

Rapport Final - Conception Circuit Numérique

Tony PEAUT & Nolan BUCHET

Novembre 2025



POLYTECH[®]
NANTES



**Pôle Sciences et technologie
Nantes Université**

Recepteur de Transmission LIN

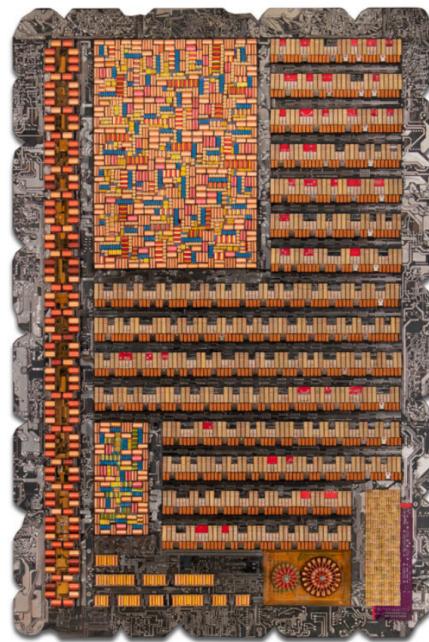


Table des matières

1	Introduction	4
2	Protocole LIN	5
3	Cahier des Charges	7
4	Description des différentes spécifications définies en travaux dirigés	10
4.1	Interface Microprocesseur	11
4.2	Reception Trame	11
4.3	FIFO	11
4.4	ETAT	12
5	Description et justification de la structure fonctionnelle	13
5.0.1	Interface Reception Trame	15
5.0.2	Interface Microprocesseur	16
5.0.3	FIFO	17
5.0.4	ETAT	17
6	Description et justification de la solution architectureale obtenue pour le circuit	19
6.1	Horloge	20
6.2	Architecture Reception Trame	20
6.3	Architecture Interface MicroProcesseur	24
6.4	Architecture FIFO	27
6.5	Implémentation de l'État Interne	28
7	Présentation du fonctionnement des fonctions	30
7.1	Interface MicroProcesseur	30
7.1.1	Synchronisation des Entrées	30
7.1.2	Réseau Combinatoire de Sortie	31
7.1.3	Réseau Combinatoire d'Entrée	31
7.1.4	Réseau Synchronisé de Sortie	32
7.2	Interface de Réception LIN	33
7.2.1	Partie opérative	33
7.2.2	Partie Commande	34
7.3	FIFO	44
7.4	Etat Interne	45
7.5	Réception LIN Complete	47
8	Simulation des fonctions	48
8.1	Interface Microprocesseur	48
8.1.1	Déclarations et signaux	48
8.1.2	Instanciation du composant testé	48
8.1.3	Environnement de test	48
8.1.4	Stimuli supplémentaires	49
8.1.5	Analyse du chronogramme de simulation	49
8.2	Interface Reception LIN	50
8.2.1	Déclarations et signaux	50
8.2.2	Instanciation du composant testé	50
8.2.3	Environnement de test	50

8.2.4	Stimuli supplémentaires	50
8.2.5	Analyse du chronogramme de simulation	53
9	Synthèse des fonctions	54
9.1	Interface Microprocesseur	54
9.2	Interface Reception Trame	57
9.3	FIFO	57
9.4	EtatInterne	57
9.5	Reception LIN	58
10	Routages des Fonctions	59
10.1	Interface Microprocesseur	59
10.2	FIFO	64
10.3	EtatInterne	64
10.4	Interface Reception Lin	65
11	Conclusion	70
12	Annexes	71
12.1	Testbench InterfaceMicroprocesseur	71

1 Introduction

Le projet réalisé dans le cadre de l'enseignement de Conception de Circuits numériques a pour objectif de développer des compétences essentielles à la conception de systèmes embarqués, notamment la mise au point d'un circuit utilisant un composant logique programmable.

L'architecture électronique d'un véhicule repose sur une organisation de calculateurs distribués. L'exemple retenu s'inspire du fonctionnement d'un calculateur embarqué dans la portière d'une automobile, chargé de la gestion des rétroviseurs et des vitres électriques.

Dans ce contexte, deux sous-ensembles sont distingués :

- un sous-ensemble de supervision, qui génère les commandes pour les moteurs des rétroviseurs et des vitres électriques,
- un sous-ensemble d'interface, assurant la communication entre le sous-ensemble de supervision et les autres calculateurs du véhicule.

Ce rapport se concentre exclusivement sur ce second sous-ensemble, l'interface microprocesseur, afin d'étudier son rôle et sa conception.

L'un des objectifs principaux est d'appréhender la conception du circuit via la méthode MCSE (Méthode de Conception de Systèmes Électroniques), en mettant l'accent sur les étapes de spécifications et de conception. Le déroulement du rapport suit la logique du diagramme en Y.

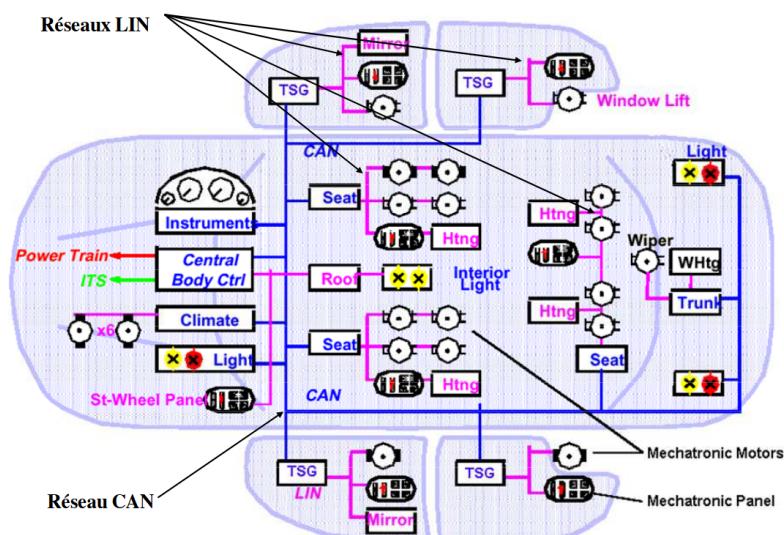


FIGURE 1 – Exemple d'architecture d'un réseau dans un véhicule

2 Protocole LIN

Architecture

Le bus LIN est un système *mono-maître et multi-esclaves*. Un seul maître initie toutes les communications, ce qui rend inutile toute fonction d'arbitrage. Le nombre d'esclaves n'est pas limité par la norme mais dépend des contraintes électriques. L'architecture est dite *flexible*, car on peut ajouter des noeuds esclaves sans modifier les noeuds existants.

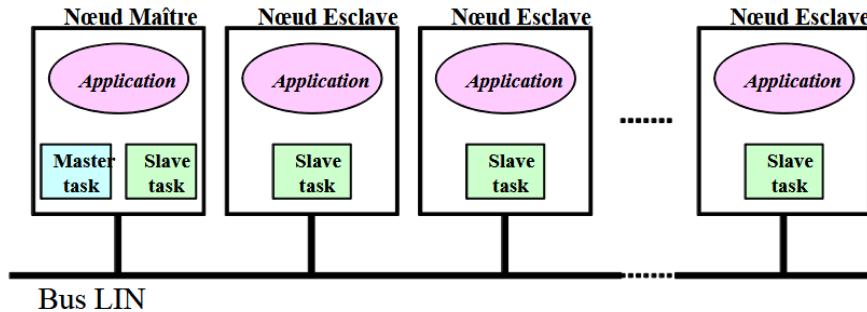


FIGURE 2 – Exemple d'architecture LIN

Connexion

Le bus est constitué d'une *seule ligne* reliée à chaque noeud par une sortie à collecteur ouvert. Le maître utilise une résistance de tirage de $1\text{ k}\Omega$, tandis que chaque esclave utilise $30\text{ k}\Omega$. La ligne est au niveau *récessif* (1) lorsqu'aucun noeud ne force l'état, et au niveau *dominant* (0) dès qu'au moins un noeud impose ce niveau.

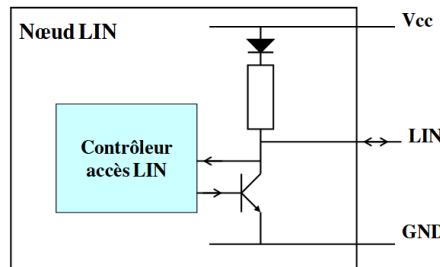


FIGURE 3 – Connexion physique d'un noeud à la ligne LIN

Vitesse de transmission

Le débit varie de 1 kbit/s à 20 kbit/s, fixé pour une architecture donnée. Trois vitesses sont recommandées :

- **Lente** : 2400 bit/s,
- **Moyenne** : 9600 bit/s,
- **Rapide** : 19200 bit/s.

Communications et trames

Les messages LIN sont composés de plusieurs champs :

- *Synchronisation Break* : marque le début du message,

- *Synchronisation Field* : alignement des horloges (valeur 0x55),
- *Identification Field* : contenu et longueur des données, avec contrôle de parité,
- *Data Field* : octets d'information transmis du LSB vers le MSB,
- *Checksum Field* : somme de contrôle des données (modulo 256 inversée).

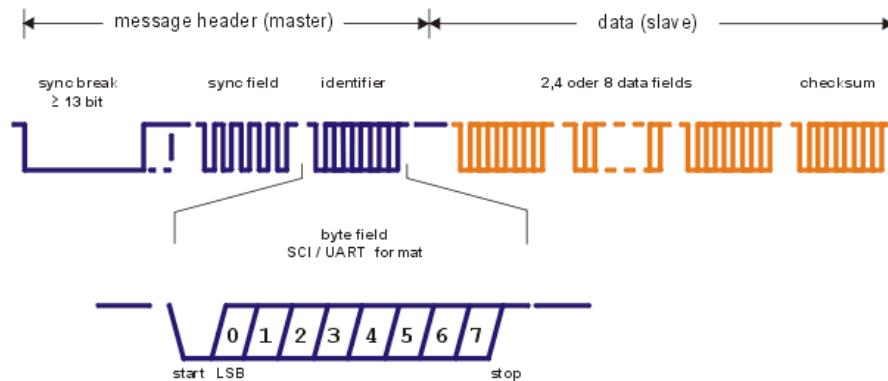


FIGURE 4 – Type de Trame Protocol LIN

Une communication peut être de deux types :

- **Écriture** : le maître envoie l'intégralité du message,
- **Lecture** : le maître envoie seulement l'entête, puis reçoit la réponse de l'esclave.

3 Cahier des Charges

Le projet se concentre sur une partie restreinte du récepteur LIN, uniquement pour la réception de trames de type « **écriture** », avec une **entrée LIN unique** et une vitesse fixée à 19 200 bit/s. La distinction maître/esclave et la connexion physique complète ne sont pas traitées.

Limitations et simplifications :

- Pas de gestion de perte d'octets,
- Vérification des bits start/stop par un seul échantillon,
- Pas de contrôle de parité ni de vérification du checksum.

Fonctionnalités attendues :

- Conversion série → parallèle (données de 8 bits) pour un microprocesseur,
- Possibilité de **filtrer les messages** grâce à un registre de comparaison **SelAddr** (8 bits),
- Signalisation de fin de réception (**M_Received**) uniquement si l'identifiant reçu correspond à **SelAddr**,
- Réinitialisation des compteurs et effacement des messages non valides.

Gestion des messages :

- Un seul message peut être stocké à la fois (FIFO),
- Les octets doivent être accessibles dans leur ordre d'arrivée, même si le message est encore en cours de réception,
- Tous les octets doivent être mémorisés, indépendamment du filtrage,
- Le récepteur doit déterminer la fin du message et l'indiquer au microprocesseur.

État du récepteur :

Accessible par registre (ETAT) à tout moment, il doit indiquer :

- si un message a été reçu (après filtrage),
- le nombre d'octets reçus,
- les erreurs simples de réception (bits START/STOP, durée du *synchro break*).

Après lecture du registre d'état, les champs sont réinitialisés (sauf le compteur d'octets reçus).

Contraintes supplémentaires :

- Interface physique avec le microprocesseur imposée,
- Caractéristiques fonctionnelles, physiques et temporelles définies,
- Temps d'échanges précisés pour assurer la compatibilité avec l'environnement.

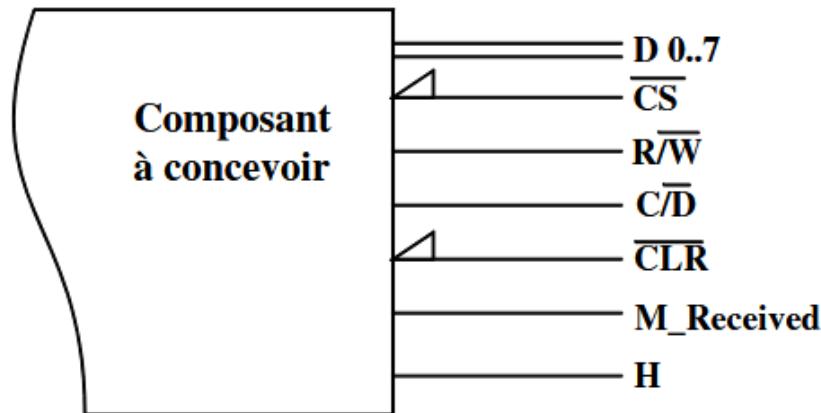


FIGURE 5 – Interface microprocesseur associée au circuit à concevoir

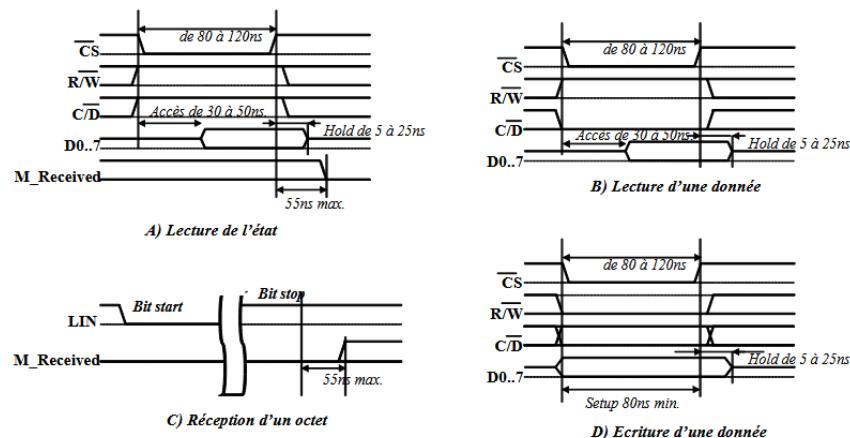


FIGURE 6 – Chronogrammes des échanges entre le circuit et son environnement

Ces figures illustrent les interfaces et les chronogrammes des échanges entre le circuit à concevoir et son environnement, mettant en évidence les interactions avec le microprocesseur et les timings associés.

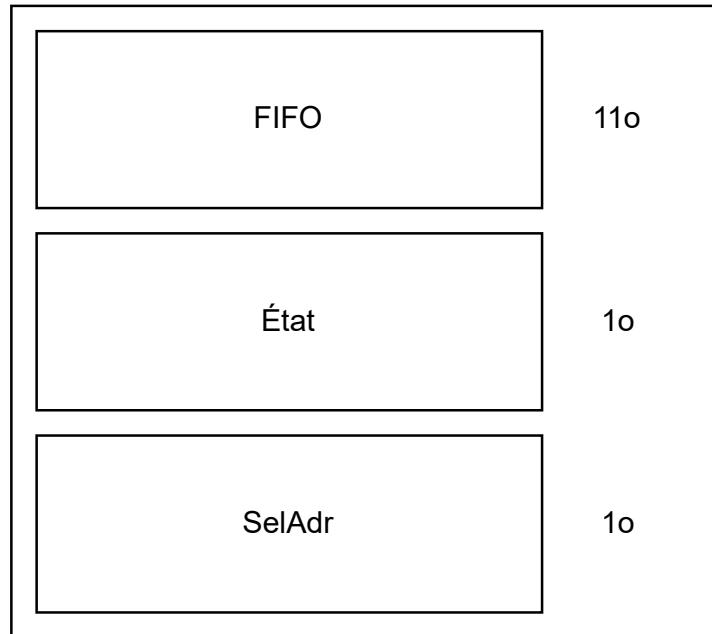


FIGURE 7 – Schema Conception Registre interne Système

Ce schéma détaille la conception des registres internes du système, incluant les registres de données, d'état et de sélection d'adresse.

4 Description des différentes spécifications définies en travaux dirigés

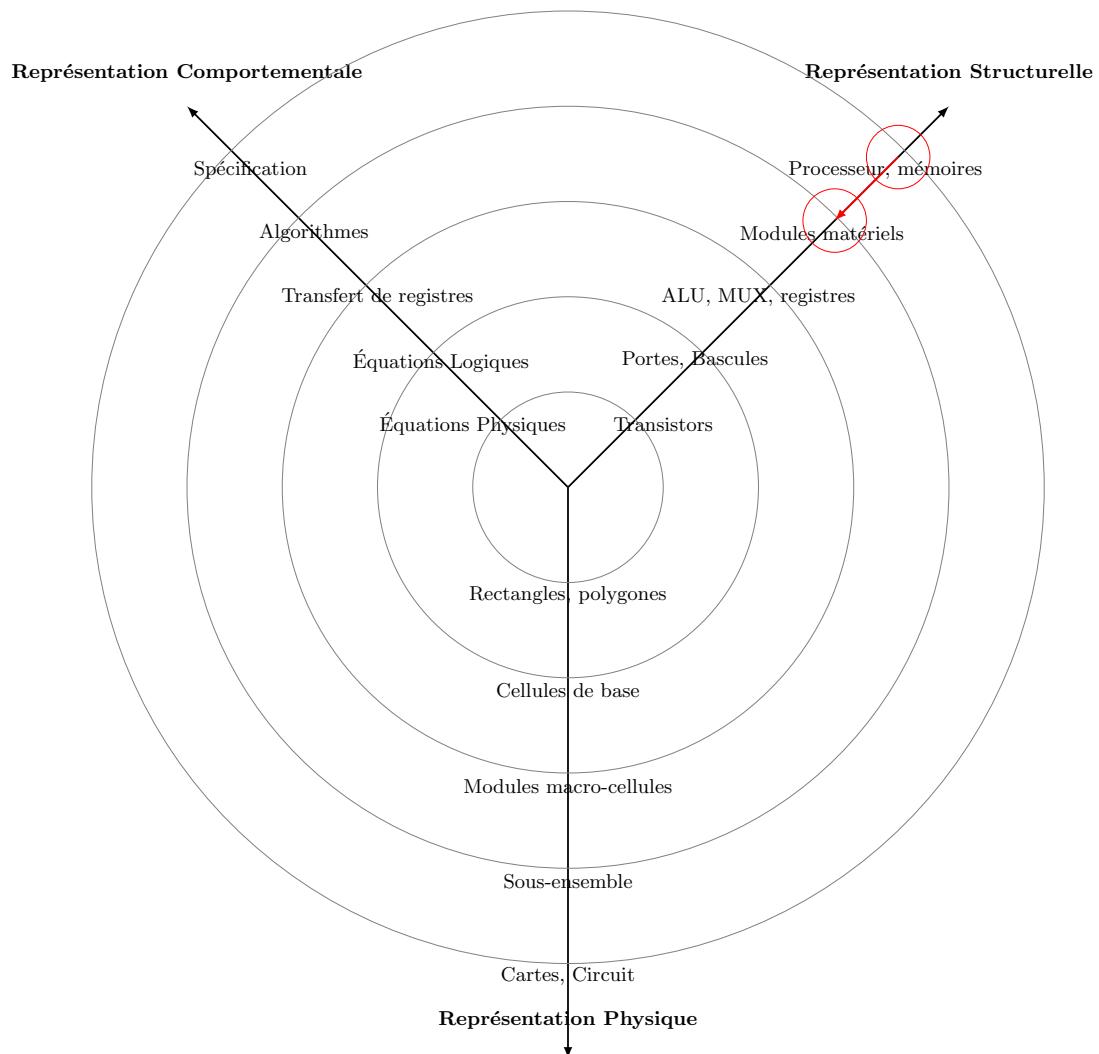


FIGURE 8 – Diagramme en Y - Spécification du circuit vers conception fonctionnelle

Objectif

Les spécifications ont pour objectif de définir le comportement attendu du système, autrement dit, ce qui doit être réalisé en réponse au cahier des charges. Elles constituent l'expression formelle des besoins fonctionnels, en se plaçant du point de vue de l'environnement du système, c'est-à-dire, tout ce qui est externe au circuit mais interagit avec lui.

La phase de spécification représente la première étape de la conception d'un circuit. Elle adopte une approche boîte noire : on s'intéresse uniquement à ce que le système doit faire, sans se préoccuper des solutions techniques internes ni du comment il fonctionnera. Les spécifications sont donc indépendantes de la technologie utilisée.

Au départ, nous avions pour indication de créer un bloc capable de communiquer à la fois avec le système de trame LIN et avec le microcontrôleur. Afin de simplifier notre étude et nos spécifications, nous avons choisi de diviser le système en deux parties : la réception de trame, qui se chargera de gérer la réception des données octet par octet, et un autre bloc dédié à la communication avec le microcontrôleur.

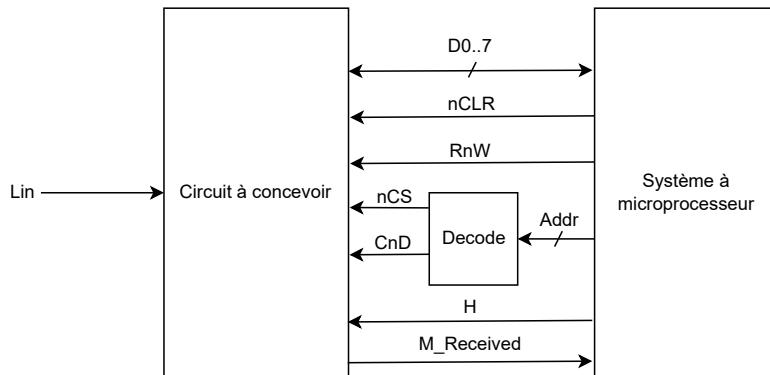


FIGURE 9 – Descetip au circuit à concevoir

4.1 Interface Micropuce

D'après les données du cahier des charges, nous pouvons définir tous les signaux nécessaires pour le système réception trame :

Signaux	Mode	Type	Description
D07	Bidirectionnel	Données	Bus de données
nCS	Entrée	Commande	Chip Select
RnW	Entrée	Commande	Opération Lecture / Ecriture
CnD	Entrée	Commande	Opération Contrôle / Données
nCLR	Entrée	Commande	Réinitialisation
M_Received	Sortie	Commande	Fin de réception de la trame
H	Entrée	Synchronisation	Horloge

4.2 Reception Trame

D'après les données du cahier des charges, nous pouvons définir tous les signaux nécessaires pour le système interface Reception Trame :

Signaux	Mode	Type	Description
LIN	Entrée	Données	Reception de la trame LIN

4.3 FIFO

Pour le moment, nous savons qu'il s'agit d'un registre de stockage fonctionnant en mode FIFO, destiné à mémoriser les données de réception d'une trame LIN :

Signaux	Mode	Type	Description
OctetRecu	Entrée	Données	Signal Entrée
OctetLu	Sortie	Données	Signal Sortie

4.4 ETAT

Ce registre peut être considéré comme le registre de log de la trame. Grâce au cahier des charges, nous connaissons précisément ses fonctionnalités :

Signaux	Mode	Type	Description
Erreur_Start	Entrée	Commande	Bit d'erreur de Start
Erreur_Stop	Entrée	Commande	Bit d'erreur de Stop
Erreur_SynchroBreak	Entrée	Commande	Bit d'erreur de Synchro Break
NbOctetReceived	Entrée	Données	Nombre d'octets reçus
MessageReceived_SET	Entrée	Commande	Indicateur de trame reçue
EtatLu	Sortie	Données	Octet d'information de la trame

5 Description et justification de la structure fonctionnelle

Objectifs

Cette section présente l'organisation du système en blocs fonctionnels et décrit leurs interactions. Chaque bloc (réception de trame, FIFO, registre d'état, interface microprocesseur) est expliqué dans son rôle et sa contribution au fonctionnement global. L'objectif est de montrer comment les fonctionnalités spécifiées sont réparties de manière logique pour répondre au cahier des charges.

