SPY FPGA

Rapport de stage

Outil de monitoring intégré pour les systèmes FPGA-SoC

Soumis par : BEKDOUCHE Amine Adel

Le 25 Aout 2025

Sommaire

Introduction **3**

1 Contexte industriel et technologique **4**

1.1 GE Vernova et la transition énergétique 4

1.2 Division Electrification 4

1.3 Grid Solutions et Grid Automation 4

2 Norme IEC 61850 et protocole GOOSE **5**

2.1 Introduction à la norme IEC 61850 5

2.2 Le protocole GOOSE 5

2.3 Structure de la trame GOOSE 5

3 Présentation du projet **7**

3.1 Contexte du stage 7

3.2 Objectif du projet 7

4 Architecture matérielle ciblée **8**

4.1 Description du calculateur de tranche 8

4.2 Composition de la carte CPU 9

4.3 Cible du projet 9

5 Flux de Conception Assisté par Ordinateur pour FPGA **11**

6 Cahier des charges **12**

6.1 Objectif général 12

6.2 Exigences fonctionnelles obligatoires (Necessary) 12

6.3 Exigences fonctionnelles souhaitables (facultatif) 12

7 Développement du POC (Proof Of Concept) GOOSE-SUB **13**

7.1 Résumé : 13

7.2 Architecture 14

7.2.1 Module 1 : Interface (Spécifique aux protocoles IEC-61850) 14

7.2.2 Module 2 : Gestionnaire (Spécifique au protocole IEC-61850 GOOSE) 14

7.2.3 Module 3 : Configuration (Générique) 15

7.2.4 Module 4 : Filtre (Spécifique au protocole IEC-61850 GOOSE) 15
7.2.5 Module 5 : Statistiques (Générique) 15
7.2.6 Module 6 : Stockage (Générique) 15
7.3 Implémentation du POC (Proof Of Concept) 15
7.3.1 Gestionnaire (Manager) 16
7.3.2 Filtre 17
7.4 Synthèse 19
7.5 Conclusion Intermédiaire 20
8 Simulation et Vérification 21
8.1 Résumé 21
8.2 Méthodologie de Simulation Automatisée 21
8.2.1 Fonctionnement du point de vue de l'utilisateur 22
8.2.2 Fonctionnement de l'arrière-plan 23
8.3 Conclusion Intermédiaire 29
Perspectives 30
Conclusion 31
Annexes 32

Introduction

Dans un contexte de transition énergétique et de digitalisation des infrastructures électriques, les systèmes embarqués jouent un rôle central dans la supervision, la protection et le contrôle des réseaux électriques. GE Vernova, acteur clé du secteur énergétique, développe une variété de solutions innovantes, allant des **calculateurs de tranche**[1] aux systèmes de contrôle et d'automatisation, afin de répondre aux défis croissants de fiabilité, de performance et de sécurité.

Ce rapport s'inscrit dans le cadre d'un stage réalisé au sein de l'équipe R&D de GE Vernova Grid Solutions à Montpellier. Il retrace le développement d'un module de monitoring embarqué, nommé SPY-FPGA, conçu pour analyser les trames **GOOSE**[3] conformément à la norme **IEC-61850**[2]. Intégré dans un FPGA au cœur du calculateur de tranche, ce module vise à renforcer la traçabilité et la robustesse du système en détectant de manière fiable les anomalies survenant lors de la transmission des trames.

Le document est structuré de manière à suivre la progression logique du projet :

- Il débute par une présentation du contexte industriel et technologique, ainsi que des normes utilisées.
- Il détaille ensuite l'architecture matérielle ciblée et les exigences fonctionnelles du projet.
- Le cœur du rapport est consacré au développement du module SPY-FPGA, à son implémentation en VHDL, et à la méthodologie de simulation automatisée mise en place pour sa vérification.
- Enfin, les perspectives d'évolution du projet sont abordées, avant de conclure sur les enseignements techniques et humains tirés de cette expérience.

Ce rapport vise à démontrer la faisabilité d'une solution embarquée compacte et configurable, tout en illustrant les compétences mobilisées dans un environnement industriel exigeant.

