation.

Bien que chacun des composants fonctionne parfaitement séparément, lorsque l'intégration de chacun des composants est réalisée, cela doit se faire progressivement, car il y a parfois des problèmes au moment de l'intégration. L'avantage de le faire progressivement est de trouver les erreurs à temps et d'être capable de les résoudre.

Enfin, la mise en œuvre du projet nous a permis d'améliorer nos compétences en VHDL et surtout de savoir quelles structures sont synthétisables et lesquelles ne le sont pas. C'est pourquoi il est important de raisonner avec des structures robustes qui nous permettent de résoudre nos problèmes et surtout qu'elles soient synthétisables.

# Feedback personnels

## José JAVIER :

Le projet a été un grand défi, surtout parce que le temps d'exécution n'était pas suffisant. J'ai dû investir plus de 90 heures pour développer le projet.

Cependant, je suis totalement satisfait car, avec mon collègue Gaétan Michaud, nous avons pu réaliser la majeure partie du projet VHDL et nos connaissances techniques se sont donc grandement enrichies. Les points d'amélioration de notre projet seraient la modification de la composante réception et l'optimisation du programme en identifiant les variables, opérations et processus redondants ou inutilisables.

## Gaëtan MICHAUD :

Ce projet m'a apporté de la confiance par rapport à mes capacités à trouver des solutions face à un problème conséquent. Il m'a aussi apporté une meilleure maîtrise du VHDL, langage que l'on avait vu au premier semestre mais uniquement jusqu'à l'étape de simulation. Enfin, j'ai pu voir comment une bonne communication permet de faire avancer le projet rapidement, et à l'inverse, comment une mauvaise communication entraîne un ralentissement conséquent du projet.

Si ce projet est refait, il serait peut-être intéressant de développer un peu plus l'utilisation de cette technique et voir un peu plus le contexte avant de commencer le projet, afin de pouvoir se plonger dans le projet plus facilement.