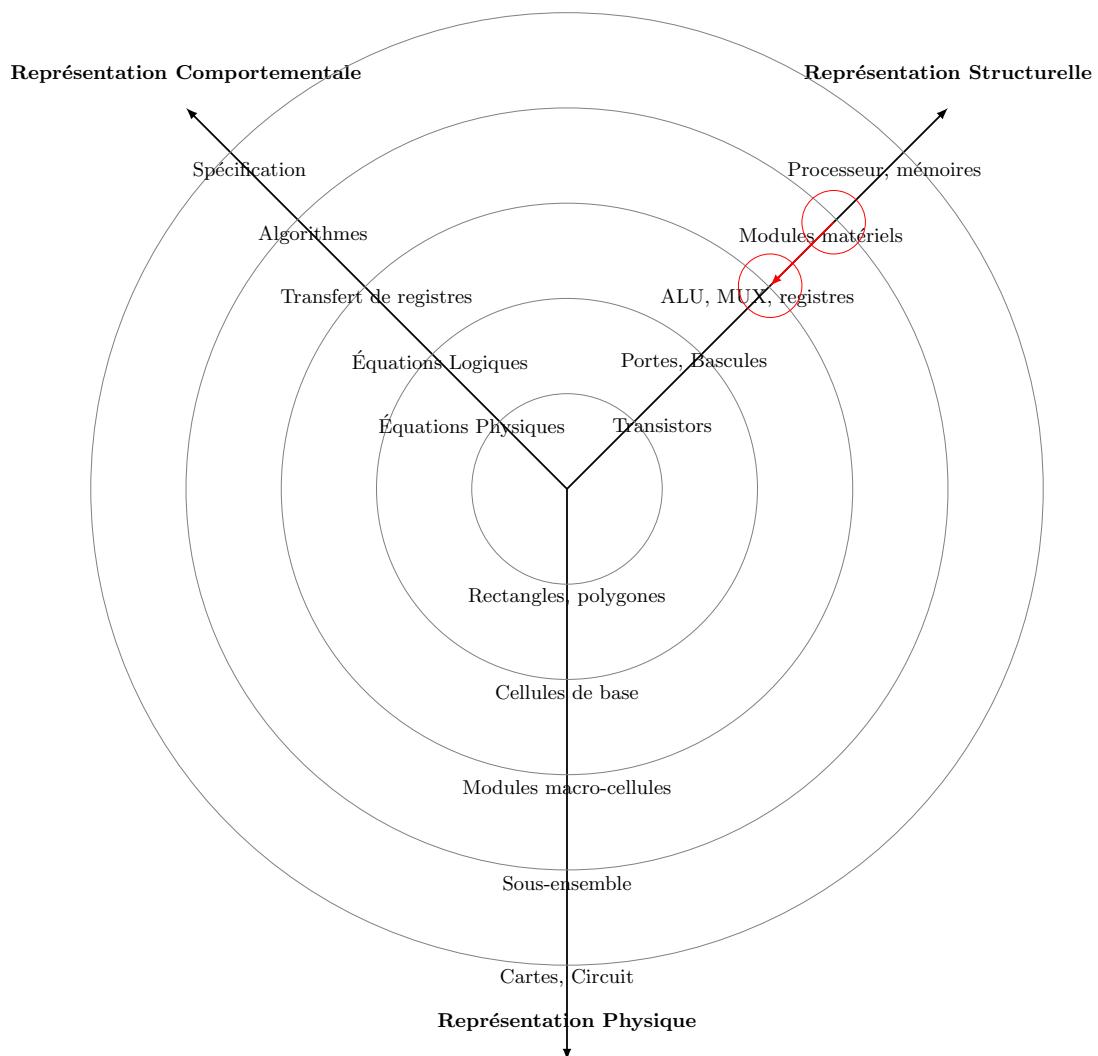


FIGURE 10 – Diagramme en Y - Conception fonctionnelle vers conception architecturale

Dans cette section, nous établissons une structure fonctionnelle indépendante de la technologie.

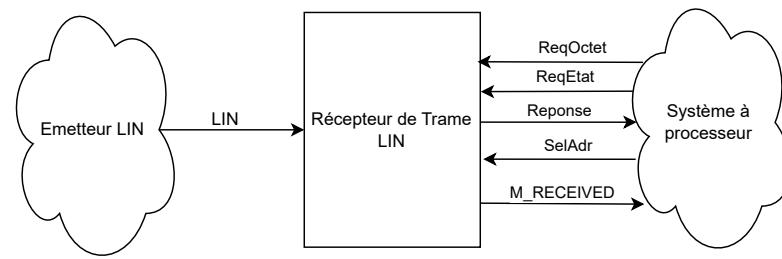


FIGURE 11 – Représentation des échanges avec les entités Émetteur LIN et Système à processeur

Pour le moment, nous nous sommes limités à deux blocs principaux : l’interface microprocesseur et la réception des trames. Ces deux blocs sont connectés à un bloc d’échange, qui permet la communication entre eux. Ce bloc assure l’interprétation des échanges avec le microprocesseur ainsi que la gestion des deux registres de données.

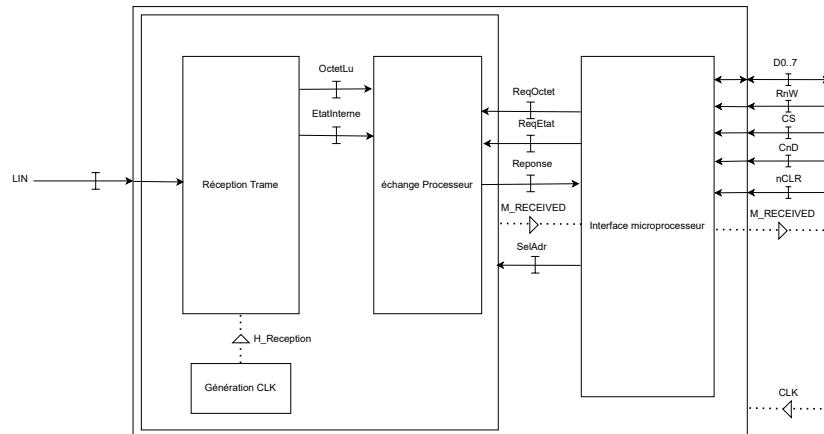


FIGURE 12 – Solution fonctionnelle après introduction des interfaces LIN V1

Dans notre démarche d’optimisation du système, nous nous sommes rendu compte que le bloc d’échange microprocesseur pouvait être intégré au bloc d’interface. Cela permettra de supprimer des signaux supplémentaires, inutiles et encombrants.

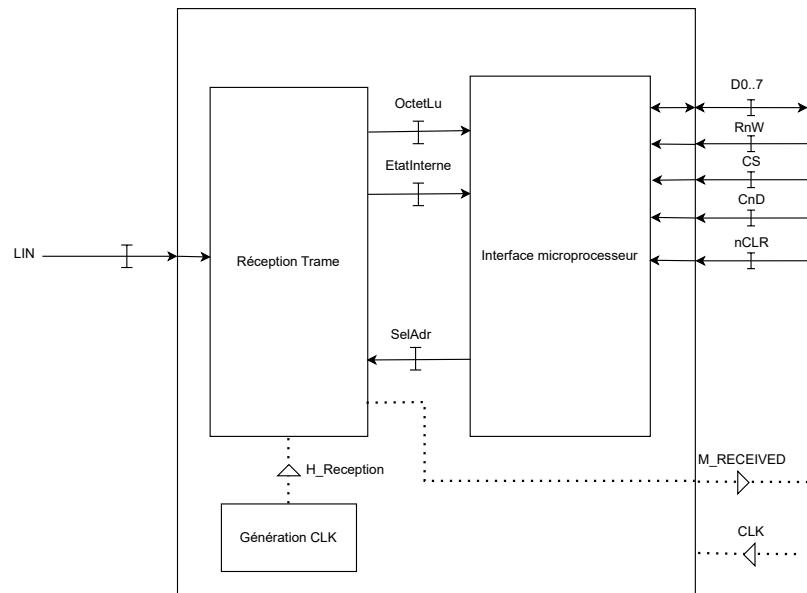


FIGURE 13 – Architecture du système de réception de trame LIN V2

Par la suite, nous avons décidé d'ajouter deux blocs de registres internes : le registre de stockage des données de la trame LIN (FIFO) et le registre d'état interne (ETAT), qui permet de connaître les informations sur l'état de la réception.

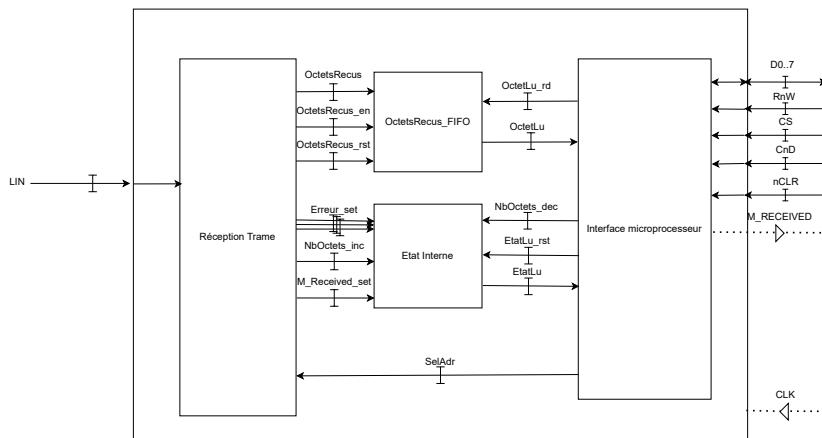


FIGURE 14 – Description finale du circuit

Comme indiqué ci-dessus, dans le schéma de notre système global, nous avons décidé d'ajouter certains signaux internes entre les blocs de registres et les interfaces.

5.0.1 Interface Reception Trame

De même, nous pouvons représenter l'automate du système de réception de trame LIN :

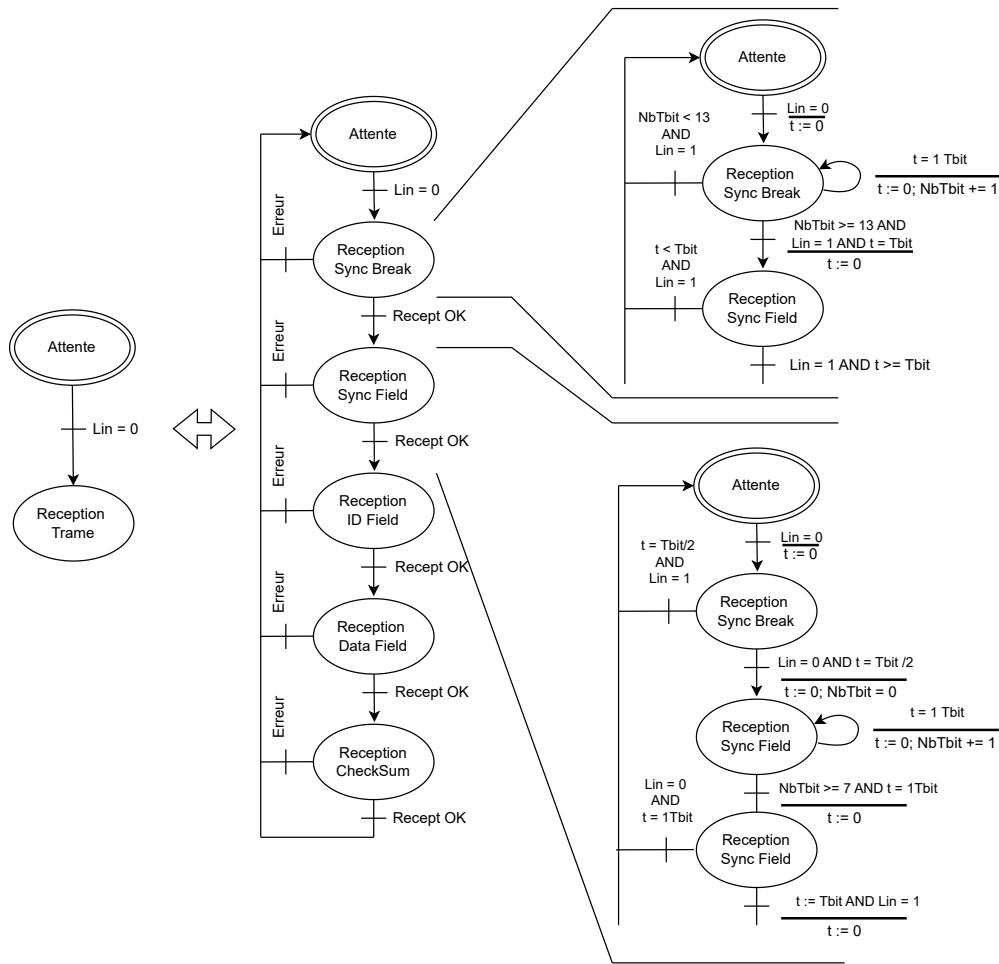


FIGURE 15 – Comportement du circuit vis à vis de l'émetteur LIN

À ce niveau de détail, les automates des états *Sync field*, *ID field*, *Data field* et *Checksum field* sont identiques.

On retrouve :

- Attente (bit de start)
- Réception du bit de start
- Réception des bits de data
- Réception du bit de stop
- 1 état d'attente

Nombre total d'états :

- Réception Sync Break : 2 états
- Autres champs : 4×4 états

Total : 18 états

5.0.2 Interface Micropuce

De même, nous pouvons représenter l'automate du système d'échange micropuce :

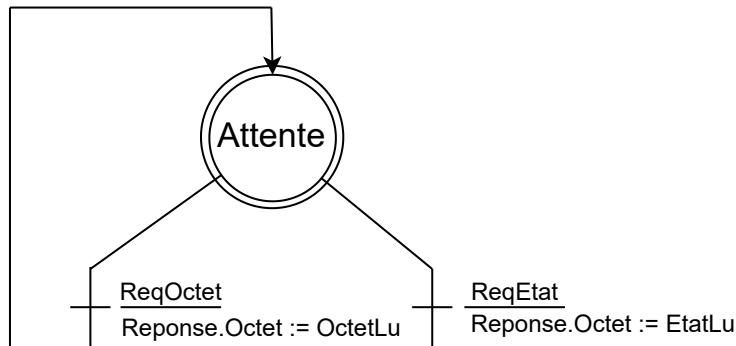


FIGURE 16 – Comportement du circuit vis à vis du microprocesseur

2 Solution Possibles :

- ReqOctet : Lecture de l'octet
- ReqEtat : Lecture de l'état

5.0.3 FIFO

Signaux	Mode	Type	Description
OctetReçu_WR	Entrée	Commande	Read / Write opération
OctetReçu_RST	Entrée	Commande	Réinitialisation des données reçue
OctetLu_RD	Entrée	Commande	Sélection de la mémoire (Control / Data)

La définition de ces nouveaux signaux permet de gérer de manière précise le fonctionnement de la FIFO. OctetReçu_WR est un signal d'écriture qui déclenche l'enregistrement d'un octet reçu dans la mémoire FIFO, garantissant que chaque donnée entrante est correctement stockée. OctetReçu_RST est un signal de réinitialisation qui vide complètement la FIFO et remet à zéro tous les compteurs internes, permettant ainsi de reprendre la réception de données sans risque de corruption ou de chevauchement. OctetLu_RD est un signal de lecture qui active l'accès aux données stockées dans la FIFO et permet leur transfert vers le microprocesseur ou d'autres blocs du système. Ensemble, ces signaux assurent une communication fiable, synchronisée et organisée entre l'interface microprocesseur et le registre de réception, tout en facilitant la gestion des flux de données entrants.

5.0.4 ETAT

Signaux	Mode	Type	Description
NbOctetReçu_RST	Entrée	Commande	Réinitialisation du compteur d'octets
DecNbOctet	Entrée	Commande	Flag de lecture pour FIFO
EtatLu_RST	Entrée	Commande	Reset de l'état lu

La définition de ces signaux du bloc ETAT permet de contrôler et de suivre l'état interne de la réception des données. NbOctetReçu_RST réinitialise le compteur d'octets reçus, assurant un suivi précis du nombre de données traitées. DecNbOctet agit comme un indicateur de lecture pour la FIFO, signalant quand un octet peut être lu et transféré, ce qui permet une gestion correcte des flux de données. EtatLu_RST permet de réinitialiser les informations d'état lues, garantissant que le système dispose toujours d'une représentation exacte de l'état actuel de la

réception. Ensemble, ces signaux assurent une supervision fiable et synchronisée des opérations internes, facilitant le contrôle et la gestion du flux de données dans le système.

6 Description et justification de la solution architecturelle obtenue pour le circuit

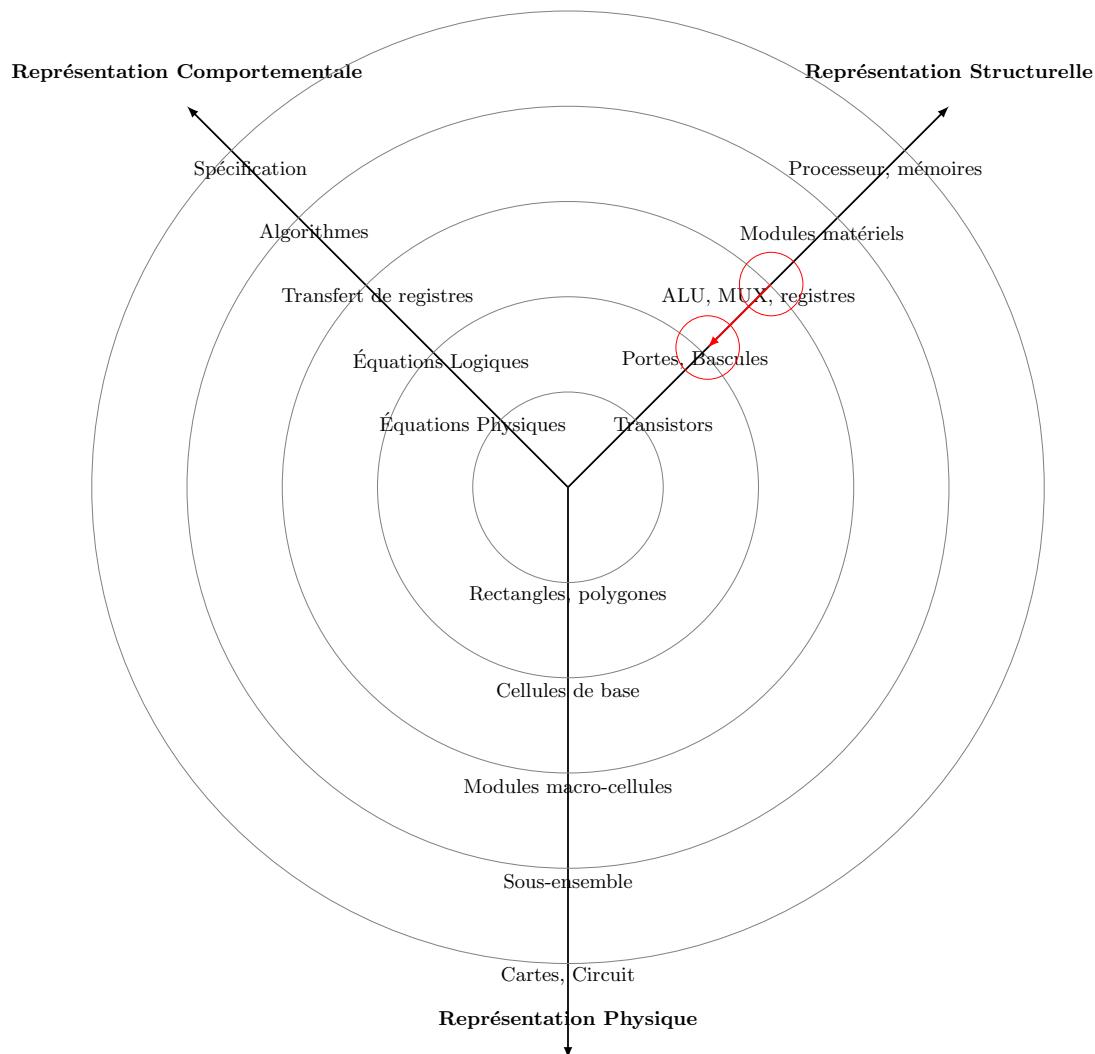


FIGURE 17 – Diagramme en Y - Conception architecturale vers prototypage

Objectifs

Une fois les algorithmes fonctionnels définis, la description fonctionnelle interne du circuit est considérée comme complète. Cette description reste indépendante de la technologie utilisée et ne prend pas nécessairement en compte les réalités physiques, telles que les interférences entre signaux et entités.

La phase de conception architecturale consiste alors à intégrer les interfaces physiques, à analyser les opportunités de simplification des algorithmes, et à définir la stratégie d'implantation du circuit.

Dans cette partie, nous nous intéressons à l'architecture matérielle à partir de la solution fonctionnelle et à la description du circuit au niveau RT (Register Transfer).

- Introduction des interfaces physiques
- Identification des ressources logiques de stockage
- Description structurelle du circuit au niveau RT
- Écriture du comportement du circuit au niveau RT

6.1 Horloge

Nous commençons par l'une des parties les plus simples du système : la gestion de l'horloge. La vitesse de l'horloge est déterminée à partir du pas d'échantillonnage N , défini par :

$$N = \frac{T_{bit}}{T_{processeur}}$$

Dans notre cas, le cycle de lecture/écriture spécifié dans le cahier des charges est en moyenne de 100 ns, avec une vitesse de transmission de 19 200 bit/s. Cela donne :

$$N = \frac{52 \mu s}{100 \text{ ns}} = 520$$

Pour notre implémentation, nous choisissons $N = 2048$, une valeur arbitraire qui facilite la synchronisation et le traitement interne.

6.2 Architecture Reception Trame

Pour rappel voici tous les signaux associés à ce bloc :

Signaux	Mode	Type	Description
LIN	Entrée	Données	Bus de données d'entrée
SelAdr	Entrée	Données	Sélection Address Composant
OctetRecu	Sortie	Données	Bus de données de sortie
OctetRecu_WR	Sortie	Commande	Read / Write opération
OctetRecu_RST	Sortie	Commande	Réinitialisation des données reçues
Erreur_Start	Sortie	Commande	Bit d'erreur de Start
Erreur_Stop	Sortie	Commande	Bit d'erreur de Stop
Erreur_SynchroBreak	Sortie	Commande	Bit d'erreur de Synchro Break
IncNbOctet	Sortie	Commande	Flag de reception pour lecture
MessageReceived_SET	Sortie	Commande	Indicateur de trame reçue
NbOctetRecu_RST	Sortie	Commande	Réinitialisation du compteur d'octets

Pour le bloc de réception de trames, nous implementons une machine séquentielle qui permet de distinguer une partie opérative et une unité de commande, assurant ainsi la gestion efficace de la réception de ce type de trame.