1- Le calculateur de tranche est un système embarqué dédié à la supervision, au contrôle et à la protection des équipements d'une tranche de production électrique.
2- La norme IEC 61850 est une norme internationale qui définit les protocoles de communication et les modèles de données pour les systèmes d'automatisation des postes électriques.
3- GOOSE (Generic Object Oriented Substation Event) est un protocole de communication défini par la norme IE61850, permettant l'échange direct et rapide de messages événementiels entre équipements d'un poste électrique.

1 Contexte industriel et technologique

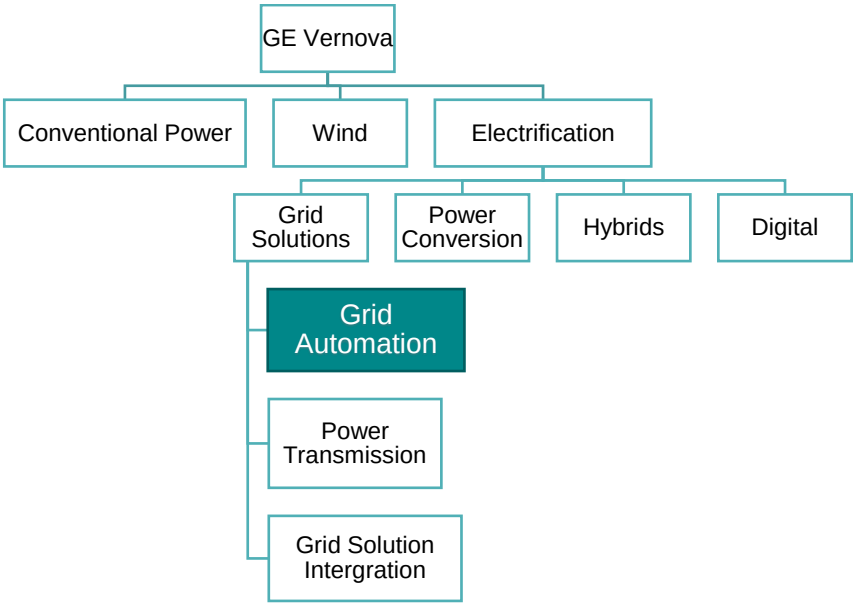


Figure 1.1 illustration de la structure hiérarchique de GE Vernova.

1.1 GE Vernova et la transition énergétique

GE Vernova est une entreprise multinationale, entièrement dédiée à la transition énergétique, issue de la transformation du groupe General Electric. Elle regroupe plusieurs pôles d'expertise couvrant la production, la conversion, la transmission et la distribution d'électricité. Sa mission est de construire un avenir énergétique plus propre, plus fiable et plus résilient.

1.2 Division Electrification

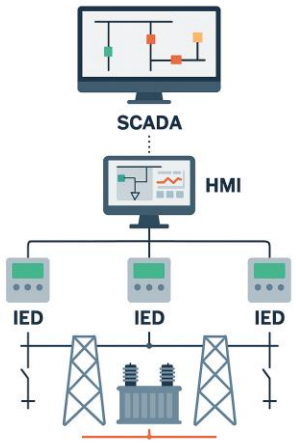
La division Electrification développe des solutions pour la transmission et la distribution de l'électricité. Elle joue un rôle clé dans la modernisation des infrastructures électriques mondiales, en intégrant des technologies numériques, des équipements haute tension et des systèmes intelligents.

1.3 Grid Solutions et Grid Automation

Au sein d'Electrification, la sous-division Grid Solutions fournit des équipements, logiciels et services pour renforcer la fiabilité, la flexibilité et la sécurité des réseaux électriques. Elle permet notamment :

- L'intégration massive des énergies renouvelables
- La gestion des flux d'énergie en temps réel
- La résilience face aux perturbations climatiques ou techniques

La branche Grid Automation est quant à elle dédiée à l'automatisation, la surveillance et le contrôle à distance du réseau électrique. Elle s'appuie sur des systèmes **SCADA**[1], des dispositifs électroniques intelligents (**IEDs**[2]), et des algorithmes avancés pour optimiser les performances du réseau.



HMI : Interface Homme Machine

Figure 1.2 : Diagrame simplifié d'un système SCADA.

La figure 1.2 illustre un diagrame simplifié d'un système SCADA pour une station électrique, incluant des **IEDs**[2], des capteurs, des actionneurs, et une interface de supervision.