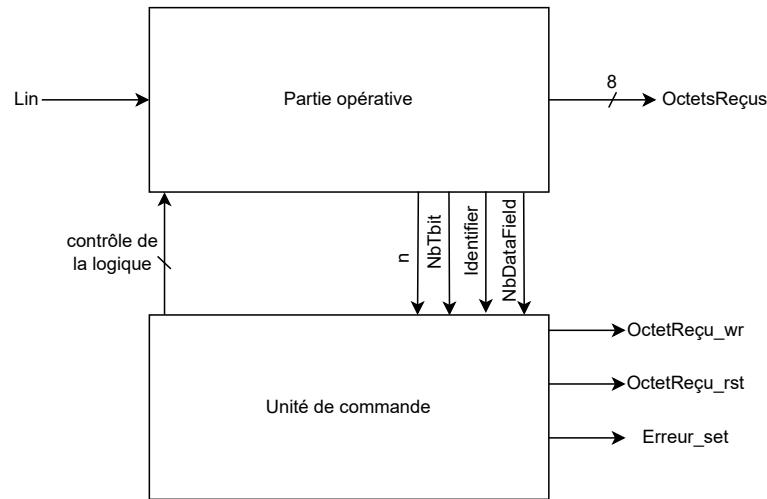


FIGURE 18 – Machine Séquentielle Reception Trame

Nous ajoutons des variable interne pour la communication entre les deux bloc de la machine afin d'obtenir un sytème correcte :

Variable	Taille (bit)	Opération	Opérateur	Signaux de contrôle
n	$\log_2(N)$	décrémentation, initialisation à $N - 1$ ou $N/2$	décompteur, Mux	n_Load, n_En, n_Select
NbTbit	4	décrémentation, initialisation à 13 ou 8	décompteur, Mux	NBTbit_Load, NBTbit_En, NBTbit_Select
Identifier	8	sauvegarde	registre 8 bits	Identifier_En
OctetsReçus	8	décalage bit à bit	registre à décalage	OctetReçu_En
NbDataField	3	décrémentation, initialisation à 1, 3 ou 7	décompteur, décodeur	NBdatafield_En, NBdatafield_Load

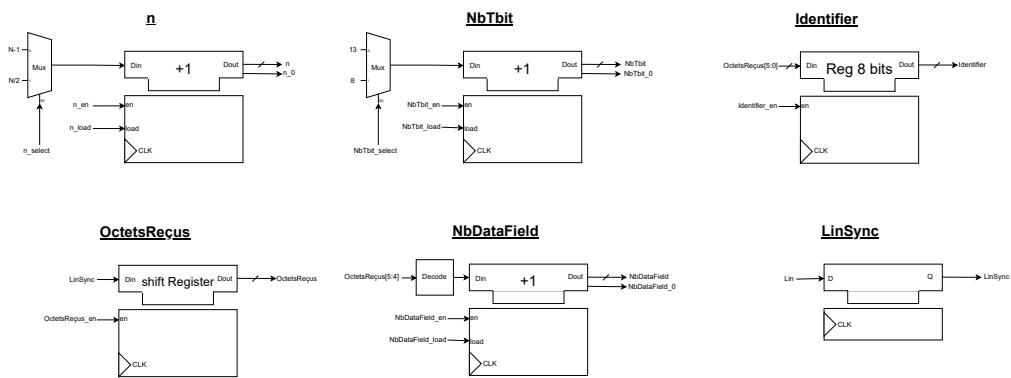


FIGURE 19 – Structure partie Opérative Reception Trame

Dans cette partie nous retrouvons les différents compteurs et registres nécessaires au fonctionnement de la réception de trame. Pour prendre un exemple le premier en haut à gauche représente le décompteur n qui permet de décompter jusqu'à N-1 ou N/2 pour l'échantillonnage. Le choix de N-1 ou N/2 dépend de l'utilisation, soit N/2 pour la detection de l'état au bout de un demi Tbit ou N-1 pour la detection de l'état à la fin de un Tbit (Potentiel Front).

Pour la partie commande, nous avons conçu et implémenté un automate afin de gérer les différentes séquences de contrôle :

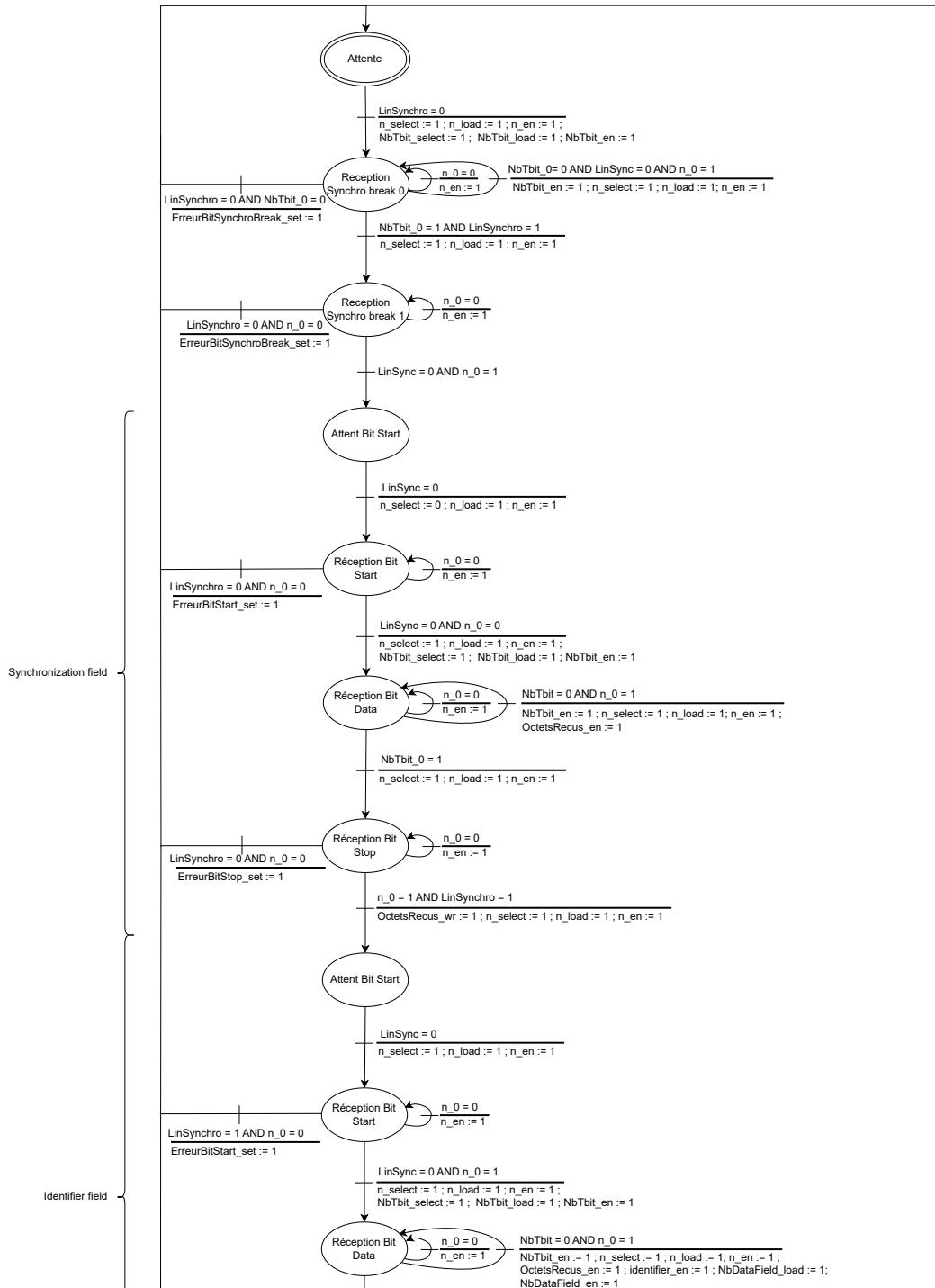


FIGURE 20 – Automate de réception de trame LIN - PART1

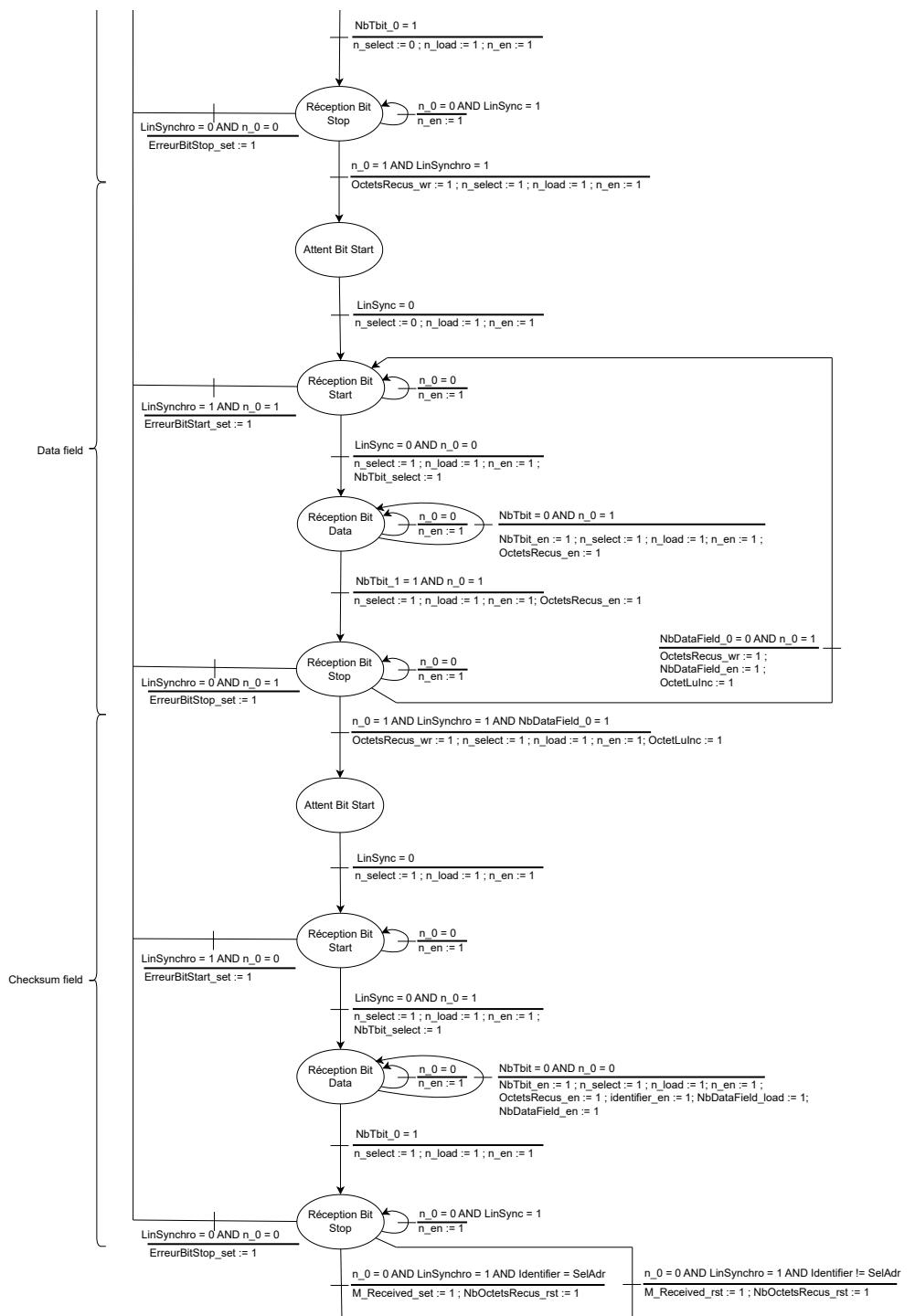


FIGURE 21 – Automate de réception de trame LIN - PART2

Cet automate peut être représenté sous forme de machine de Mealy, ce qui simplifie l'écriture du code VHDL :

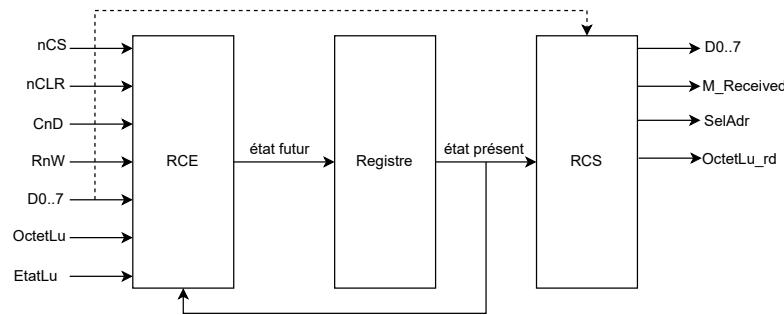


FIGURE 22 – Machine de MEALY Unité de Commande Interface Micro

Description de l'automate

Cet automate décrit un processus de réception de trame LIN (Local Interconnect Network), organisé en états et transitions conditionnées par des signaux de synchronisation, des compteurs et des champs de données. Il commence par une phase d'**attente**, suivie d'une **synchronisation sur break** où la ligne (LinSynchro) est surveillée pour détecter le début de la trame. Des erreurs de synchro ou de bit de start peuvent être signalées via des flags comme ErreurBitSynchroBreak_set ou ErreurBitStart_set.

Ensuite, l'automate passe à la réception des différents champs de la trame :

- **Synchro** (octet de synchronisation),
- **Identifier** (champ d'identifiant),
- **Data field** (données, avec gestion du compteur NbDataField),
- **Checksum** (vérification de l'intégrité).

À chaque octet, le système contrôle le bit de start, les données, et le bit de stop, en décrémentant les compteurs comme NbTbit ou NbOctetsRecus. Selon que l'identifiant reçu correspond ou non à l'adresse sélectionnée (SelAdr), l'automate valide la trame (M_Received_set) ou la rejette. Des réinitialisations (rst) ont lieu après réception complète ou en cas d'erreur.

Ce processus assure une réception séquentielle et robuste, typique des protocoles série, avec gestion d'erreurs et validation conditionnelle des trames.

Cet automate peut être représenté sous forme de machine de Mealy, ce qui simplifie l'écriture du code VHDL :

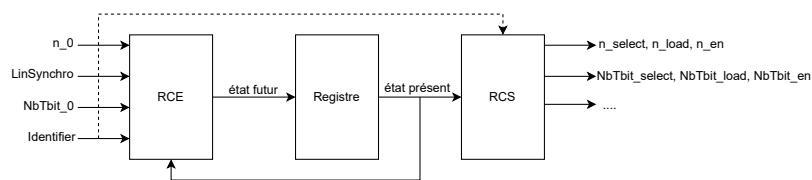


FIGURE 23 – Machine de MEALY Unité de Commande Reception Trame

6.3 Architecture Interface MicroProcesseur

Pour rappel voici tous les signaux associés à ce bloc :

Signaux	Mode	Type	Description
D07	Bidirectionnel	Données	Bus de données
nCS	Entrée	Commande	Chip Select
RnW	Entrée	Commande	Opération Lecture / Écriture
CnD	Entrée	Commande	Opération Contrôle / Données
nCLR	Entrée	Commande	Réinitialisation
M_Received	Sortie	Commande	Fin de réception de la trame
H	Entrée	Synchronisation	Horloge
EtatLu	Entrée	Données	Octet d'information de la Trame
DecNbOctet	Sortie	Commande	Flag de lecture pour FIFO
EtatLu_RST	Sortie	Commande	Reset de l'état lu
OctetLu	Entrée	Données	Bus de données de sortie
OctetLu_RD	Sortie	Commande	Sélection de la mémoire (Control / Data)

Pour le bloc d'interface Microprocesseur, nous implémentons une machine séquentielle qui permet de distinguer une partie opérative et une partie commande, assurant ainsi la gestion efficace du contrôle du système et la lecture des différentes informations.

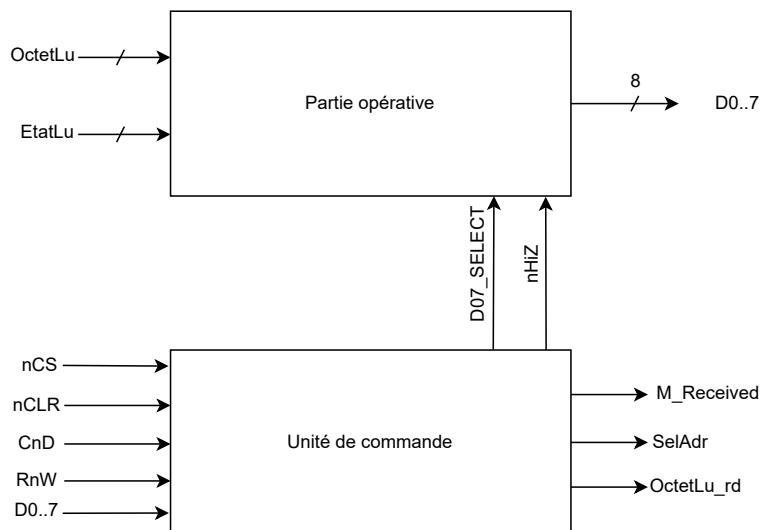


FIGURE 24 – Machine Séquentielle Reception Trame

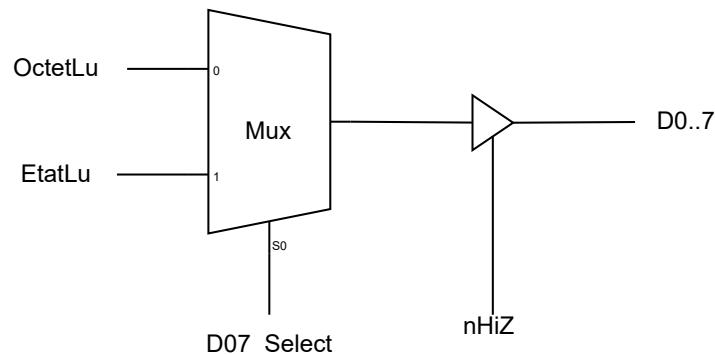


FIGURE 25 – Structure partie Opérative Interface Microprocesseur

Pour la partie opérative de l'interface microprocesseur, nous obtenons un schéma simple de multiplexeur permettant de choisir quelles données mettre dans le vecteur D07 en sortie, soit celles provenant de la FIFO, soit celles de l'État interne.

Pour la partie commande, nous avons conçu et implémenté un automate afin de gérer les différentes séquences de contrôle :

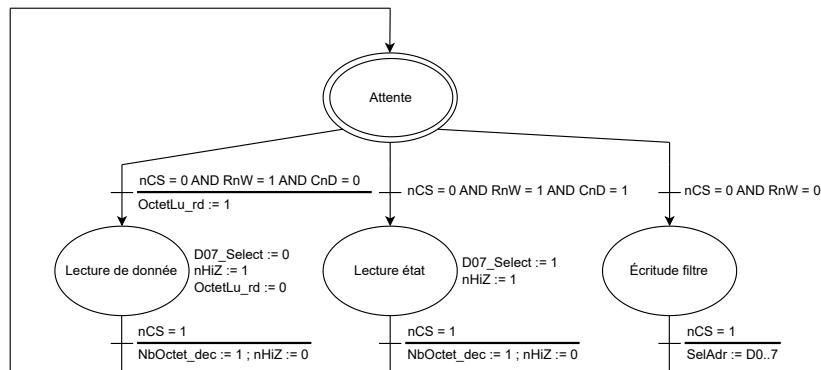


FIGURE 26 – Structure partie Commande Interface Microprocesseur

Cet automate peut être représenté sous forme de machine de Mealy, ce qui simplifie l'écriture du code VHDL :

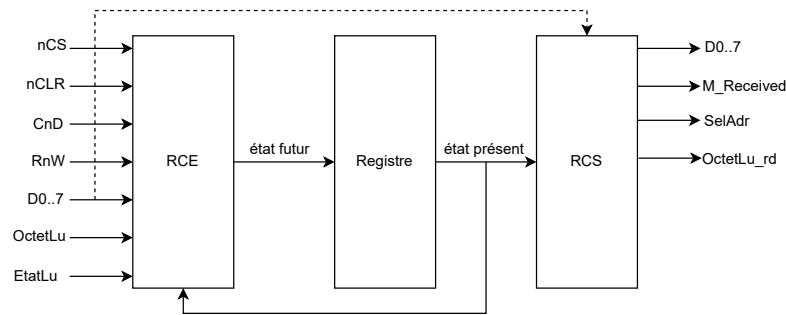


FIGURE 27 – Machine de MEALY Unité de Commande Interface Micro

6.4 Architecture FIFO

Pour rappel voici tous les signaux associés à ce bloc :

Signaux	Mode	Type	Description
OctetReçu	Entrée	Données	Bus de données d'entrée
OctetReçu_WR	Entrée	Commande	Read / Write opération
OctetReçu_RST	Entrée	Commande	Réinitialisation des données reçues
OctetLu	Sortie	Données	Bus de données de sortie
OctetLu_RD	Entrée	Commande	Sélection de la mémoire (Control / Data)

Étant donné que ce système est moins complexe que les deux interfaces précédentes, nous avons choisi de ne pas le réaliser sous forme de machine séquentielle et de le représenter directement comme un bloc, comme montré ci-dessous :

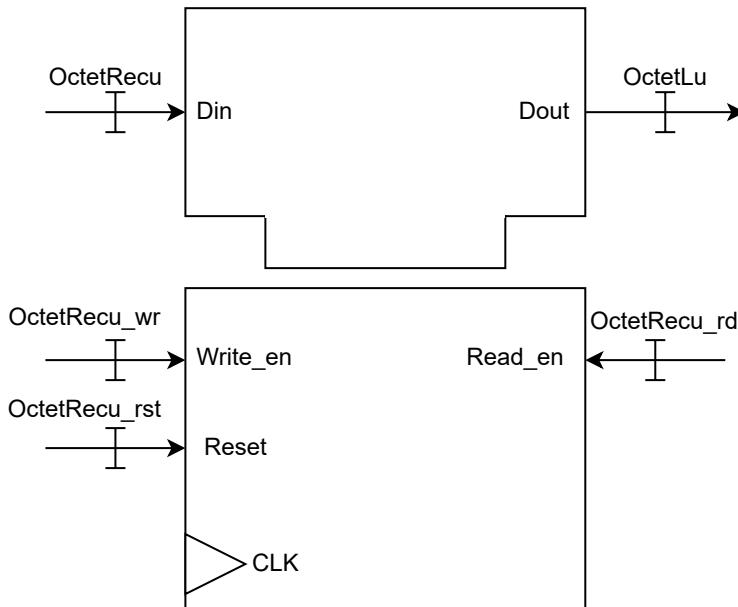


FIGURE 28 – Implémentation de la FIFO

6.5 Implémentation de l'État Interne

Pour rappel voici tous les signaux associés à ce bloc :

Signaux	Mode	Type	Description
Erreur_Start	Entrée	Commande	Bit d'erreur de Start
Erreur_Stop	Entrée	Commande	Bit d'erreur de Stop
Erreur_SynchroBreak	Entrée	Commande	Bit d'erreur de Synchro Break
IncNbOctet	Entrée	Commande	Flag de reception pour lecture
MessageReceived.SET	Entrée	Commande	Indicateur de trame reçue
NbOctetRecu_RST	Entrée	Commande	Réinitialisation du compteur d'octets
EtatLu	Sortie	Données	Octet d'information de la Trame
DecNbOctet	Entrée	Commande	Flag de lecture pour FIFO
EtatLu_RST	Entrée	Commande	Reset de l'état lu

Étant donné que ce système présente une complexité similaire et une facilité d'implémentation comparable, nous le conservons également sous forme de bloc, comme montré ci-dessous :

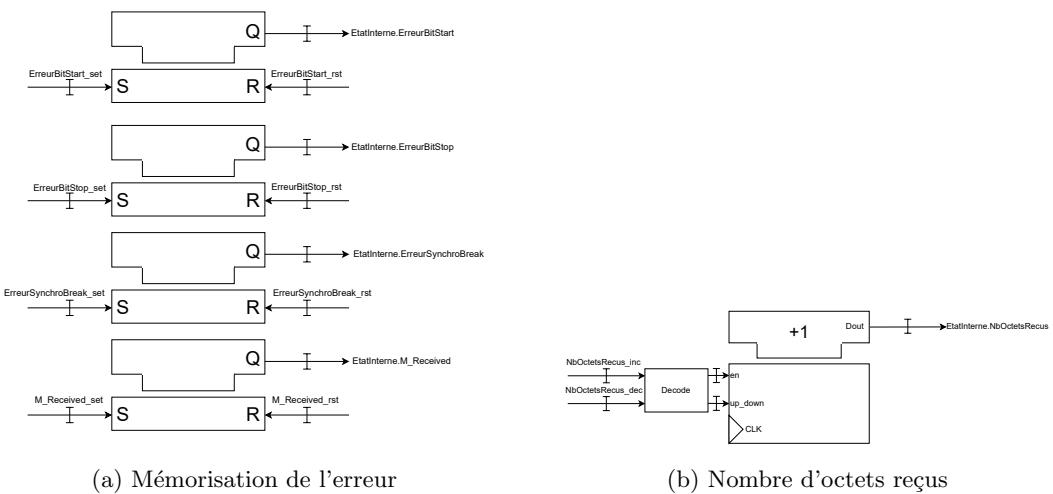


FIGURE 29 – implémentation de l'état interne en représentation structurelle au niveau RT

7 Présentation du fonctionnement des fonctions

7.1 Interface MicroProcesseur

Dans cette partie, nous avons initié une séance de travaux pratiques pour nous familiariser avec le logiciel HDL Designer. Le programme «Interface Microprocesseur», préalablement implémenté par les enseignants, respecte strictement les données présentées dans le TD et développées dans les sections précédentes du rapport. Nous allons l'étudier en détail afin de démontrer sa correspondance avec le modèle théorique.

Pour rappel, l'interface Microprocesseur a été conçue selon une machine séquentielle, tandis que la partie commande a été développée sur le modèle d'une machine de Mealy. Le code présenté respecte rigoureusement la structure des blocs : réseau combinatoire d'entrée, réseau combinatoire de sortie et registres correspondant à la machine à états.

Voici le tableau récapitulant les entrées et sorties du bloc Interface Microprocesseur complet :

Signal	Direction	Type
Cnd	IN	std_logic
EtatLu	IN	std_logic_vector(7 DOWNTO 0)
H	IN	std_logic
OctetLu	IN	std_logic_vector(7 DOWNTO 0)
RnW	IN	std_logic
nCS	IN	std_logic
nRST	IN	std_logic
DecNbOctet	OUT	std_logic
EtatLu_RST	OUT	std_logic
M_Received	OUT	std_logic
OctetLu_RD	OUT	std_logic
SelAdr	OUT	std_logic_vector(7 DOWNTO 0)
D07	INOUT	std_logic_vector(7 DOWNTO 0)

TABLE 1 – Signaux du module Interface Microprocesseur

7.1.1 Synchronisation des Entrées

```

1 InputProc_Synchro :  PROCESS(H, nRST)
2 BEGIN
3   IF (nRST='0') THEN
4     nCS_Synchro <= '1';
5     RnW_Synchro <= '1';
6     Cnd_Synchro <= '1';
7     D07_Synchro <= (others => '0');
8   ELSIF (H'EVENT AND H='1') THEN
9     nCS_Synchro <= nCS;
10    RnW_Synchro <= RnW;
11    Cnd_Synchro <= Cnd;
12    D07_Synchro <= D07;
13  END IF;
14 END PROCESS InputProc_Synchro;

```

Listing 1 – Réseau Combinatoire d'entrée

Ce bloc VHDL gère la synchronisation des signaux provenant du microprocesseur. Le processus **InputProc_Synchro** lit les signaux d'entrée à chaque front montant de l'horloge H et les initialise lors de la mise à zéro nRST. Les signaux synchronisés (nCS_Synchro, RnW_Synchro, CnD_Synchro, D07_Synchro) sont ensuite utilisés par le reste de l'interface.

7.1.2 Réseau Combinatoire de Sortie

```

1 OutputProc_Comb : PROCESS(nCS_Synchro, CnD_Synchro, RnW_Synchro,
2   EtatCourant, OctetLu, EtatLu)
3 BEGIN
4   D07 <= (others => 'Z');
5   OctetLu_RD <= '0';
6   EtatLu_RST <= '0';
7   DecNbOctet <= '0';
8   CASE EtatCourant IS
9     WHEN Attente =>
10       IF (nCS_Synchro='0' AND CnD_Synchro='0' AND RnW_Synchro='1') THEN
11         OctetLu_RD <= '1';
12       END IF;
13     WHEN LectureData =>
14       D07 <= OctetLu;
15       IF (nCS_Synchro='1') THEN
16         DecNbOctet <= '1';
17       END IF;
18     WHEN LectureEtat =>
19       D07 <= EtatLu;
20       IF (nCS_Synchro='1') THEN
21         EtatLu_RST <= '1';
22       END IF;
23     WHEN EcritureFiltre =>
24   END CASE;
END PROCESS OutputProc_Comb;
```

Listing 2 – Réseau Combinatoire de Sortie

Le processus **OutputProc_Comb** contrôle la sortie des données et des états vers le microprocesseur. Il met à jour les signaux D07, OctetLu_RD, EtatLu_RST, DecNbOctet en fonction de l'état courant de la machine et des signaux synchronisés d'entrée. La logique combinatoire assure la correspondance entre les actions de lecture/écriture et l'état de la machine.

7.1.3 Réseau Combinatoire d'Entrée

```

1 ClockedProc : PROCESS(H, nRST)
2 BEGIN
3   IF (nRST='0') THEN
4     EtatCourant <= Attente;
5   ELSIF (H'EVENT AND H='1') THEN
6     EtatCourant <= EtatSuivant;
7   END IF;
8 END PROCESS ClockedProc;
9
10 NextStateProc : PROCESS(nCS_Synchro, CnD_Synchro, RnW_Synchro,
11   EtatCourant)
12 BEGIN
13   EtatSuivant <= EtatCourant;
14   CASE EtatCourant IS
```

```

14 WHEN Attente =>
15   IF (nCS_Synchro='0' AND CnD_Synchro='0' AND RnW_Synchro='1') THEN
16     EtatSuivant <= LectureData;
17   ELSIF (nCS_Synchro='0' AND CnD_Synchro='1' AND RnW_Synchro='1') THEN
18     EtatSuivant <= LectureEtat;
19   ELSIF (nCS_Synchro='0' AND CnD_Synchro='0' AND RnW_Synchro='0') THEN
20     EtatSuivant <= EcritureFiltre;
21   ELSE
22     EtatSuivant <= Attente;
23   END IF;
24 WHEN LectureData =>
25   IF (nCS_Synchro='1') THEN
26     EtatSuivant <= Attente;
27   ELSE
28     EtatSuivant <= LectureData;
29   END IF;
30 WHEN LectureEtat =>
31   IF (nCS_Synchro='1') THEN
32     EtatSuivant <= Attente;
33   ELSE
34     EtatSuivant <= LectureEtat;
35   END IF;
36 WHEN EcritureFiltre =>
37   IF (nCS_Synchro='1') THEN
38     EtatSuivant <= Attente;
39   ELSE
40     EtatSuivant <= EcritureFiltre;
41   END IF;
42 END CASE;
43 END PROCESS NextStateProc;

```

Listing 3 – Registres

Les processus `ClockedProc` et `NextStateProc` implémentent la machine séquentielle. `ClockedProc` met à jour l'état courant à chaque front montant de l'horloge et réinitialise l'état au démarrage. `NextStateProc` définit l'état suivant selon les conditions des signaux d'entrée et l'état courant, en suivant la logique de la machine de Mealy.

7.1.4 Réseau Synchronisé de Sortie

```

1 OutputProc_Synchro : PROCESS(H, nCLR)
2 BEGIN
3   IF (nCLR='0') THEN
4     SelAddr <= (others => '0');
5   ELSIF (H'EVENT AND H='1') THEN
6     CASE EtatCourant IS
7       WHEN EcritureFiltre =>
8         IF (nCS_Synchro='1') THEN
9           SelAddr <= D07_Synchro;
10        END IF;
11       WHEN OTHERS =>
12         END CASE;
13     END IF;
14 END PROCESS OutputProc_Synchro;
15
16 M_Received <= EtatLu(4);

```

Listing 4 – Réseau Synchronisé de Sortie

Le processus **OutputProc_Synchro** synchronise la sélection d'adresse **SelAdr** avec l'horloge **H**. Il est actif principalement pendant l'état **EcritureFiltre**, assurant que les données de l'entrée **D07_Synchro** sont correctement mémorisées. Le signal **M_Received** est également mis à jour pour refléter l'état du bit correspondant.

7.2 Interface de Réception LIN

L'interface de réception LIN a été étudiée sous la forme d'une machine séquentielle, composée d'une partie opérative et d'une partie commande. La partie opérative est développée sous la forme d'un schéma fonctionnel comprenant différents blocs tels que des multiplexeurs et des compteurs/décompteurs. La partie commande, quant à elle, a été modélisée sous la forme d'un automate, traduit en machine de Moore, puis implémenté en VHDL.

7.2.1 Partie opérative

La partie opérative, déjà définie dans la section Description de la solution architecturale, a été reprise sous forme de blocs fonctionnels. Il a simplement été nécessaire de reproduire le schéma global dans HDL Designer, afin d'assurer la cohérence entre la conception théorique et la modélisation pratique.

Le schéma correspondant est présenté ci-dessous :

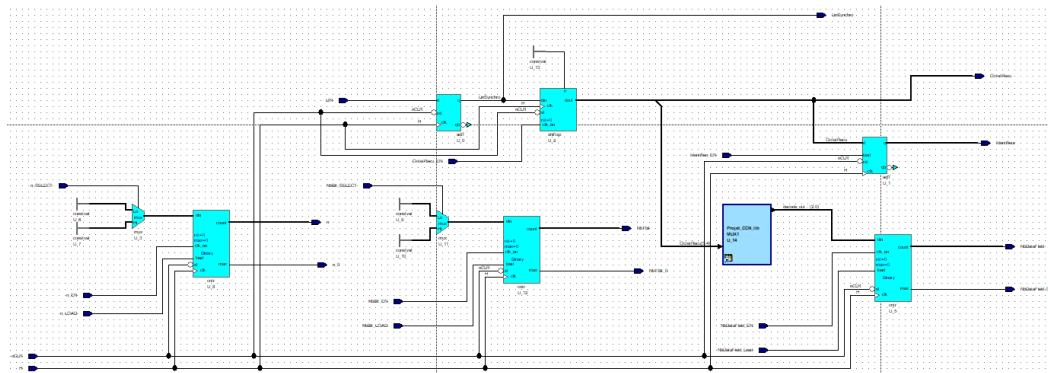


FIGURE 30 – Schéma partie opérative réception LIN

Le decodeur présents avant le décompteurs DataField permet de gérer la valeur chargé dans celui-ci. Voici la table de vérité associée :

OctetRecu	Decode_OUT
00	001
01	001
10	010
11	100

Voici aussi le tableau des entrées et sorties de la partie opérative :

Signal	Direction	Type
H	IN	std_logic
Identifien_EN	IN	std_logic
LIN	IN	std_logic
NbBit_EN	IN	std_logic
NbBit_LOAD	IN	std_logic
NbBit_SELECT	IN	std_logic
NbDataField_EN	IN	std_logic
NbDataField_Load	IN	std_logic
OctetReçu_EN	IN	std_logic
nCLR	IN	std_logic
n_EN	IN	std_logic
n_LOAD	IN	std_logic
n_SELECT	IN	std_logic
Identifieur	OUT	std_logic_vector(7 DOWNTO 0)
LinSynchro	OUT	std_logic
NbDataField	OUT	std_logic_vector(2 DOWNTO 0)
NbDataField_0	OUT	std_logic
NbTBit_0	OUT	std_logic
NbTbit	OUT	std_logic_vector(3 DOWNTO 0)
OctetReçu	OUT	std_logic_vector(7 DOWNTO 0)
n	OUT	std_logic_vector(11 DOWNTO 0)
n_0	OUT	std_logic

TABLE 2 – Signaux du module (partie opérative de réception LIN)

7.2.2 Partie Commande

La partie commande consiste principalement à traduire l'automate présenté en figure ?? en code VHDL. Cette étape reste relativement simple, car elle repose sur la création de trois processus principaux :

1. **Le réseau combinatoire d'entrée**, chargé de déterminer les *états futurs* de l'automate en fonction des *états présents* et des *signaux d'entrée*.
2. **Le réseau de registres**, synchronisé sur l'horloge, permettant de *mémoriser les états* et d'assurer la transition entre les *états présents* et les *états futurs*.
3. **Le réseau combinatoire de sortie**, qui met à jour les *signaux de sortie* en fonction de l'état courant de l'automate.

Voici le code suivant qui permet de traduire l'automate :

```

1 -- Etat de la machine
2 type CFM is (
3     R_BRK_0 , R_BRK_1 ,                               -- Reception Break
4     SYN_A , SYN_RST , SYN_RD , SYN_RSP ,           -- Synchro
5     IDN_A , IDN_RST , IDN_RD , IDN_RSP ,          -- Identifier
6     DAT_A , DAT_RST , DAT_RD , DAT_RSP ,           -- Datafield
7     CHK_A , CHK_RST , CHK_RD , CHK_RSP ,           -- Checksum
8     REPOS                                         -- Repos
9 );

```

Listing 5 – Déclaration des états

Dans une première étape, nous déclarons les différents états de l'automate sous la forme d'un type énuméré nommé CFM. Chaque état correspond à une étape spécifique du processus de réception LIN, facilitant ainsi la gestion des transitions et des actions associées à chaque état.

```

1  -- Register
2  CFM_Register : process(H, nCLR)
3  begin
4      if nCLR = '0' then
5          P_CFM <= REPOS;
6      elsif rising_edge(H) then
7          P_CFM <= N_CFM;
8      end if;
9  end process CFM_Register;

```

Listing 6 – Registres Reception Trame

Dans ce code nous retrouvons la clock qui permet de synchroniser les états de l'automate avec le signal d'horloge H. Le changement d'état se fait au front montant de l'horloge. Le reset asynchrone nCLR permet de remettre l'automate dans son état initial

```

1  -- Reseau Combinatoire d'Entree
2  CFM_RCE : process(P_CFM, H, Identifier, LinSynchro, NbTbit_0,
3                      NbDataField_0, SelAddr, nCLR, n_0)
4  begin
5      -- Next State <= Present State
6      N_CFM <= P_CFM;
7
8      case P_CFM is
9          when REPOS => -- Attente
10              if LinSynchro = '0' then
11                  N_CFM <= R_BRK_0;
12              end if;
13          when R_BRK_0 => -- Synchro Break 0
14              if NbTbit_0 = '1' AND LinSynchro = '1' then
15                  N_CFM <= R_BRK_1;
16              elsif NbTbit_0 = '0' AND LinSynchro = '1' then
17                  N_CFM <= REPOS;
18              elsif n_0 = '0' AND LinSynchro = '0' AND NbTbit_0 = '0' then
19                  N_CFM <= R_BRK_0;
20              elsif NbTbit_0 = '0' AND LinSynchro = '0' AND n_0 = '1' then
21                  N_CFM <= R_BRK_0;
22              end if;
23          when R_BRK_1 => -- Synchro Break 1
24              if n_0 = '1' AND LinSynchro = '1' then
25                  N_CFM <= SYN_A;
26              elsif n_0 = '0' AND LinSynchro = '0' then
27                  N_CFM <= REPOS;
28              elsif n_0 = '0' then
29                  N_CFM <= R_BRK_1;
30              end if;
31          when SYN_A => -- Attente bit Start Synchronisation
32              if LinSynchro = '0' then
33                  N_CFM <= SYN_RST;
34              end if;
35          when SYN_RST => -- Reception Start Synchronisation
36              if n_0 = '0' AND LinSynchro = '0' then

```

```

36         N_CFM <= SYN_RD;
37     elsif n_0 = '0' AND LinSynchro = '1' then
38         N_CFM <= REPOS;
39     elsif n_0 = '0' then
40         N_CFM <= SYN_RST;
41     end if;
42 when SYN_RD => -- Reception Data Synchronisation
43     if NbTbit_0 = '1' then
44         N_CFM <= SYN_RSP;
45     elsif n_0 = '1' AND NbTbit_0 = '0' then
46         N_CFM <= SYN_RD;
47     elsif n_0 = '0' then
48         N_CFM <= SYN_RD;
49     end if;
50 when SYN_RSP => -- Reception bit Stop Synchronisation
51     if n_0 = '1' AND LinSynchro = '1' then
52         N_CFM <= IDN_A;
53     elsif n_0 = '0' AND NbTbit_0 = '0' then
54         N_CFM <= REPOS;
55     elsif n_0 = '0' then
56         N_CFM <= SYN_RSP;
57     end if;
58 when IDN_A => -- Attente bit Start Identifier
59     if LinSynchro = '0' then
60         N_CFM <= IDN_RST;
61     end if;
62 when IDN_RST => -- Reception bit Start Identifier
63     if n_0 = '1' AND LinSynchro = '0' then
64         N_CFM <= IDN_RD;
65     elsif n_0 = '1' AND LinSynchro = '1' then
66         N_CFM <= REPOS;
67     elsif n_0 = '0' AND LinSynchro = '0' then
68         N_CFM <= IDN_RST;
69     end if;
70 when IDN_RD => -- Reception Data Identifier
71     if NbTbit_0 = '1' AND n_0 = '1' then
72         N_CFM <= IDN_RSP;
73     elsif n_0 = '0' then
74         N_CFM <= IDN_RD;
75     elsif n_0 = '1' AND NbTbit_0 = '0' then
76         N_CFM <= IDN_RD;
77     end if;
78 when IDN_RSP => -- Reception bit Stop Identifier
79     if n_0 = '1' AND LinSynchro = '1' then
80         N_CFM <= DAT_A;
81     elsif n_0 = '1' AND LinSynchro = '0' then
82         N_CFM <= REPOS;
83     elsif n_0 = '0' AND LinSynchro = '1' then
84         N_CFM <= IDN_RSP;
85     end if;
86 when DAT_A => -- Attente bit Start Datafield
87     if LinSynchro = '0' then
88         N_CFM <= DAT_RST;
89     end if;
90 when DAT_RST => -- Reception bit Start Datafield
91     if n_0 = '1' AND LinSynchro = '0' then
92         N_CFM <= DAT_RD;

```

```

93         elsif n_0 = '1' AND LinSynchro = '1' then
94             N_CFM <= REPOS;
95         elsif n_0 = '0' AND LinSynchro = '0' then
96             N_CFM <= DAT_RST;
97         end if;
98     when DAT_RD => -- Reception bit Data Datafield
99         if NbTbit_0 = '1' AND LinSynchro = '1' then
100            N_CFM <= DAT_RSP;
101        elsif n_0 = '1' AND NbTbit_0 = '0' then
102            N_CFM <= DAT_RD;
103        elsif n_0 = '0' then
104            N_CFM <= DAT_RD;
105        end if;
106    when DAT_RSP => -- Reception bit Stop Datafield
107        if n_0 = '0' AND NbDataField_0 = '1' AND LinSynchro = '1'
108            then
109                N_CFM <= CHK_A;
110            elsif NbDataField_0 = '0' AND n_0 = '1' AND LinSynchro = '1'
111            then
112                N_CFM <= DAT_A;
113            elsif n_0 = '1' AND LinSynchro = '0' then
114                N_CFM <= REPOS;
115            elsif n_0 = '0' AND LinSynchro = '1' then
116                N_CFM <= DAT_RSP;
117            end if;
118        when CHK_A => -- Attente bit Start Checksum
119            if LinSynchro = '0' then
120                N_CFM <= CHK_RST;
121            end if;
122        when CHK_RST => -- Reception bit Start Checksum
123            if n_0 = '1' AND LinSynchro = '0' then
124                N_CFM <= CHK_RD;
125            elsif n_0 = '1' AND LinSynchro = '1' then
126                N_CFM <= REPOS;
127            elsif n_0 = '0' AND LinSynchro = '0' then
128                N_CFM <= CHK_RST;
129            end if;
130        when CHK_RD => -- Reception bit Data Checksum
131            if NbTbit_0 = '1' AND LinSynchro = '1' then
132                N_CFM <= CHK_RSP;
133            elsif n_0 = '1' AND NbTbit_0 = '0' then
134                N_CFM <= CHK_RD;
135            elsif n_0 = '0' then
136                N_CFM <= CHK_RD;
137            end if;
138        when CHK_RSP => -- Reception bit Stop Checksum
139            if n_0 = '1' AND LinSynchro = '1' AND Identifier = SelAddr
140                then
141                    N_CFM <= REPOS; -- Fin de Trame Complete
142            elsif n_0 = '1' AND LinSynchro = '1' AND Identifier /= SelAddr
143                then
144                    N_CFM <= REPOS;
145            elsif n_0 = '0' AND LinSynchro = '0' then
146                N_CFM <= REPOS;
147            elsif n_0 = '0' AND LinSynchro = '1' then
148                N_CFM <= CHK_RSP;
149            end if;

```

```

146     end case;
147
148 end process CFM_RCE;

```

Listing 7 – Réseau Combinatoire d’Entrée Reception Trame

Ce code VHDL traduit l’automate de réception LIN en utilisant un processus combinatoire nommé CFM_RCE. Il détermine l’état suivant (N_CFM) en fonction de l’état actuel (P_CFM) et des signaux d’entrée tels que LinSynchro, NbTbit_0, Identifier, etc. Chaque état de l’automate est représenté par une branche dans la structure CASE, avec des conditions spécifiques pour les transitions entre états.

```

1   -- Reseau Combinatoire de Sortie
2 CFM_RES : process(P_CFM, H, Identifier, LinSynchro, NbTbit_0,
3                     NbDataField_0, SelAddr, nCLR, n_0 )
4 begin
5   -- Valeurs par defaut
6   Error_Start      <= '0';
7   Error_Stop       <= '0';
8   Error_Synchro    <= '0';
9   Identifieur_en   <= '0';
10  IncNbOctet      <= '0';
11  MessageReceiveSet <= '0';
12  NbDataField_EN   <= '0';
13  NbDataField_load <= '0';
14  NbOcyeyRechu_RST <= '0';
15  OctetRechu_RST   <= '0';
16  OctetRechu_WR    <= '0';
17  OctetRechu_en    <= '0';
18  n_Tbit_Load      <= '0';
19  n_Tbit_en         <= '0';
20  n_Tbit_select    <= '0';
21  n_en              <= '0';
22  n_load             <= '0';
23  n_select            <= '0';

24 case P_CFM is
25   when REPOS => -- Attente
26     if LinSynchro = '0' then
27       n_select      <= '0';
28       n_load        <= '1';
29       n_en          <= '1';
30       n_Tbit_select <= '0';
31       n_Tbit_en     <= '1';
32       n_Tbit_Load   <= '1';
33     end if;

34   when R_BRK_0 => -- Synchro Break 0
35     if NbTbit_0 = '1' and LinSynchro = '1' then
36       n_select <= '0';
37       n_en     <= '1';
38       n_load   <= '1';
39     elsif NbTbit_0 = '0' and LinSynchro = '1' then
40       Error_Synchro <= '1';
41     elsif NbTbit_0 = '0' and LinSynchro = '0' and n_0 = '1' then
42       n_Tbit_en <= '1';
43       n_select  <= '0';

```

```

45         n_load      <= '1';
46         n_en       <= '1';
47     elsif n_0 = '0' AND LinSynchro = '0' AND NbTbit_0 = '0' then
48         n_en <= '1';
49     end if;
50
51 when R_BRK_1 => -- Synchro Break 1
52     if n_0 = '1' and LinSynchro = '0' then
53         -- Nothing
54     elsif n_0 = '0' and LinSynchro = '0' then
55         Error_Synchro <= '1';
56     elsif n_0 = '0' then
57         n_en <= '1';
58     end if;
59
60 when SYN_A => -- Attente bit Start Synchronisation
61     if LinSynchro = '0' then
62         n_select <= '0';
63         n_load   <= '1';
64         n_en     <= '1';
65     end if;
66
67 when SYN_RST => -- Reception Start Synchronisation
68     if n_0 = '0' AND LinSynchro = '0' then
69         n_select      <= '1'; -- passage de fin de front a milieu
70             bit
71         n_load        <= '1';
72         n_en         <= '1';
73         n_Tbit_select <= '1';
74         n_Tbit_en    <= '1';
75         n_Tbit_Load  <= '1';
76     elsif n_0 = '0' AND LinSynchro = '1' then
77         Error_Start <= '1';
78     elsif n_0 = '0' then
79         n_en         <= '1';
80     end if;
81
82 when SYN_RD => -- Reception Data Synchronisation
83     if NbTbit_0 = '1' then
84         n_select <= '0';
85         n_load   <= '1';
86         n_en     <= '1';
87     elsif n_0 = '1' AND NbTbit_0 = '0' then
88         n_select <= '0';
89         n_load   <= '1';
90         n_en     <= '1';
91         n_Tbit_en <= '1';
92         OctetRecu_en <= '1';
93     elsif n_0 = '0' then
94         n_en         <= '1';
95     end if;
96
97 when SYN_RSP => -- Reception bit Stop Synchronisation
98     if n_0 = '1' AND LinSynchro = '1' then
99         OctetRecu_WR <= '1';
100        n_select <= '1';
100        n_load   <= '1';