2 Norme IEC 61850 et protocole GOOSE

2.1 Introduction à la norme IEC 61850

Dans le cadre de la digitalisation des postes électriques, la norme IEC 61850 s'est imposée comme un standard de communication incontournable. Elle repose sur une architecture orientée objet pour modéliser les fonctions électriques et permet une interopérabilité entre équipements de différents fabricants.

2.2 Le protocole GOOSE

Parmi les mécanismes définis par cette norme, le protocole GOOSE (Generic Object Oriented Substation Event) est essentiel pour la transmission rapide d'événements critiques entre **IEDs**[2]. Il utilise une communication Ethernet multicast, sans passer par TCP/IP, permettant des temps de transmission inférieurs à 4 ms, ce qui est crucial pour les applications de protection.

2.3 Structure de la trame GOOSE

Une trame GOOSE est composée de plusieurs sections, chacune contenant des informations spécifiques nécessaires à la transmission et à l'interprétation des événements. La structure générale est la suivante :

1. SCADA (Supervisory Control And Data Acquisition) : est un système informatique qui permet de surveiller, contrôler et collecter des données en temps réel sur des processus industriels à distance.
2. Un IED (pour Intelligent Electronic Device) est un équipement électronique intelligent utilisé dans les postes électriques pour surveiller, contrôler, protéger et automatiser les systèmes électriques.

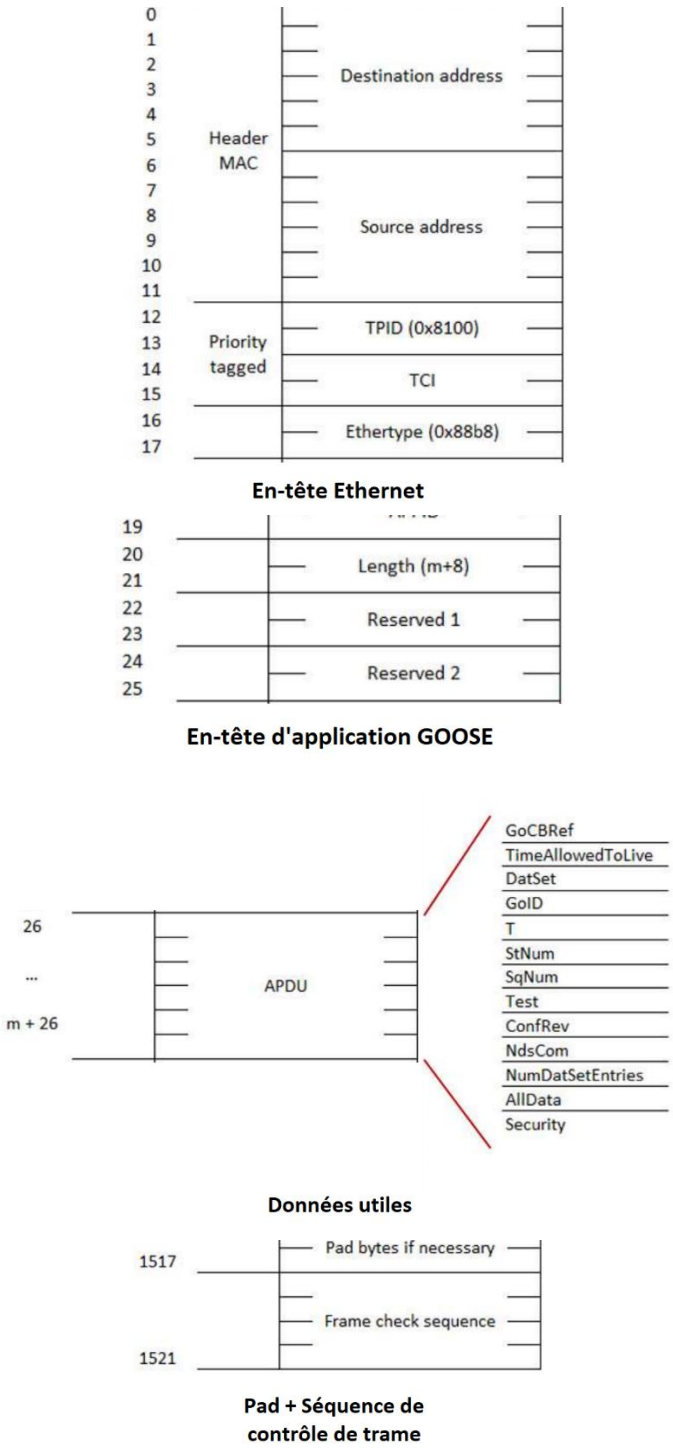


Figure 2.1 Structure de la trame GOOSE.