```

```

101         n_en          <= '1';
102     elsif n_0 = '0' AND NbTbit_0 = '0' then
103         Error_Stop      <= '1';
104     elsif n_0 = '0' then
105         n_en          <= '1';
106     end if;
107
108 when IDN_A => -- Attente bit Start Identifier
109     if LinSynchro = '0' then
110         n_select        <= '0';
111         n_load          <= '1';
112         n_en           <= '1';
113     end if;
114
115 when IDN_RST => -- Reception bit Start Identifier
116     if n_0 = '1' AND LinSynchro = '0' then
117         n_select        <= '0';
118         n_load          <= '1';
119         n_en           <= '1';
120         n_Tbit_select   <= '1';
121         n_Tbit_Load    <= '1';
122         n_Tbit_en       <= '1';
123     elsif n_0 = '1' AND LinSynchro = '1' then
124         Error_Start     <= '1';
125     elsif n_0 = '0' AND LinSynchro = '0' then
126         n_en           <= '1';
127     end if;
128
129 when IDN_RD => -- Reception Data Identifier
130     if NbTbit_0 = '1' AND n_0 = '1' then
131         n_select        <= '0';
132         n_load          <= '1';
133         n_en           <= '1';
134         OctetRechu_en  <= '1';
135     elsif n_0 = '0' then
136         n_en           <= '1';
137     elsif n_0 = '1' AND NbTbit_0 = '0' then
138         n_select        <= '0';
139         n_load          <= '1';
140         n_en           <= '1';
141         n_Tbit_en       <= '1';
142         OctetRechu_en  <= '1';
143     end if;
144
145 when IDN_RSP => -- Reception bit Stop Identifier
146     if n_0 = '1' AND LinSynchro = '1' then
147         OctetRechu_en  <= '1';
148         NbDataField_EN <= '1';
149         Identifieur_en <= '1';
150         NbDataField_load <= '1';
151     elsif n_0 = '0' AND LinSynchro = '0' then
152         Error_Stop      <= '1';
153     elsif n_0 = '0' AND LinSynchro = '1' then
154         n_select        <= '0';
155         n_load          <= '0';
156         n_en           <= '1';
157     end if;

```

```

158
159     when DAT_A => -- Attente bit Start Datafield
160         if LinSynchro = '0' then
161             n_select      <= '0';
162             n_load       <= '1';
163             n_en        <= '1';
164         end if;
165
166     when DAT_RST => -- Reception bit Start Datafield
167         if n_0 = '1' AND LinSynchro = '0' then
168             n_select      <= '0';
169             n_load       <= '1';
170             n_en        <= '1';
171             n_Tbit_select <= '1';
172             n_Tbit_Load   <= '1';
173             n_Tbit_en    <= '1';
174         elsif n_0 = '1' AND LinSynchro = '1' then
175             Error_Start  <= '1';
176         elsif n_0 = '0' AND LinSynchro = '0' then
177             n_en        <= '1';
178         end if;
179
180     when DAT_RD => -- Reception bit Data Datafield
181         if NbTbit_0 = '1' AND LinSynchro = '1' then
182             n_select      <= '0';
183             n_load       <= '1';
184             n_en        <= '1';
185             OctetRechu_en <= '1';
186         elsif n_0 = '0' then
187             n_en        <= '1';
188         elsif n_0 = '1' AND NbTbit_0 = '0' then
189             n_select      <= '0';
190             n_load       <= '1';
191             n_en        <= '1';
192             n_Tbit_en   <= '1';
193             OctetRechu_en <= '1';
194         end if;
195
196     when DAT_RSP => -- Reception bit Stop Datafield
197         if n_0 = '0' AND NbDataField_0 = '1' AND LinSynchro = '1' then
198             OctetRechu_WR  <= '1';
199             n_select      <= '0';
200             n_load       <= '1';
201             n_en        <= '1';
202             IncNbOctet   <= '1';
203         elsif NbDataField_0 = '0' AND n_0 = '1' AND LinSynchro = '1'
204             then
205             OctetRechu_WR  <= '1';
206             NbDataField_EN <= '1';
207             n_select      <= '0';
208             n_load       <= '1';
209             n_en        <= '1';
210             IncNbOctet   <= '1';
211         elsif n_0 = '1' AND LinSynchro = '0' then
212             Error_Stop   <= '1';
213         elsif n_0 = '0' AND LinSynchro = '1' then
214             n_en        <= '1';

```

```

214         end if;
215
216     when CHK_A => -- Attente bit Start Checksum
217         if LinSynchro = '0' then
218             n_select      <= '0';
219             n_load       <= '1';
220             n_en        <= '1';
221         end if;
222
223     when CHK_RST => -- Reception bit Start Checksum
224         if n_0 = '1' AND LinSynchro = '0' then
225             n_select      <= '0';
226             n_load       <= '1';
227             n_en        <= '1';
228             n_Tbit_select <= '1';
229         elsif n_0 = '1' AND LinSynchro = '1' then
230             Error_Start   <= '1';
231         elsif n_0 = '0' AND LinSynchro = '0' then
232             n_en        <= '1';
233         end if;
234
235     when CHK_RD => -- Reception bit Data Checksum
236         if NbTbit_0 = '1' AND LinSynchro = '1' then
237             n_select      <= '0';
238             n_load       <= '1';
239             n_en        <= '1';
240         elsif n_0 = '1' AND NbTbit_0 = '0' then
241             n_en        <= '1';
242         elsif n_0 = '0' then
243             n_Tbit_en    <= '1';
244             n_select      <= '0';
245             n_load       <= '1';
246             n_en        <= '1';
247             OctetRecu_en <= '1';
248             Identifieur_en <= '1';
249             NbDataField_load <= '1';
250             NbDataField_EN   <= '1';
251         end if;
252
253     when CHK_RSP => -- Reception bit Stop Checksum
254         if n_0 = '0' AND LinSynchro = '1' AND Identifier = SelAddr then
255             MessageReceiveSet <= '1';
256             NbOcyeyRecu_RST  <= '1';
257         elsif n_0 = '0' AND LinSynchro = '1' AND Identifier /= SelAddr
258             then
259                 MessageReceiveSet <= '1';
260                 NbOcyeyRecu_RST  <= '1';
261             elsif n_0 = '0' AND LinSynchro = '0' then
262                 Error_Stop      <= '1';
263             elsif n_0 = '0' then
264                 n_en        <= '1';
265             end if;
266
267     when others =>
268         -- Sec
269         null;
270     end case;

```

270 | **end process CFM_RES ;**

Listing 8 – Réseau Combinatoire de Sortie Reception Trame

Ce code VHDL implémente le réseau combinatoire de sortie (CFM_RES) pour l'automate de réception LIN. Il met à jour les signaux de sortie tels que `Error_Start`, `Error_Stop`, `OctetReçu_WR`, etc., en fonction de l'état actuel (`P_CFM`) et des signaux d'entrée. Chaque état de l'automate est géré dans une structure `CASE`, avec des conditions spécifiques pour définir les actions à entreprendre dans chaque état. Des valeurs par défaut sont également définies au début du processus pour éviter des comportements indésirables.

Voici un tableau récapitulant les entrées et les sorties du bloc Repetition LIN partie commande :

Signal	Direction	Type
H	IN	std_logic
Identifier	IN	std_logic_vector(7 DOWNTO 0)
LinSynchro	IN	std_logic
NbTbit_0	IN	std_logic
NbDataField_0	IN	std_logic
SelAdr	IN	std_logic_vector(7 DOWNTO 0)
nCLR	IN	std_logic
n_0	IN	std_logic
Error_Start	OUT	std_logic
Error_Stop	OUT	std_logic
Error_Synchro	OUT	std_logic
Identifieur_en	OUT	std_logic
IncNbOctet	OUT	std_logic
MessageReceiveSet	OUT	std_logic
NbDataField_EN	OUT	std_logic
NbDataField_load	OUT	std_logic
NbOcyeyReçu_RST	OUT	std_logic
OctetReçu_RST	OUT	std_logic
OctetReçu_WR	OUT	std_logic
OctetReçu_en	OUT	std_logic
n_Tbit_Load	OUT	std_logic
n_Tbit_en	OUT	std_logic
n_Tbit_select	OUT	std_logic
n_en	OUT	std_logic
n_load	OUT	std_logic
n_select	OUT	std_logic

TABLE 3 – Signaux du module (partie commande de réception LIN)

Une fois ces processus implémentés, il nous reste qu'à les assemblés pour ne formé qu'une machine Complete.

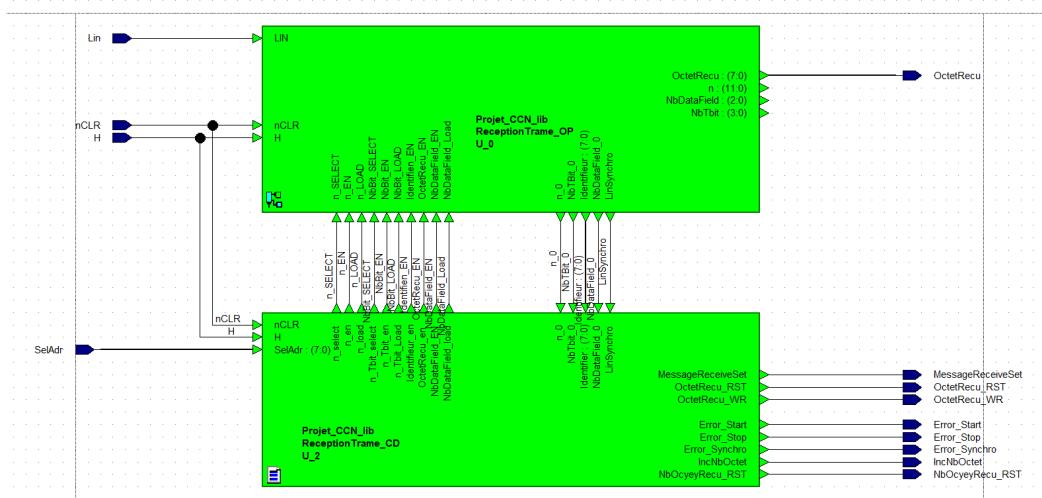


FIGURE 31 – Machine Séquentielle réception LIN

Le schéma complet de l'interface de réception LIN est présenté ci-dessus, illustrant l'intégration des différentes parties opératives et commandes (Réprésentant la machine séquentielle).

Voici le tableau des entrées et sorties du bloc Repetion LIN complet :

Signal	Direction	Type
H	IN	std_logic
Lin	IN	std_logic
SelAddr	IN	std_logic_vector(7 DOWNTO 0)
nCLR	IN	std_logic
Error_Start	OUT	std_logic
Error_Stop	OUT	std_logic
Error_Synchro	OUT	std_logic
IncNbOctet	OUT	std_logic
MessageReceiveSet	OUT	std_logic
NbOcyeyRechu_RST	OUT	std_logic
OctetRechu	OUT	std_logic_vector(7 DOWNTO 0)
OctetRechu_RST	OUT	std_logic
OctetRechu_WR	OUT	std_logic

TABLE 4 – Signaux du module (partie réception LIN)

7.3 FIFO

La mémoire FIFO (First In, First Out) a été conçue pour stocker temporairement les données reçues via l'interface LIN avant leur traitement ultérieur. Elle est structurée autour d'un composant clef, la FIFO. Cette section a été réalisée sur un schéma blocs, en reprenant les concepts vus en cours.

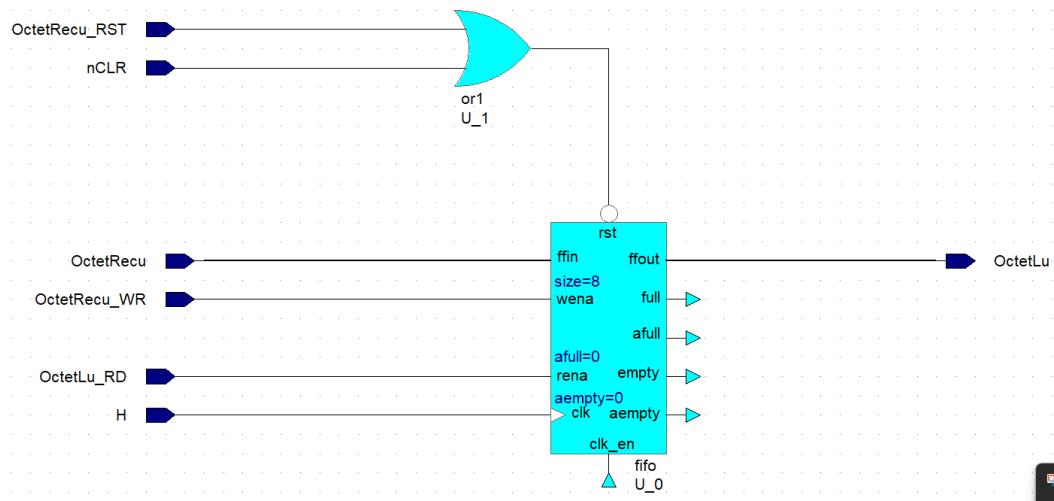


FIGURE 32 – Schéma bloc FIFO

Voici le tableau des entrées et sorties du bloc FIFO :

Signal	Direction	Type
H	IN	std_logic
OctetLu_RD	IN	std_logic
OctetRecu	IN	std_logic_vector(7 DOWNTO 0)
OctetRecu_RST	IN	std_logic
OctetRecu_WR	IN	std_logic
nCLR	IN	std_logic
OctetLu	OUT	std_logic_vector(7 DOWNTO 0)

TABLE 5 – Signaux du module (lecture des octets LIN)

7.4 Etat Interne

Etat interne est un composant VHDL qui gère les états internes du système de réception LIN. Le système a été conçu pour suivre et contrôler les différentes étapes du processus de réception des trames LIN. L'objectifs étant d'additionner des fonctionnalités de contrôle et de gestion des états internes pour ne former qu'une seul trame.

Ce système à lui aussi été conçu sous la forme d'un schéma bloc, reprenant les concepts vus en cours. Avec des bascules D, des compteurs et l'ajout de OR pour gérer les RESET.

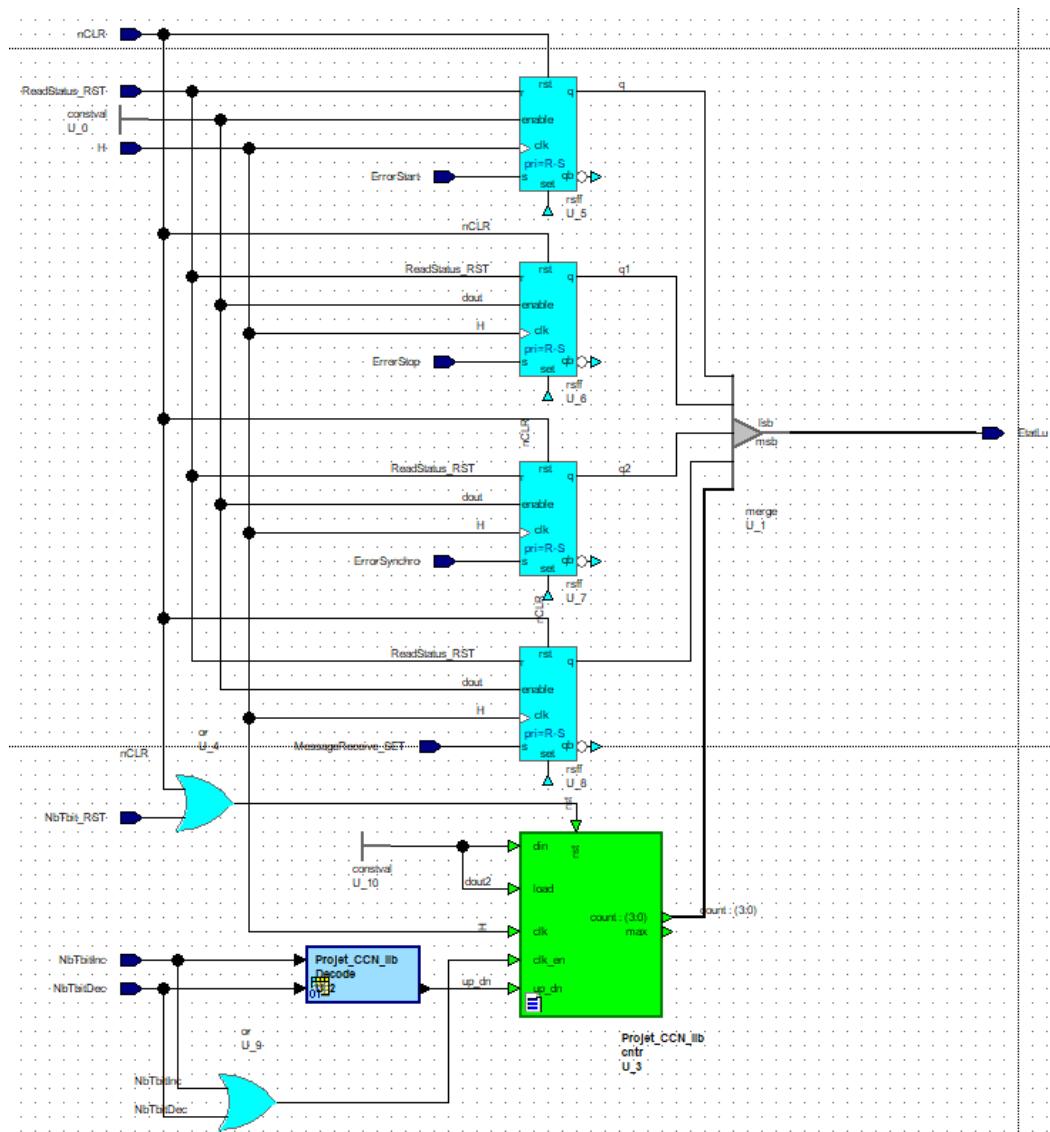


FIGURE 33 – Schéma bloc EtatInterne

Le module de Decode présent avant le compteur/decompteur permet de savoir si il doit compter ou décompter. Ce signal sera généré en fonction des états des signaux de la partie commande.

NbTbitInc	NbTbitDec	UP/DN
0	0	X
0	1	1
0	0	0
1	1	X

Voici le tableau des entrées et sorties du bloc Etat Interne :

Signal	Direction	Type
ErrorStart	IN	std_logic
ErrorStop	IN	std_logic
ErrorSynchro	IN	std_logic
H	IN	std_logic
MessageReceive_SET	IN	std_logic
NbTbitDec	IN	std_logic
NbTbitInc	IN	std_logic
NbTbit_RST	IN	std_logic
ReadStatus_RST	IN	std_logic
nCLR	IN	std_logic
EtatLu	OUT	std_logic_vector(7 DOWNTO 0)

TABLE 6 – Signaux du module (lecture de l'état LIN)

7.5 Réception LIN Complete

Après avoir développé les différentes parties du système de réception LIN, nous procédons à l'intégration de ces composants pour former une unité cohérente et fonctionnelle.

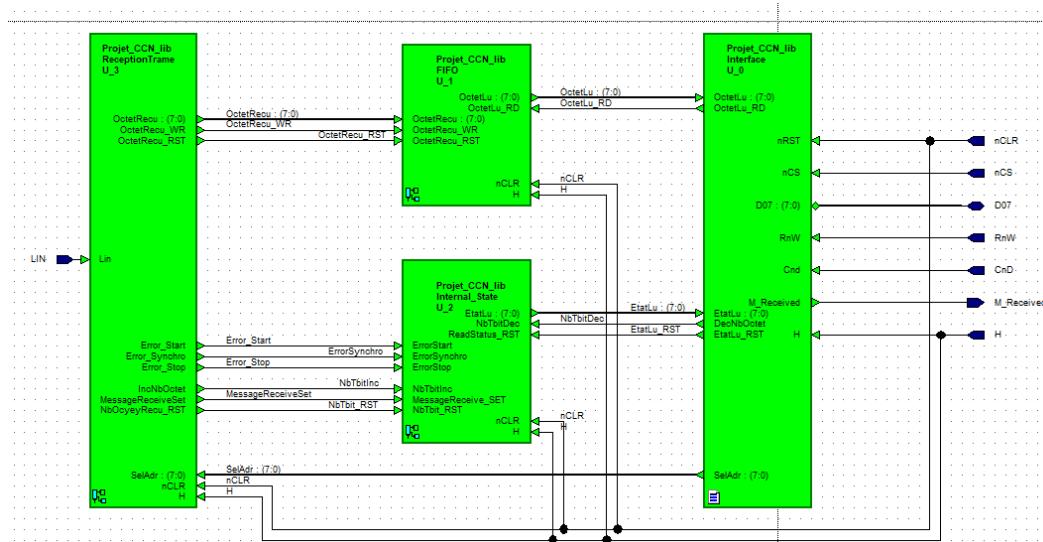


FIGURE 34 – Schéma bloc Réception LIN Complete

On retrouve les différentes parties vues précédemment, à savoir l'interface de réception LIN, la mémoire FIFO et le module d'état interne.

8 Simulation des fonctions

8.1 Interface Microprocesseur

Dans cette section, nous présentons la simulation du bloc *Interface Microprocesseur* et l'analyse des chronogrammes obtenus. La simulation a été réalisée à l'aide d'un *testbench*, implémenté sous la forme d'un bloc nommé **EnvTest_InterfaceMicroprocesseur**, connecté au composant **InterfaceMicroprocesseur**. L'objectif est de vérifier la conformité du fonctionnement par rapport à l'automate décrit dans la section *Architecture*.

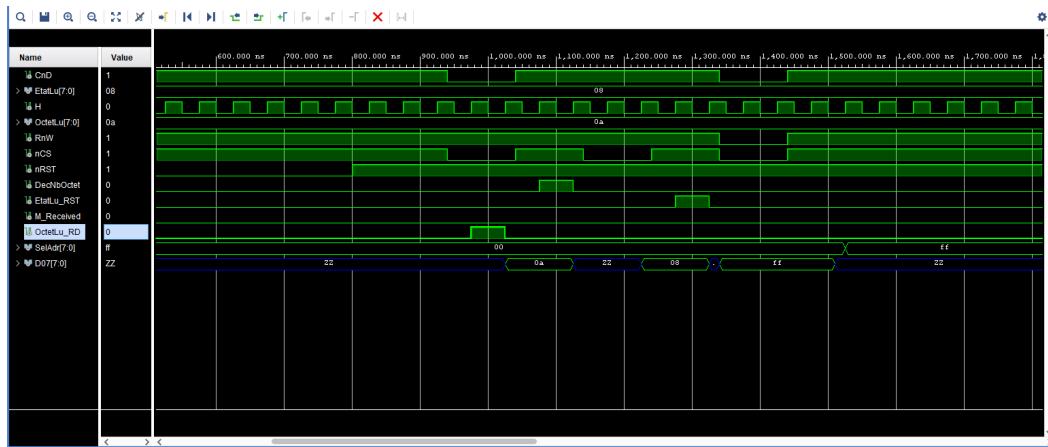


FIGURE 35 – Chronogramme de simulation de l'Interface Microprocesseur

Le testbench (fourni en annexe) a pour rôle de reproduire l'environnement dans lequel le composant est amené à fonctionner. Il émule le comportement d'un microprocesseur en générant automatiquement les stimuli nécessaires à la validation du bloc testé.

8.1.1 Déclarations et signaux

Le testbench commence par la déclaration des librairies IEEE, nécessaires à la manipulation des types logiques et des vecteurs binaires. Les principaux signaux utilisés sont :

- CnD, RnW, nCS, nRST, H : lignes de contrôle classiques d'une interface microprocesseur (commande/données, lecture/écriture, sélection du composant, reset, horloge),
- OctetLu, EtatLu, SelAdr, D07 : bus de données et d'adresses sur 8 bits,
- DecNbOctet, EtatLu_RST, M_Received, OctetLu_RD : signaux internes utilisés pour la communication avec le composant testé.

8.1.2 Instanciation du composant testé

Le composant **InterfaceMicroprocesseur** est instancié dans l'architecture de simulation. Il est relié à l'ensemble des signaux déclarés, permettant ainsi l'observation de son comportement face aux stimuli générés.

8.1.3 Environnement de test

Le composant **EnvTest_InterfaceMicroprocesseur** simule le rôle du microprocesseur en générant automatiquement les signaux nécessaires :

- génération de l'horloge (H),
- gestion du reset global (nRST),

- activation des commandes de lecture/écriture (RnW, CnD, nCS),
 - pilotage du bus de données (D07).
- Cet environnement est donné par plusieurs paramètres génériques :
- **CLOCK_PERIOD** : période d'horloge (50 ns),
 - **RESET_OFFSET** et **RESET_DURATION** : moment et durée du reset (500 ns et 300 ns),
 - **ACCESS_TIME** et **HOLD_TIME** : contraintes temporelles d'accès et de maintien (40 ns et 70 ns).

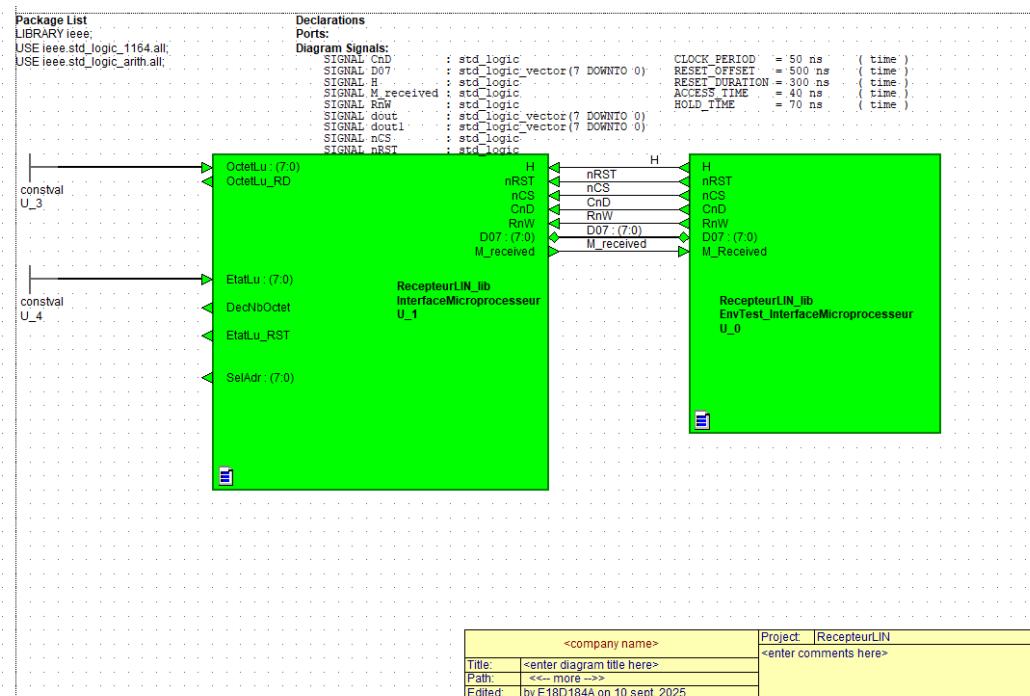


FIGURE 36 – Block Diagramme de test de l'Interface Microprocesseur

8.1.4 Stimuli supplémentaires

Un processus spécifique (**StimProc**) complète la génération des signaux. Après la fin du reset, il impose des valeurs constantes sur certaines lignes :

- **OctetLu** \leftarrow 10 (codé sur 8 bits),
- **EstatLu** \leftarrow 8 (codé sur 8 bits).

Ces valeurs permettent de vérifier la gestion correcte des données reçues par l'interface. La simulation est ensuite maintenue en attente infinie.

8.1.5 Analyse du chronogramme de simulation

L'analyse du chronogramme met en évidence le comportement attendu du composant :

- lorsque les signaux de contrôle actifs à l'état bas (nRST, nCS) sont à l'état haut, aucune action n'est effectuée,
- lorsque ces signaux sont activés (passage à l'état bas), le composant réagit conformément à l'automate interne,
- les signaux RnW et CnD permettent de sélectionner respectivement les opérations de lecture/écriture et le type d'accès (commande ou données),
- les valeurs imposées sur OctetLu et EtatLu sont correctement lues via le bus de données D07.

Ce chronogramme confirme ainsi le bon fonctionnement du composant **InterfaceMicroprocesseur** : après la levée du reset, l'environnement de test génère des cycles de lecture et d'écriture auxquels le composant répond correctement, en échangeant les données prévues et en activant les signaux de contrôle appropriés.

8.2 Interface Reception LIN

Cette partie présente la simulation du bloc final Interface Réception LIN. La simulation a été réalisée à l'aide d'un environnement pour le Réception LIN qui génère une trame LIN complète sur l'entrée Lin. L'intégration des quatre sous-modules et vérifier que la réponse des modules est bien validée par le cahier des charges.

8.2.1 Déclarations et signaux

La simulation est effectuée sur l'entité de plus haut niveau le Récepteur LIN. L'environnement de test permet de simuler l'émetteur LIN et le microprocesseur. Il pilote donc le signal d'entrée LIN, l'horloge H, le reset nCLR, nCS, RnW et CnD. Ainsi que le signal d'échange des données D07 et le signal de sortie M Received.

8.2.2 Instanciation du composant testé

Le composant Recepteur LIN instancié. Il connecté en interne les quatres modules :

- Le Recepteur de trame
- La FIFO
- La mémoire de l'état interne
- L'interface microprocesseur

8.2.3 Environnement de test

La simulation se déroule avec une trame complète LIN envoyé contenant un Sync Field (0x55), un Identifiant, et une série d'octets de données. Et une interaction microprocesseur effectuant les cycles de lecture de données, d'états et d'écriture d'adresse.

8.2.4 Stimuli supplémentaires

Échantillonnage au milieu des bits

La bonne lecture de chaque bit doit être échantillonné à la moitié de la période d'un bit émis par la trame LIN

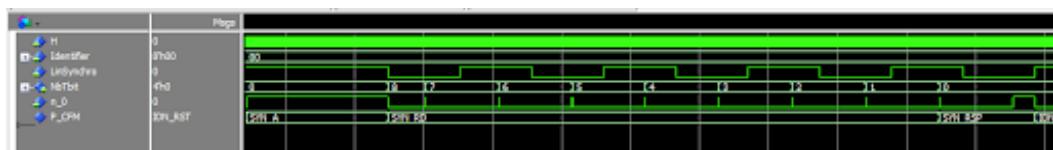


FIGURE 37 – Simulation de l'échantillonnage au milieu des bits

L'analyse du chronogramme montre que chaque bit reçu sur la ligne LIN est échantillonné précisément au milieu de sa période. Cela est confirmé par l'alignement des fronts montants du signal NbTbit avec le milieu de chaque bit sur le signal LinSynchro.

Échantillonnage au milieu des bits

La vérification se fait sur l'envoie du premier octet d'une trame, l'octet Sync Field envoyé sur la trame LIN, soit 01010101 en binaire. Le stockage se fait en stockant les bits un à un sur la position LSB du registre, et décalant le contenu existant la donnée à chaque nouvelle valeur reçue dans le registre à décalage.

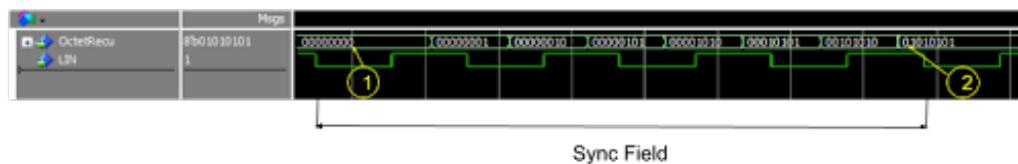


FIGURE 38 – Simulation du stockage des bits dans le registre à décalage

La position du MSB est de niveau bas en n°1, après réception du 7 autres bits et avoir décalé le 0 de 7 position, il est bien en MSB dans le signal à 8 bit de l'octet reçu en n°3. La gestion de l'ordre de bits est donc bien corriger dans le récepteur LIN.

Le circuit a converti avec succès le flux série en l'octet parallèle. Ce résultat valide que le registre à décalage opère correctement et assure la gestion correcte de l'ordre des bits dans le récepteur LIN.

Réception correcte des octets d'une trame

Le circuit doit bien enregistrer et séparer dans la FIFO chaque champ de la trame LIN (Sync field, Data, Checksum). L'écriture des champs est signalée par l'envoie du signal OctetReçu_WR au niveau haut. L'analyse se fait avec la vérification du champ Sync Field.

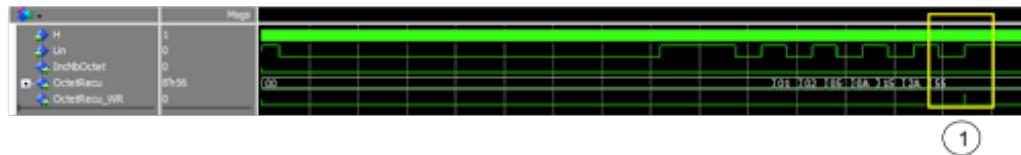


FIGURE 39 – Simulation de la réception correcte des octets d'une trame

Dans l'encadré n°1, à la fin du champ Sync field le signal est stable et à la bonne valeur. Le signal `OctetReceu_WR` génère une impulsion à la fin de la réception de l'octet complet. Cela permet de transférer l'octet à la FIFO et de le mémoriser.

L'analyse confirme donc la réception correcte et séquentielle de l'intégralité des octets de la trame par le circuit. L'analyse se fait avec la vérification du champ Sync Field.

Comptage/Décomptage du nombre d'octets

Cette partie suit la progression de la réception des octets de la trame LIN. Lors de la réception de chaque octet on décompte `NbOctetsReçus`, jusqu'à arriver au dernier octet de donnée de la trame LIN.

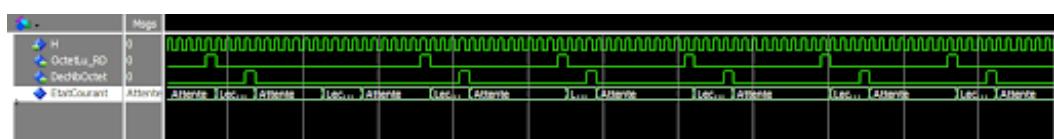


FIGURE 40 – Simulation du comptage/décomptage du nombre d'octets

Lorsque la lecture est effectuée par le microprocesseur, l'analyse montre que le signal `DecNbOctet` passe brièvement à l'état haut à chaque cycle de lecture. Ce front valide le bon décrément du compteur d'octets au fur et à mesure que le microprocesseur lit les données depuis la FIFO.

La synchronisation entre les impulsions de OctetReçu_RD et de DecNbOctet prouve ainsi que la gestion du comptage/décomptage est correcte et respecte le cahier des charges : Le compteur NbOctet doit revenir à zéro à la fin du cycle complet, confirmant la libération de la FIFO après transfert intégral des données vers le microprocesseur. Cependant ce signal n'a pas été implémenté dans la FIFO.

Gestion correcte du bit de signalisation M_Received

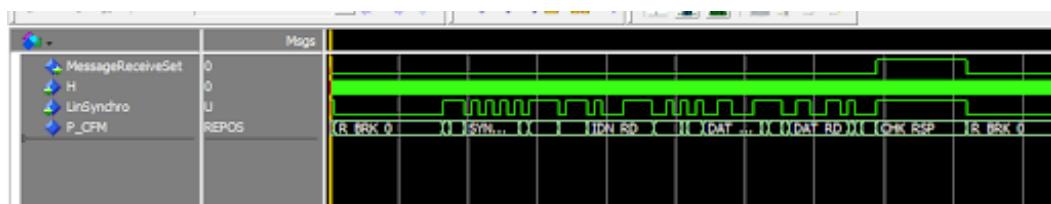


FIGURE 41 – Simulation de la gestion correcte du bit de signalisation M_Received

Le signal M_Received est mis à 1 lorsque la trame LIN est entièrement reçue et stockée dans la FIFO. L'analyse du chronogramme montre que le signal M_Received passe à l'état haut juste après la réception complète de la trame LIN (L'état étant arrivé au niveau Checksum Reception bit de Stop (CHK_RSP)), indiquant que toutes les données sont prêtes à être lues par le microprocesseur

Gestion des erreurs



FIGURE 42 – Simulation de la gestion des erreurs - Cas 1 : Erreur de Stop

Le chronogramme illustre un scénario de défaillance où le système rencontre une erreur de réception de bit Stop. On observe l'activation du signal Error_Stop, qui intervient dans une séquence de fin de trame LIN, dans le champ Checksum Reception de bit de Stop. Cette erreur provoque une réponse du système, matérialisée par la mise à 1 du signal Error.

8.2.5 Analyse du chronogramme de simulation

L'analyse des chronogrammes valide plusieurs points clés du cahier des charges et de l'énoncé de TP :

9 Synthèse des fonctions

Une fois la validation en simulation des différents blocs effectuée, il est nécessaire de réaliser la synthèse sur FPGA afin d'observer les ressources logiques attribuées à notre système. Dans le cadre de ce projet, nous avons utilisé un FPGA *AMD Xilinx Artix-7*, plus précisément le modèle **7A35TCPG236**. L'objectif de cette étape est d'analyser les ressources logiques mobilisées, ainsi que les éléments matériels effectivement utilisés par notre conception.

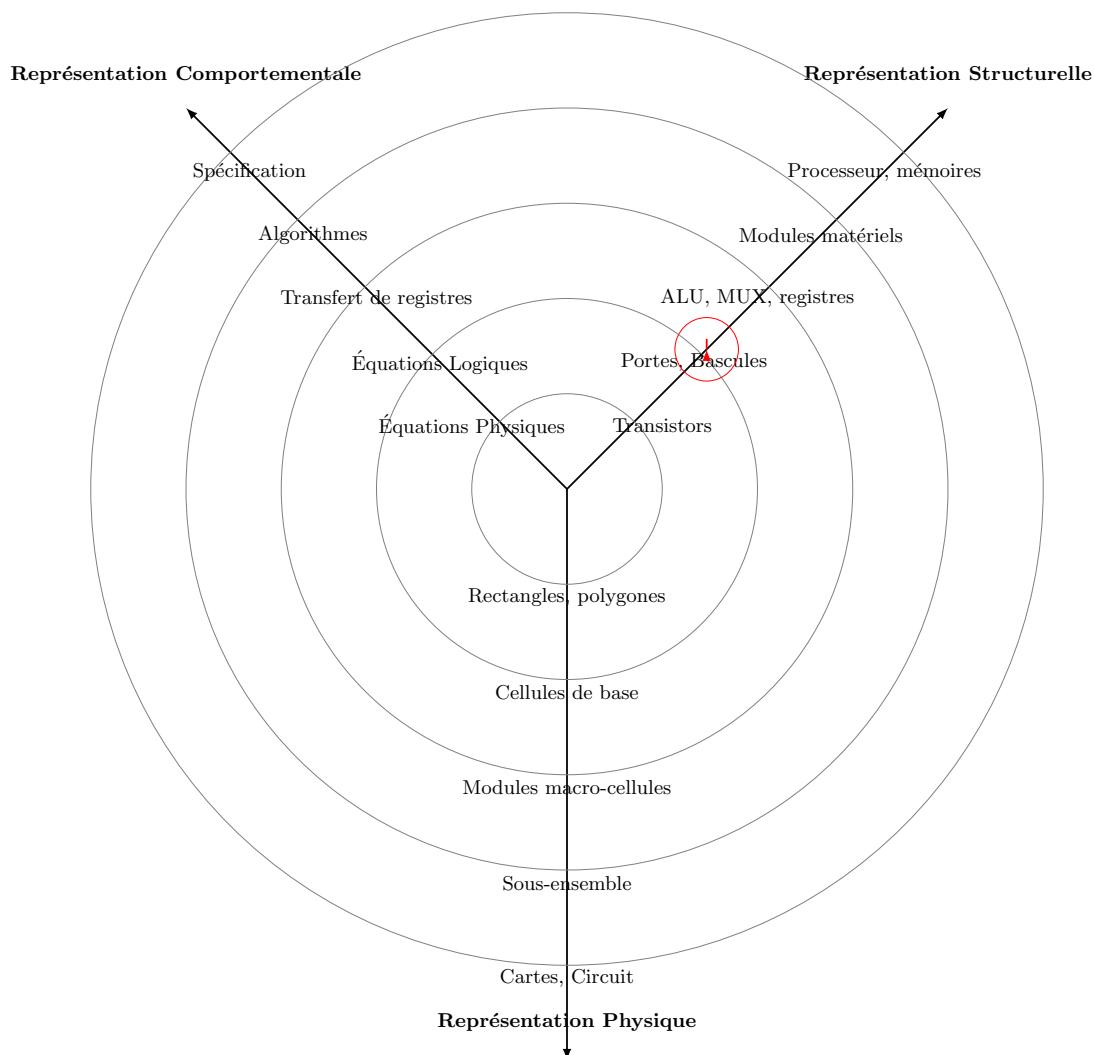


FIGURE 43 – Diagramme en Y - Action de synthèse

9.1 Interface Micropuce

Le schéma RTL (*Register Transfer Level*) représente une implémentation synthétisée d'un module matériel décrit en VHDL ou Verilog. Il illustre les registres, les multiplexeurs, les portes logiques, ainsi que la logique séquentielle et combinatoire du circuit.

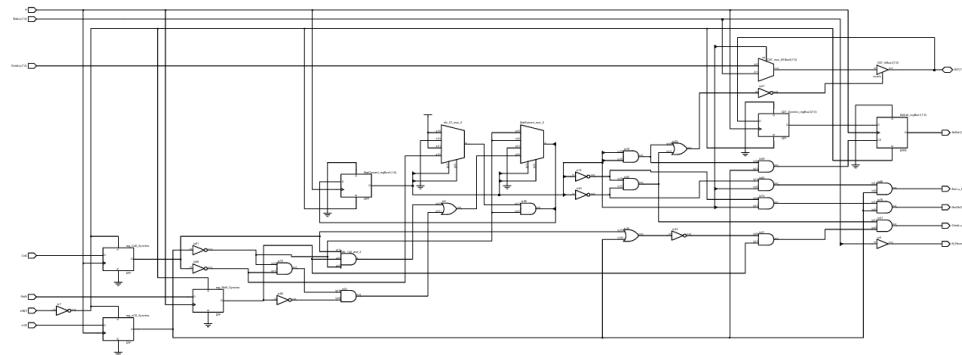


FIGURE 44 – Schéma RTL InterfaceMicroprocesseur

Structure générale

Le schéma peut être décomposé en plusieurs parties :

- **Entrées principales** : signaux tels que H, CnD, RnW, nRST, nCS, etc.
- **Registres (Flip-Flops D)** : éléments synchronisés par l'horloge, servant à mémoriser l'état interne du circuit.
- **Multiplexeurs (MUX)** : permettent de sélectionner une donnée parmi plusieurs, selon les conditions de contrôle.
- **Logique combinatoire** : réalisée par des portes AND, OR, NOT et XOR, afin de générer les conditions de transition et les sorties.
- **Sorties** : plusieurs signaux dérivés de l'état interne, comme State_XX, Output_XX, etc.

Fonctionnement global

Le circuit implémente une **machine à états finis** :

- Les **registres** contiennent l'état courant.
- La **logique combinatoire** calcule l'état suivant en fonction de l'état courant et des entrées.
- Les **multiplexeurs** dirigent les transitions entre états.
- Les **sorties** sont activées ou désactivées selon l'état courant et certaines combinaisons d'entrées.

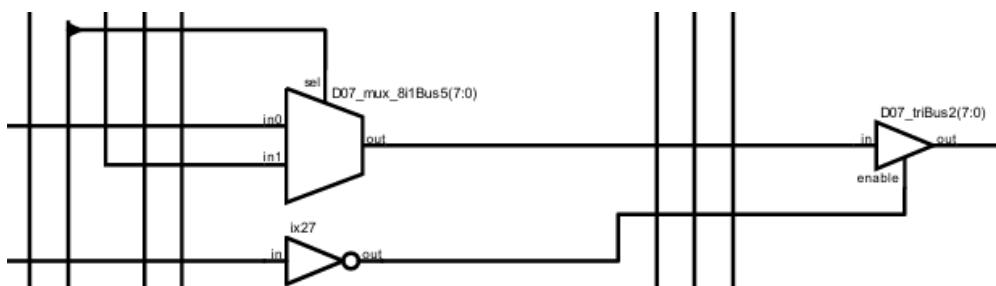


FIGURE 45 – Partie opérative avec multiplexeur et Tristate : InterfaceMicroprocesseur

De plus, nous pouvons retrouver la partie opérative dessinée en classe, lors de nos TD, qui permet, grâce à un multiplexeur, de sélectionner **EtallLu** ou **OctetLu** pour l'envoyer vers une porte **Tristate**. Cela démontre la cohérence entre la réalisation théorique et la mise en œuvre

pratique.

À la suite de la synthèse logique nous pouvons avoir la synthèse matériel qui transforme la logique en ressource matérielle.

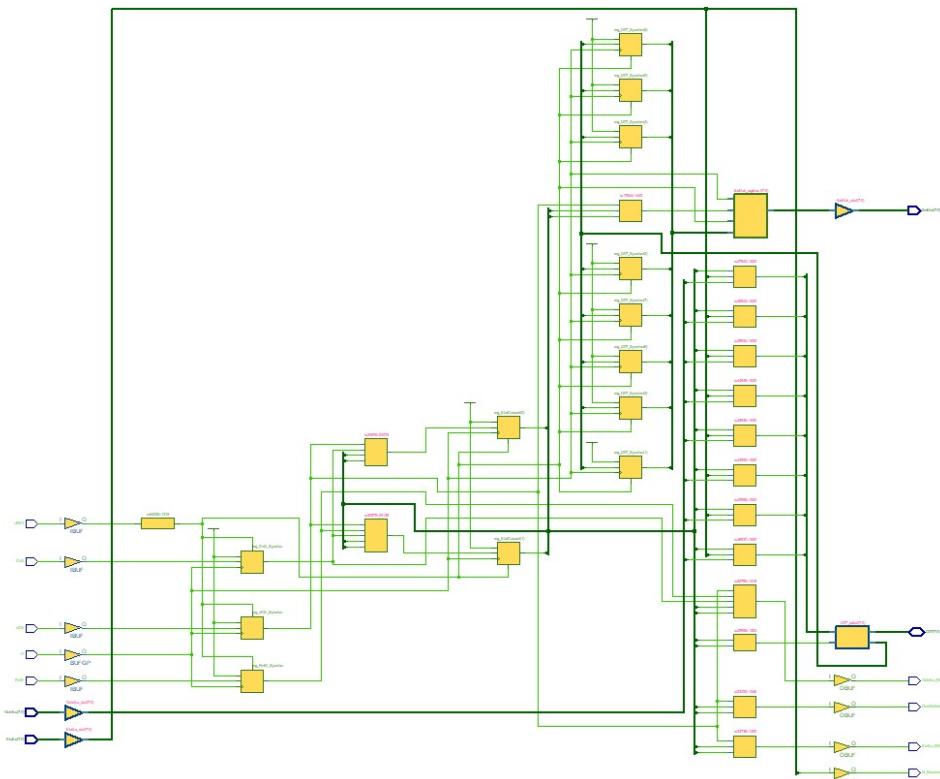


FIGURE 46 – Synthèse matérielle InterfaceMicropuceur

Cette transformation consiste à mapper les éléments logiques du schéma RTL, tels que les portes **AND**, **OR**, **NOT** ou les **multiplexeurs**, sur les **ressources matérielles physiques** disponibles dans le FPGA, notamment :

- **LUT (Look-Up Tables)** : les fonctions combinatoires, comme les portes logiques ou les multiplexeurs, sont réalisées à l'aide de LUT. Chaque LUT peut implémenter n'importe quelle fonction booléenne sur un nombre limité d'entrées, ce qui permet de reproduire fidèlement la logique définie dans le HDL.
- **Flip-flops** : les éléments séquentiels tels que les registres ou les bascules sont mappés sur des flip-flops pour stocker les bits et synchroniser les signaux dans le temps.
- **Buffers** : certains chemins logiques nécessitent des buffers pour renforcer les signaux ou adapter les niveaux électriques, garantissant ainsi la stabilité et l'intégrité du routage sur la puce.

Grâce à cette conversion, le design passe d'une **représentation abstraite de la logique** à une **implémentation matérielle concrète**, optimisée pour le FPGA cible. Cela permet non seulement de visualiser l'organisation physique des composants.

9.2 Interface Reception Trame

Comme pour l'Interface Microprocesseur, nous avons le schéma RTL de l'Interface Reception Trame. Celui ci représente la machine squénce finie implémentée dans le FPGA. Celui ci nous permet de visualiser les différents registres, multiplexeurs et portes logiques utilisés pour la conception de cette machine.

Voici le schéma RTL de l'Interface Reception Trame :

Structure générale

Le schéma peut être décomposé en plusieurs parties :

- **Entrées principales** : signaux tels que LIN, SeldADR, nCLR, H, etc.
- **Registres (Flip-Flops D)** : éléments synchronisés par l'horloge, servant à mémoriser l'état interne du circuit.
- **Multiplexeurs (MUX)** : permettent de sélectionner une donnée parmi plusieurs, selon les conditions de contrôle.
- **Logique combinatoire** : réalisée par des portes AND, OR, NOT et XOR, afin de générer les conditions de transition et les sorties.
- **Sorties** : plusieurs signaux de sortie pour la FIFO et l'EtatInterne.

Fonctionnement global

Nous pouvons retrouver la partie opérative dessinée en classe, lors de nos TD, celle ci contenant, des multiplexeurs et des portes logiques, ainsi que des compteurs/décompteurs.

Voici la partie opérative de l'Interface Reception Trame :

A la suite nous obtenons la synthèse matérielle qui nous permet de visualiser les ressources matérielles utilisées pour la conception de cette machine.

Voici la synthèse matérielle de l'Interface Reception Trame :

Comme pour l'Interface Microprocesseur, cette transformation consiste à mapper les éléments logiques du schéma RTL, tels que les portes **AND**, **OR**, **NOT** ou les **multiplexeurs**, sur les **ressources matérielles physiques** disponibles dans le FPGA, notamment :

- **LUT (Look-Up Tables)** : les fonctions combinatoires, comme les portes logiques ou les multiplexeurs, sont réalisées à l'aide de LUT. Chaque LUT peut implémenter n'importe quelle fonction booléenne sur un nombre limité d'entrées, ce qui permet de reproduire fidèlement la logique définie dans le HDL.
- **Flip-flops** : les éléments séquentiels tels que les registres ou les bascules sont mappés sur des flip-flops pour stocker les bits et synchroniser les signaux dans le temps.
- **Buffers** : certains chemins logiques nécessitent des buffers pour renforcer les signaux ou adapter les niveaux électriques, garantissant ainsi la stabilité et l'intégrité du routage sur la puce

9.3 FIFO

Comme pour les deux autres modules, nous avons le schéma RTL de la FIFO :

9.4 EtatInterne

Comme pour les deux autres modules, nous avons le schéma RTL de la EtatInterne :

Dans celui ci nous pouvons observer les différents registres, multiplexeurs et portes logiques utilisés pour la conception de cette machine.

Nous observons une bascule D qui permet de socket l'erreur de bit Synchro, comme vu dans la partie TD.

9.5 Reception LIN

A la suite nous avons décidé d'observer le schéma RTL de la Reception LIN, celui ci regroupant tous les systèmes précédents. Dans ce schéma nous allons retrouver les différents blocs que nous avons synthétisé précédemment, à savoir :

- Interface Microprocesseur
- Interface Reception Trame
- FIFO
- EtatInterne

Ce schéma nous permet d'observer toute les ressources logiques utilisées pour la conception de notre projet.

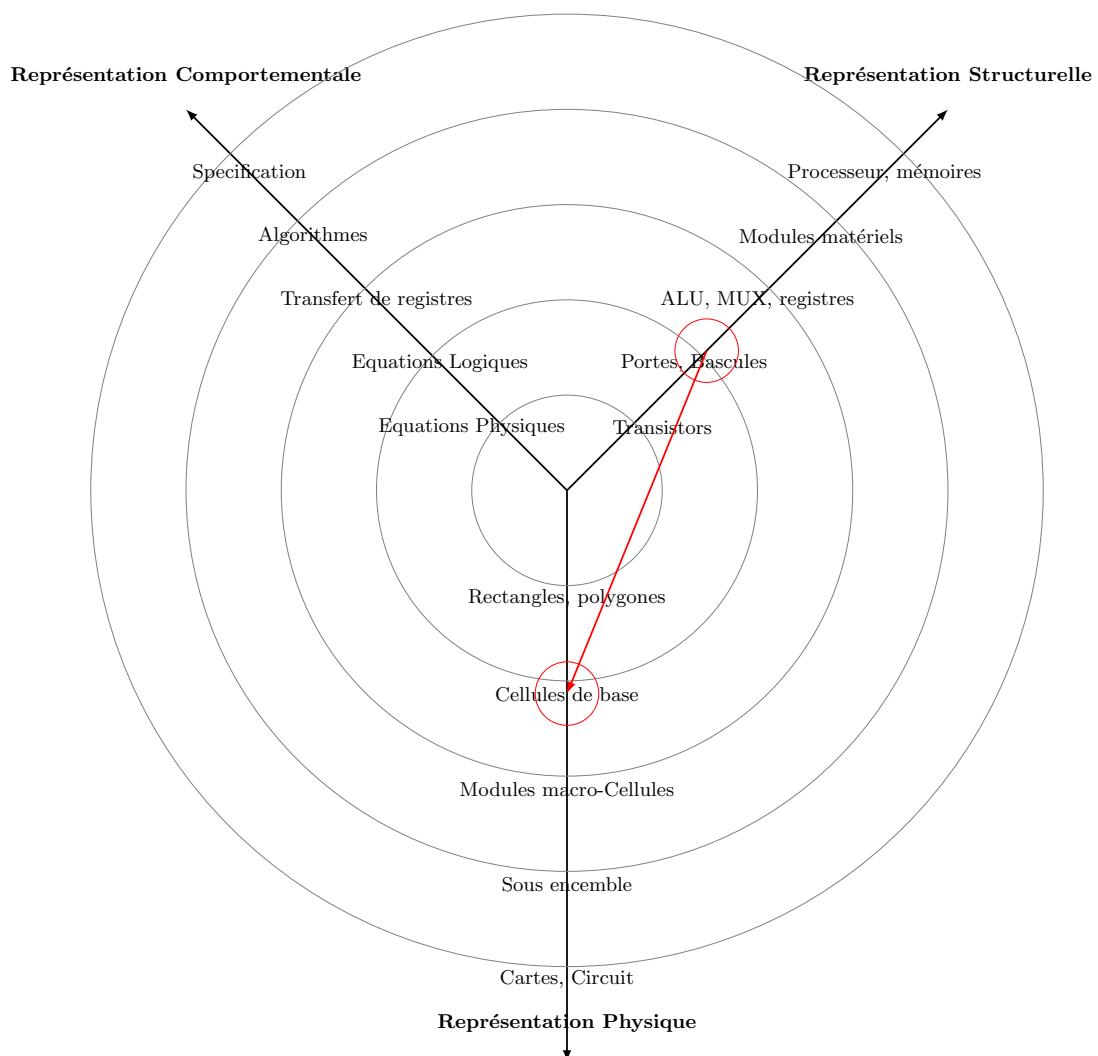


FIGURE 47 – Diagramme en Y - Action de routage

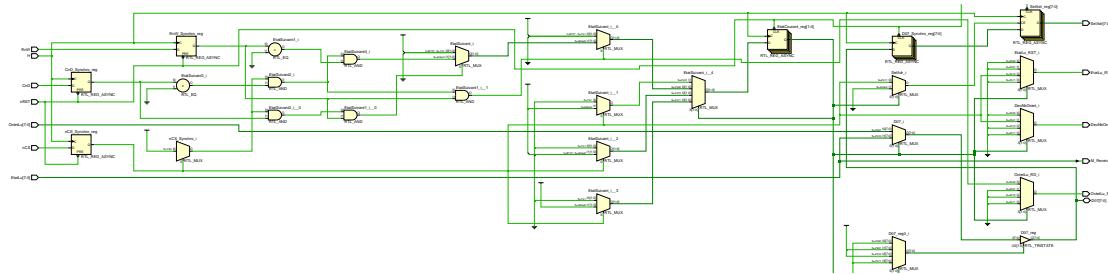
10 Routages des Fonctions

10.1 Interface Microprocesseur

Après l'étape de **synthèse**, nous pouvons nous intéresser à l'assignation des ressources matérielles du système.

Pour cela, nous utilisons le logiciel **Vivado** (étant donné que les outils de HDL n'étaient pas disponibles le jour du TP), qui permet de générer un schéma RTL ainsi qu'une vue du routage associé à l'interface microprocesseur.

Dans un premier temps, Nous allons resynthétiser le design afin d'obtenir le schéma RTL. Ce schéma, présenté ci-dessous, illustre les ressources logiques utilisées pour implémenter l'interface microprocesseur.



Par la suite, nous pouvons également observer la synthèse matérielle réalisée sur Vivado, qui transforme les ressources logiques en ressources matérielles pour le FPGA.

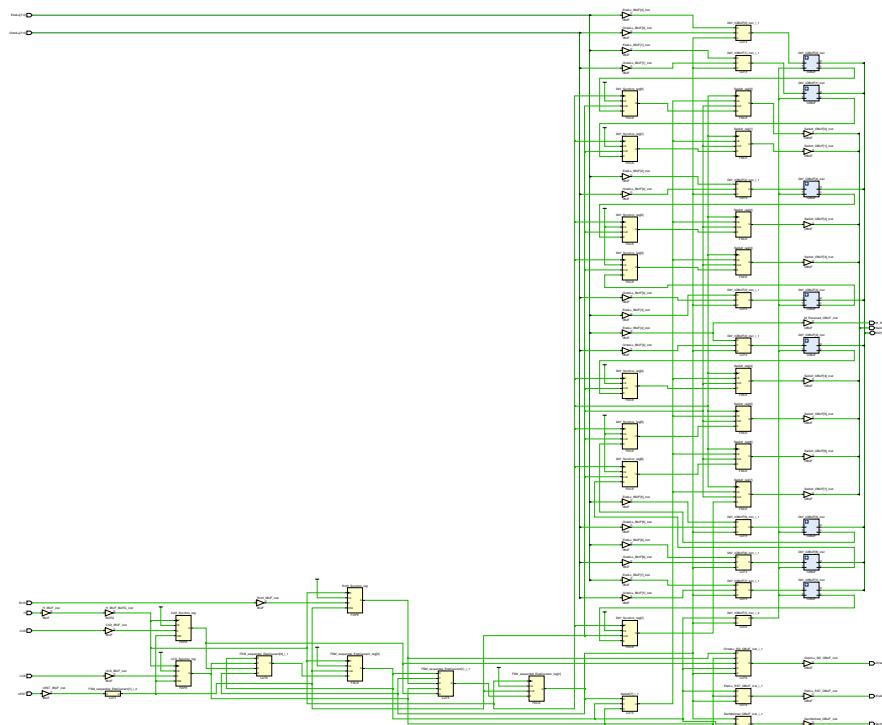


FIGURE 48 – Synthèse matérielle Vivado

Cette synthèse nous montre que la logique est transformée en ressource matérielle, notamment des LUT (Look-Up Tables) et des Flip-Flops, qui sont les éléments de base pour implémenter la logique dans un FPGA.

Les **LUT** (Look-Up Tables), éléments fondamentaux d'un FPGA, peuvent être considérées comme des portes logiques programmables capables de réaliser toute fonction combinatoire. Elles constituent la base de l'implémentation matérielle et offrent une vision schématique complète du système.

Prenons l'exemple d'une **LUT3** :

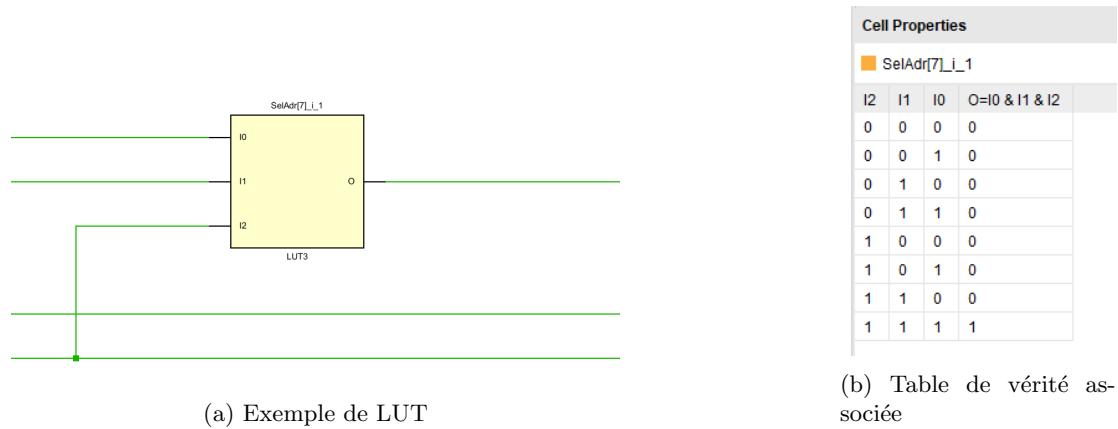


FIGURE 49 – Illustration d'une LUT et de sa table de vérité

Grace à la table de vérité de la LUT nous remarquons que cette LUT3 implémente une fonction logique ET à trois entrées.

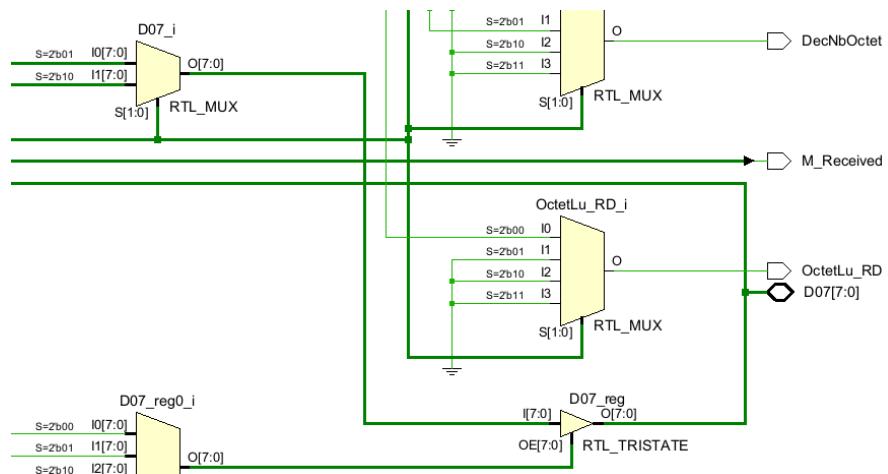


FIGURE 50 – Synthèse Logique Vivado

Nous observons également la présence de la partie opérative de l'interface Microprocesseur.

La suite du flot de conception consiste à lancer l'**implémentation** du système afin d'obtenir le routage complet. Vivado propose alors une vue générale du FPGA, mettant en évidence ses différentes zones fonctionnelles :

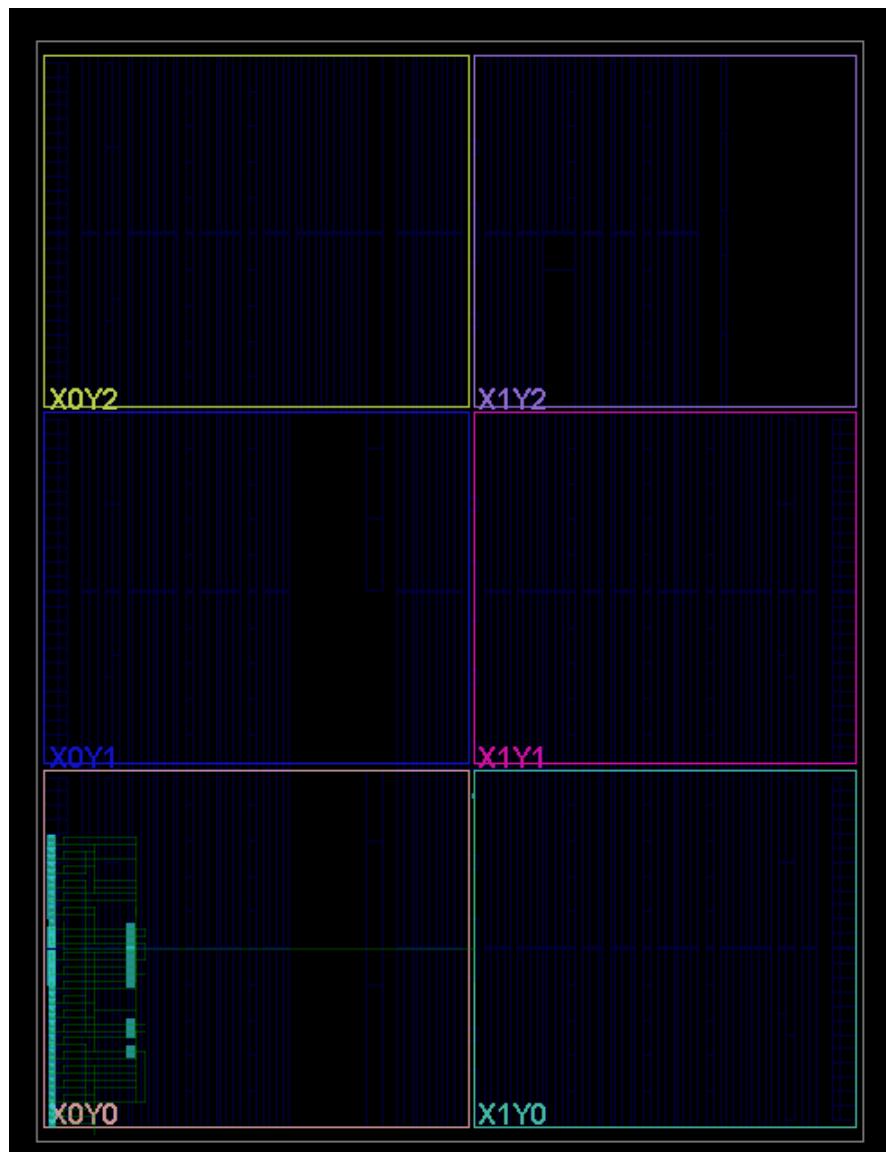


FIGURE 51 – Slice du FPGA après routage

En effectuant un zoom, il est possible de constater que le système a été implémenté dans la zone **0** du FPGA. Nous observons que la zone située à gauche correspond aux entrées de chaque variable, celles-ci étant toutes reliées à des **buffers**.

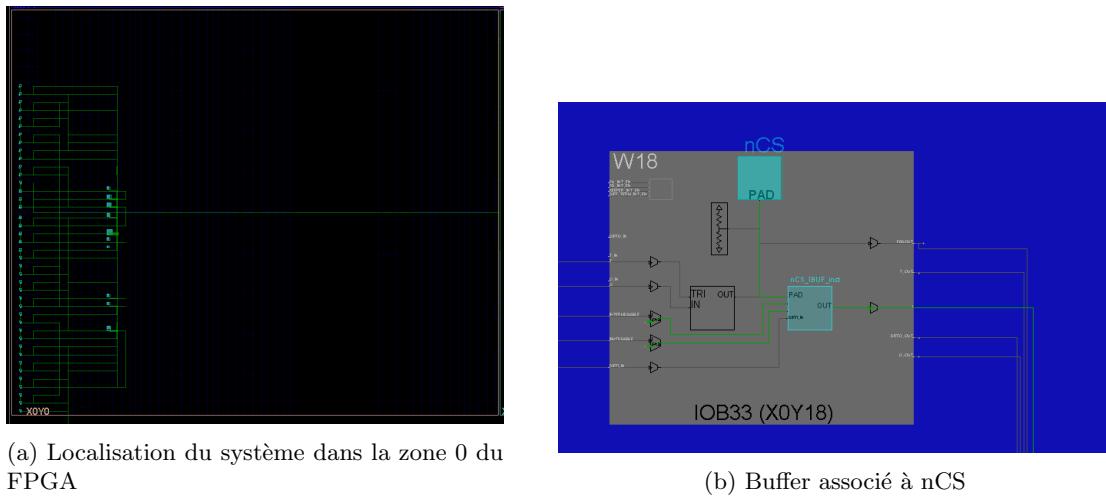


FIGURE 52 – Vue du système routé et des buffers associés sur le FPGA

Un zoom encore plus détaillé permet d'observer le câblage interne des ressources identifiées lors de la synthèse. Par exemple, une **LUT3** est câblée de la manière suivante :

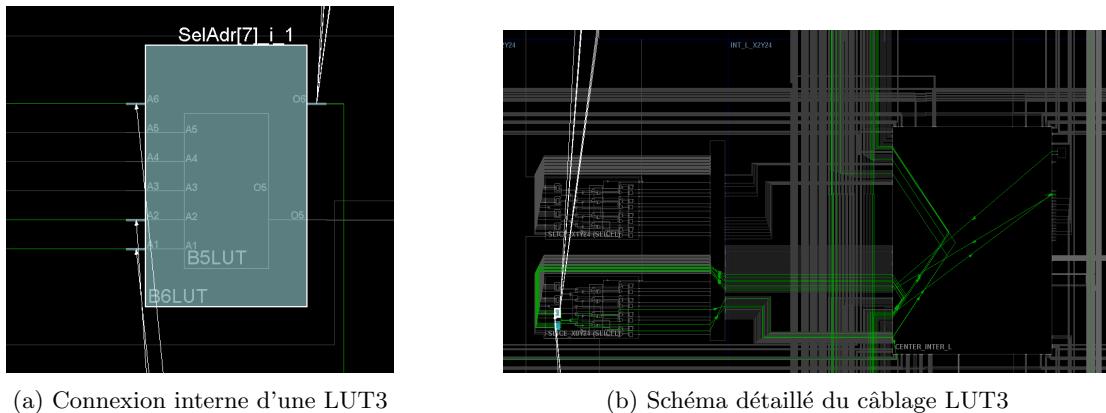


FIGURE 53 – Exemple de routage d'une LUT3 dans le FPGA

Le résultat final est une représentation complète et hiérarchisée du système, directement mappée sur le FPGA. **Vivado** offre ainsi la possibilité de visualiser l'ensemble du flot, depuis la description logique RTL jusqu'au routage physique détaillé.

En résumé, la conception suit une progression en trois étapes :

- la **synthèse** génère une description logique optimisée du système (LUT, registres, blocs fonctionnels) ;
- le **placement** attribue ces ressources aux cellules physiques du FPGA ;
- le **routage** établit les interconnexions nécessaires au bon fonctionnement du circuit.

Cette approche permet de passer d'une description abstraite en langage HDL à une implémentation matérielle concrète, où chaque fonction logique est traduite en ressources physiques. Vivado fournit alors une vision globale et détaillée du FPGA, allant de la logique combinatoire jusqu'au câblage interne des composants.

10.2 FIFO

Après l'étape de synthèse, nous pouvons nous intéresser à l'assignation des ressources matérielles du système FIFO, qui permet de générer un schéma de ressources utilisées pour l'implémentation de la FIFO.

Voici le schéma RTL de la FIFO implémentée :

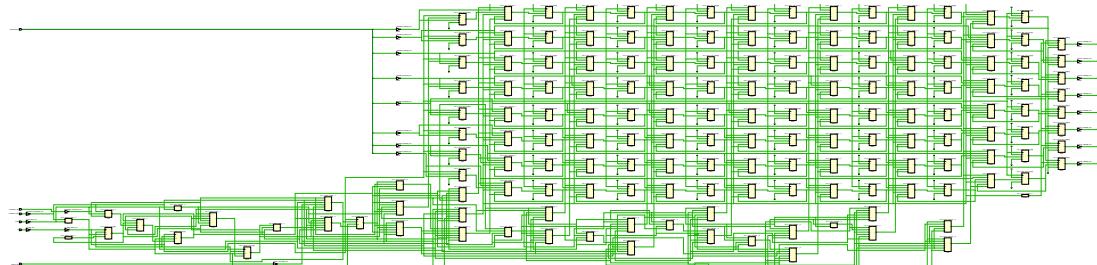


FIGURE 54 – Schéma Vivado de la FIFO - Vue générale

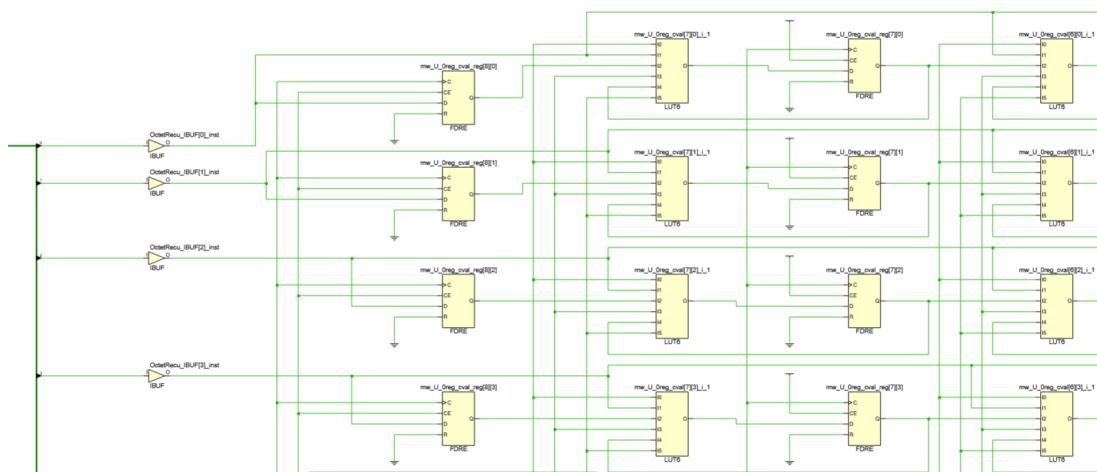


FIGURE 55 – Schéma Vivado de la FIFO - Vue détaillée

On identifie clairement l'architecture dual-port de la FIFO avec ses **registres** organisés en matrice 9x8 bits pour le stockage des données. La gestion des adresses est assurée par un **compteur 4 bits** implanté avec des bascules **FDCE** et des **LUT** qui calculent les incrémentations et les conditions de débordement. Les entrées-sorties sont conditionnées par des **buffers** pour l'interface avec l'extérieur du FPGA, tandis qu'un **buffer global** distribue l'horloge à l'ensemble du système. La logique de contrôle, répartie dans de multiples **LUT2 à LUT6**, gère simultanément les opérations d'écriture via **OctetRecu_WR** et de lecture via **OctetLu_RD** tout en maintenant la cohérence des pointeurs et en générant les signaux de statut.

10.3 EtatInterne

Après l'étape de synthèse, nous pouvons nous intéresser à l'assignation des ressources matérielles du système EtatInterne, qui permet de générer un schéma de ressources utilisées pour l'implémentation

de l'EtatInterne.

Voici le schéma RTL de l'EtatInterne implémentée :

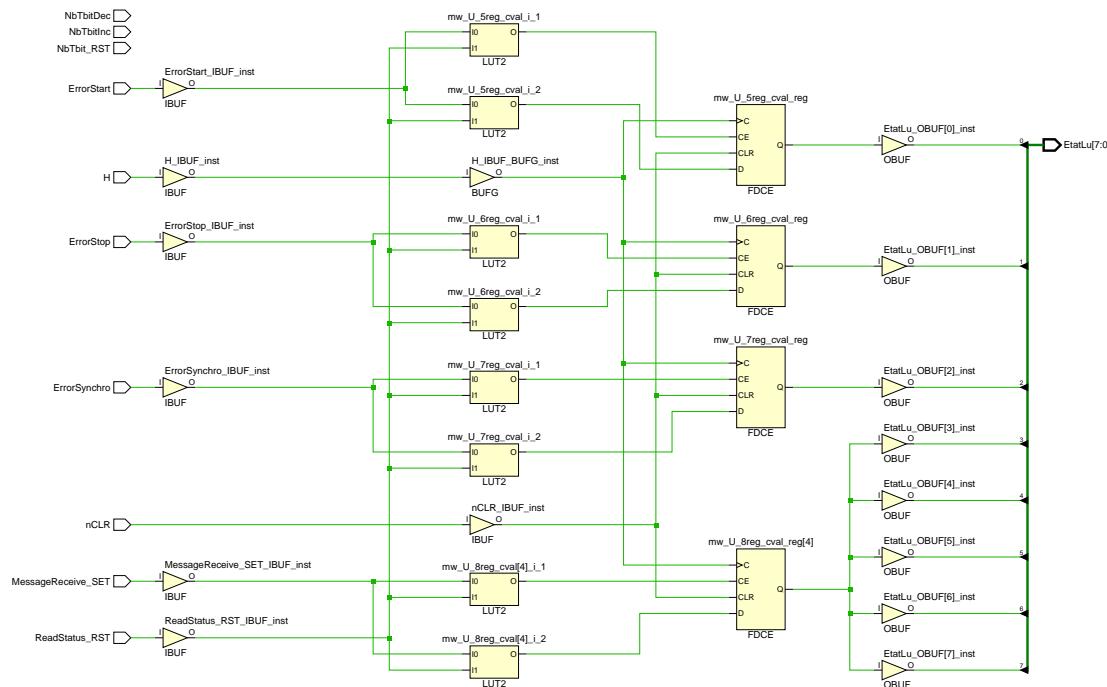


FIGURE 56 – Schéma RTL de l'EtatInterne

On identifie clairement un ensemble de **bascules D**) organisées pour mémoriser les différents états du système. Ces bascules sont pilotées par une logique combinatoire implémentée via des **LUT** , qui définissent les transitions d'état en fonction des signaux de commande et des conditions internes. Les entrées sont conditionnées par des **buffers** afin d'isoler et d'adapter les signaux externes aux ressources internes du FPGA, assurant ainsi une intégrité électrique et temporelle. En sortie, des **buffers** sont utilisés pour transmettre l'état courant vers les blocs suivants.

Dans ce schéma nous avons des signaux non, reliés surement une erreur de Vivado. Normalement ces signaux devraient être connectés à une LUT dont la table de vérité permettrait de réaliser la fonction **DECODE**, qui permet de choisir entre incrémentation et décrémentation du compteur du nombre d'octet. .

10.4 Interface Reception Lin

Cette Phase n'étant pas réellement nécessaire pour la validation du projet, nous avons décidé de juste de faire l'implémentation du projet global, et non de montrer les ressources de chacune des sections.

Donc comme l'interface microprocesseur, nous avons fait l'implémentation de l'interface de réception LIN (global) et voici le résultat du routage :

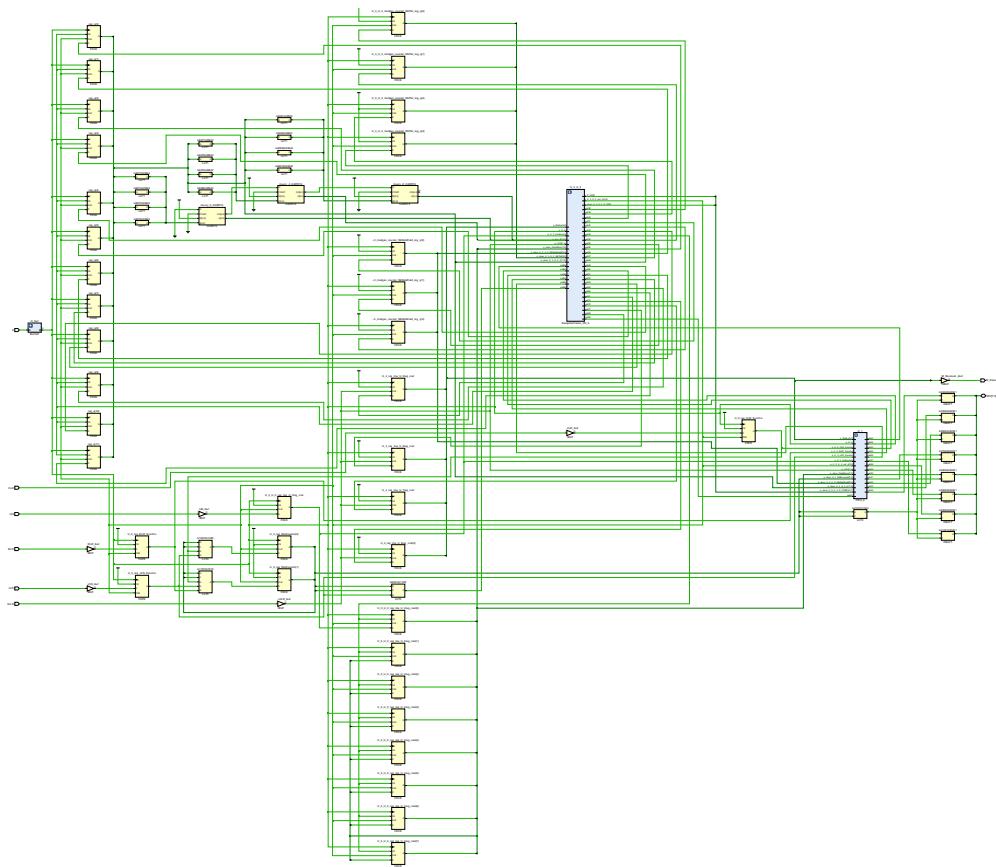


FIGURE 57 – Schéma Implémentation Matérielle Reception LIN

Dans celui ci, nous pouvons observer les différentes ressources matérielles utilisées pour l'implémentation de l'interface de réception LIN. notamment les LUTs, les Flip-Flops, et les autres composants logiques.

Ce schéma étant dense et complexe, nous n'allons pas détailler chaque partie, mais il illustre bien comment les ressources matérielles sont utilisées pour réaliser les fonctions de l'interface de réception LIN. Voici un exemple de LUT retrouvée dans le routage de l'interface de réception Trame :

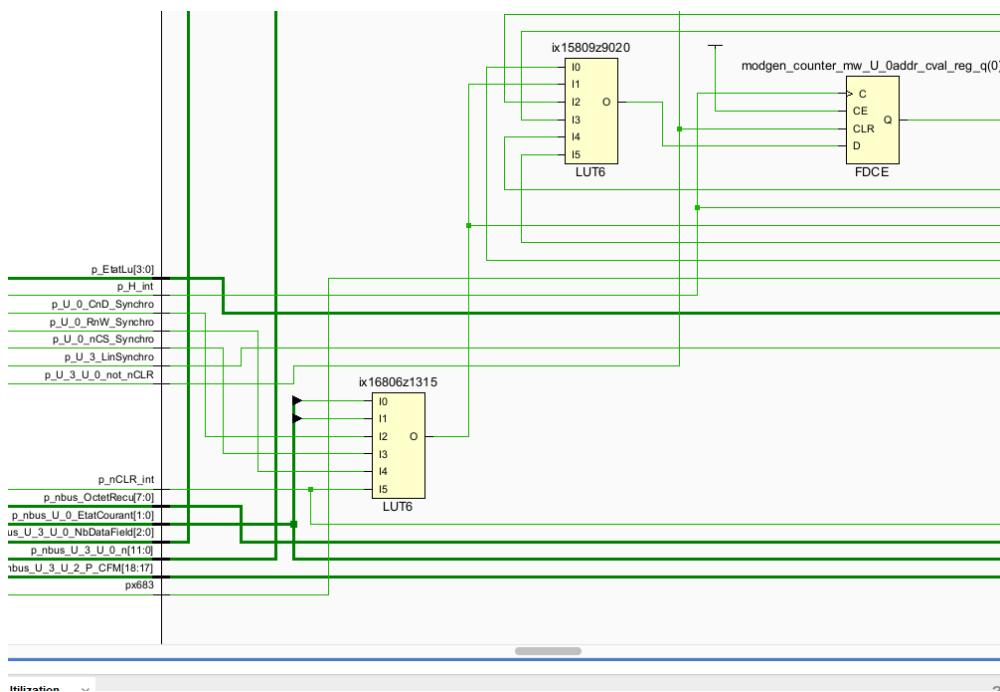


FIGURE 58 – Schéma Implémentation Matérielle Reception LIN Zoom 1

Dans ce zoom, nous pouvons voir une LUT spécifique utilisée dans le routage de l'interface de réception LIN. La présentation d'un LUT est plus détaillée dans la section précédente.

Voici également le routage global de l'interface de réception LIN sur le FPGA :



FIGURE 59 – Slice du FPGA après routage de l'interface de réception LIN

Pareillement, cette représentation étant dense, nous n'allons pas détailler chaque partie, mais elle illustre comment les ressources matérielles sont utilisées pour réaliser les fonctions de l'interface de réception LIN.

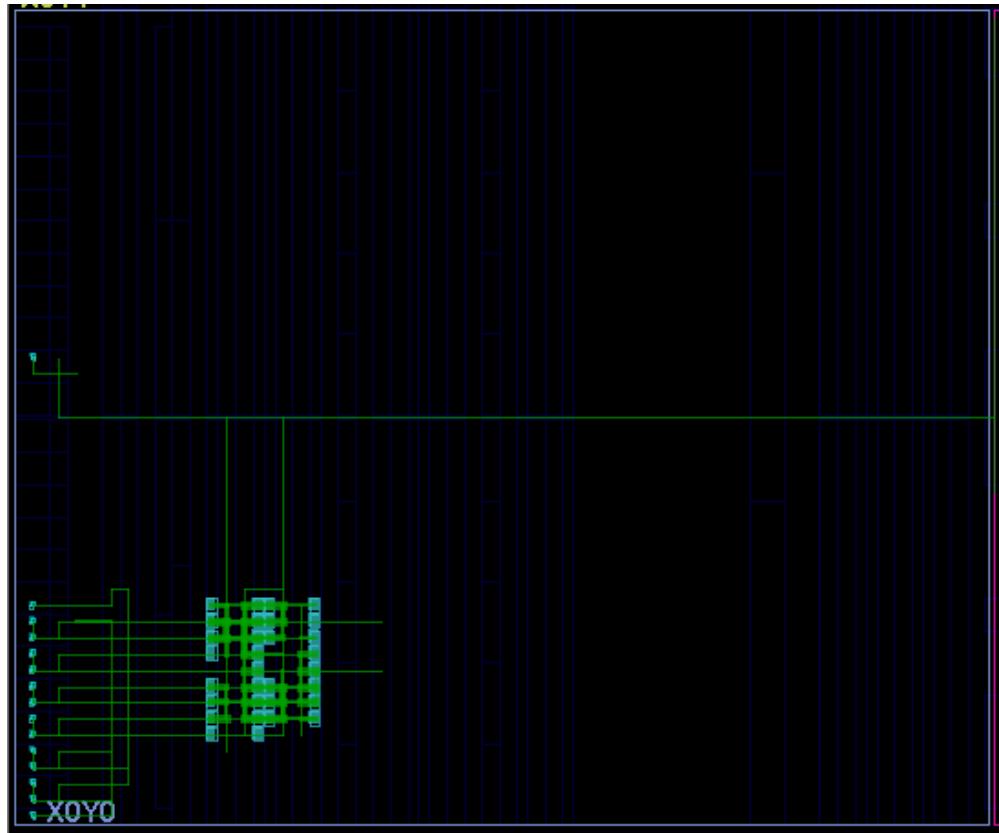


FIGURE 60 – Slice du FPGA après routage de l'interface de réception LIN - Zoom sur Zone 0

A la fin du projet nous avons la possibilité de voir l'utilisation des ressources du FPGA par Vivado.

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFCTRL (32)
RecepteurLin	140	132	55	140	15	1

TABLE 7 – Utilisation des ressources FPGA pour le module RecepteurLin

11 Conclusion

Ce projet de **Réception LIN** en conception de circuit numérique nous a permis de mettre en œuvre une démarche complète, depuis l'analyse du cahier des charges jusqu'à l'implémentation finale sur FPGA.

Au cours des différentes étapes, nous avons appliqué une méthodologie rigoureuse basée sur le **diagramme en Y**, ce qui nous a permis de structurer efficacement notre réflexion et de maîtriser chaque niveau de conception.

La phase de **spécification fonctionnelle** a défini les ressources nécessaires et les signaux principaux. La **solution architecturale** a ensuite permis de modéliser clairement le système à l'aide de machines séquentielles (Moore et Mealy), en distinguant la partie commande et la partie opérative. Cette approche a facilité l'écriture d'un code **VHDL clair, modulaire et cohérent**.

Les phases de **simulation** et de **synthèse** ont confirmé la validité du fonctionnement logique et le bon découpage du système en ressources matérielles (LUT, bascules, multiplexeurs, etc.). Enfin, l'**implémentation sur FPGA** a validé le fonctionnement global du récepteur LIN, en assurant le respect des contraintes temporelles et la conformité au cahier des charges.

Ainsi, ce projet nous a permis de concevoir et de valider avec succès un système complet de **réception LIN**, pleinement opérationnel et prêt à être intégré dans un environnement embarqué. Il nous a également apporté une meilleure maîtrise des outils de conception numérique (*Vivado*, simulations *VHDL*) et une compréhension approfondie du lien entre la modélisation logique et la réalisation matérielle.

12 Annexes

12.1 Testbench InterfaceMicroprocesseur

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.std_logic_arith.all;
4
5 ENTITY EnvTest_InterfaceMicroprocesseur IS
6     GENERIC(
7         CLOCK_PERIOD      : time := 50 ns;
8         RESET_OFFSET     : time := 500 ns;
9         RESET_DURATION   : time := 300 ns;
10        ACCESS_TIME     : time := 40 ns;
11        HOLD_TIME       : time := 70 ns
12    );
13    PORT(
14        M_Received : IN      std_logic;
15        CnD        : OUT     std_logic;
16        H          : OUT     std_logic;
17        RnW        : OUT     std_logic;
18        nCS        : OUT     std_logic;
19        nRST       : OUT     std_logic;
20        D07        : INOUT   std_logic_vector (7 DOWNTO 0)
21    );
22    -- Declarations
23
24 END EnvTest_InterfaceMicroprocesseur ;
25
26 --
27 ARCHITECTURE arch OF EnvTest_InterfaceMicroprocesseur IS
28     TYPE DefState IS (Waiting, DataReading, StateReading, FilterWriting);
29
30     SIGNAL ProcessorState : DefState;
31
32 BEGIN
33
34     ClockGeneratorProc : PROCESS
35     BEGIN
36         H <= '0';
37         WAIT FOR CLOCK_PERIOD/2;
38         H <= '1';
39         WAIT FOR CLOCK_PERIOD/2;
40     END PROCESS ClockGeneratorProc;
41
42     ResetGeneratorProc : PROCESS
43     BEGIN
44         nRST <= '1';
45         WAIT FOR RESET_OFFSET;
46         nRST <= '0';
47         WAIT FOR RESET_DURATION;
48         nRST <= '1';
49         WAIT;
50     END PROCESS ResetGeneratorProc;
51
52     ProcessorBehaviorProc : PROCESS

```

```

53 BEGIN
54     D07 <= (others => 'Z');
55 --Waiting cycle--
56     ProcessorState <= Waiting;
57     nCS <= '1';
58     CnD <= '1';
59     RnW <= '1';
60     WAIT FOR RESET_OFFSET+RESET_DURATION+2*CLOCK_PERIOD;
61 --Reading data cycle--
62     ProcessorState <= DataReading;
63     WAIT FOR ACCESS_TIME;
64     nCS <= '0';
65     CnD <= '0';
66     RnW <= '1';
67     WAIT FOR 2*CLOCK_PERIOD;
68 --Waiting cycle--
69     ProcessorState <= Waiting;
70     nCS <= '1';
71     CnD <= '1';
72     RnW <= '1';
73     WAIT FOR 2*CLOCK_PERIOD-ACCESS_TIME;
74 --Reading state cycle--
75     ProcessorState <= StateReading;
76     WAIT FOR ACCESS_TIME;
77     nCS <= '0';
78     CnD <= '1';
79     RnW <= '1';
80     WAIT FOR 2*CLOCK_PERIOD;
81 --Waiting cycle--
82     ProcessorState <= Waiting;
83     nCS <= '1';
84     CnD <= '1';
85     RnW <= '1';
86     WAIT FOR 2*CLOCK_PERIOD-ACCESS_TIME;
87 --Writing cycle--
88     ProcessorState <= FilterWriting;
89     WAIT FOR ACCESS_TIME;
90     nCS <= '0';
91     CnD <= '0';
92     RnW <= '0';
93     D07 <= (others => '1');
94     WAIT FOR 2*CLOCK_PERIOD;
95 --Waiting cycle--
96     ProcessorState <= Waiting;
97     nCS <= '1';
98     CnD <= '1';
99     RnW <= '1';
100    WAIT FOR HOLD_TIME;
101    D07 <= (others => 'Z');
102    WAIT;
103 END PROCESS ProcessorBehaviorProc;
104
105 END ARCHITECTURE arch;

```

Listing 9 – Testbench InterfaceMicroprocesseur

Table des figures

1	Exemple d'architecture d'un réseau dans un véhicule	4
2	Exemple d'architecture LIN	5
3	Connexion physique d'un noeud à la ligne LIN	5
4	Type de Trame Protocol LIN	6
5	Interface microprocesseur associée au circuit à concevoir	8
6	Chronogrammes des échanges entre le circuit et son environnement	8
7	Schema Conception Registre interne Système	9
8	Diagramme en Y - Spécification du circuit vers conception fonctionnelle	10
9	Descetip au circuit à concevoir	11
10	Diagramme en Y - Conception fonctionnelle vers conception architecturale	13
11	Représentation des échanges avec les entités Émetteur LIN et Système à processeur	14
12	Solution fonctionnelle après introduction des interfaces LIN V1	14
13	Architecture du système de réception de trame LIN V2	15
14	Description finale du circuit	15
15	Comportement du circuit vis à vis de l'émetteur LIN	16
16	Comportement du circuit vis à vis du microprocesseur	17
17	Diagramme en Y - Conception architecturale vers prototypage	19
18	Machine Sequentielle Reception Trame	21
19	Structure partie Opérative Reception Trame	21
20	Automate de réception de trame LIN - PART1	22
21	Automate de réception de trame LIN - PART2	23
22	Machine de MEALY Unité de Commande Interface Micro	24
23	Machine de MEALY Unité de Commande Reception Trame	24
24	Machine Sequentielle Reception Trame	25
25	Structure partie Opérative Interface Microprocesseur	26
26	Structure partie Commande Interface Microprocesseur	26
27	Machine de MEALY Unité de Commande Interface Micro	27
28	Implémentation de la FIFO	28
29	implémentation de l'état interne en représentation structurelle au niveau RT	29
30	Schéma partie opérative réception LIN	33
31	Machine Séquentielle réception LIN	44
32	Schéma bloc FIFO	45
33	Schéma bloc EtatInterne	46
34	Schéma bloc Réception LIN Complete	47
35	Chronogramme de simulation de l'Interface Microprocesseur	48
36	Block Diagramme de test de l'Interface Microprocesseur	49
37	Simulation de l'échantillonnage au milieu des bits	50
38	Simulation du stockage des bits dans le registre à décalage	51
39	Simulation de la réception correcte des octets d'une trame	52
40	Simulation du comptage/décomptage du nombre d'octets	52
41	Simulation de la gestion correcte du bit de signalisation M_Received	53
42	Simulation de la gestion des erreurs - Cas 1 : Erreur de Stop	53
43	Diagramme en Y - Action de synthèse	54
44	Schéma RTL InterfaceMicroprocesseur	55
45	Partie opérative avec multiplexeur et Tristate : InterfaceMicroprocesseur	55
46	Synthese matérielle InterfaceMicroprocesseur	56
47	Diagramme en Y - Action de routage	59
48	Synthèse matérielle Vivado	60
49	Illustration d'une LUT et de sa table de vérité	61
50	Synthèse Logique Vivado	61

51	Slice du FPGA après routage	62
52	Vue du système routé et des buffers associés sur le FPGA	63
53	Exemple de routage d'une LUT3 dans le FPGA	63
54	Schéma Vivado de la FIFO - Vue générale	64
55	Schéma Vivado de la FIFO - Vue détaillée	64
56	Schéma RTL de l'EtatInterne	65
57	Schéma Implémentation Matérielle Reception LIN	66
58	Schéma Implémentation Matérielle Reception LIN Zoom 1	67
59	Slice du FPGA après routage de l'interface de réception LIN	68
60	Slice du FPGA après routage de l'interface de réception LIN - Zoom sur Zone 0	69