La trame commence par l'en-tête Ethernet (MAC, Piority tagged, Ethertype), suivie de l'en-tête d'application GOOSE (AppID, Length, Reserved), puis des données utiles, et se termine par une séquence de contrôle de trame (FCS). Chaque sous-section de la charge utile (**APDU**[1]) est

1. Un APDU (Application Protocol Data Unit) est un format de message utilisé pour la communication entre deux entités dans un réseau, souvent entre un client et un serveur ou entre des équipements intelligents. Dans le protocole GOOSE l'APDU est utilisé pour transporter les messages d'événements critiques entre les équipements d'une sous-station (comme les relais de protection, les automates, etc.).

formatée selon le schéma Type-Length-Value (TLV), ce qui impose une interprétation dynamique des champs pour assurer une réception et un traitement corrects.

Pour mieux visualiser cette trame GOOSE voici un exemple :

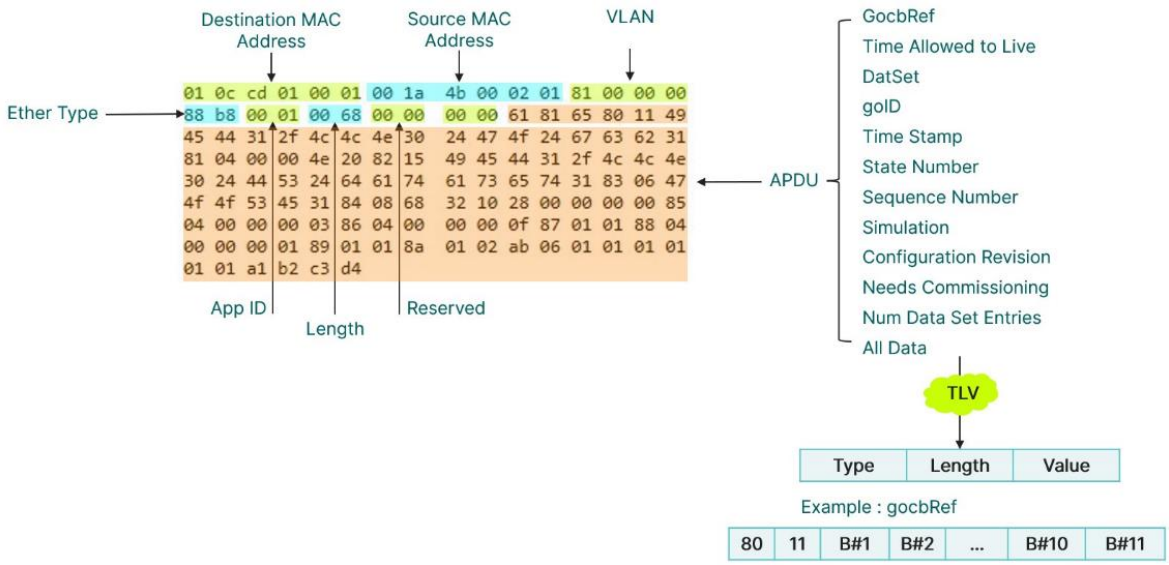


Figure 2.2 Example de trame GOOSE

3 Présentation du projet

3.1 Contexte du stage

Le stage s'inscrit au sein de l'équipe R&D de GE Vernova Grid Solutions à Montpellier, spécialisée dans le développement de calculateurs de tranche pour postes électriques conventionnels et digitaux. Ces calculateurs assurent des fonctions de contrôle-commande, de protection et de supervision du réseau électrique.

3.2 Objectif du projet

L'objectif du projet est de développer un module de monitoring ciblant un composant clé du système : un module FPGA chargé d'interpréter les trames GOOSE. Ce module est intégré dans la carte CPU du calculateur de tranche.

4 Architecture matérielle ciblée

4.1 Description du calculateur de tranche

Figure 4.1 MiCOM C264 Calculateur de Tranche

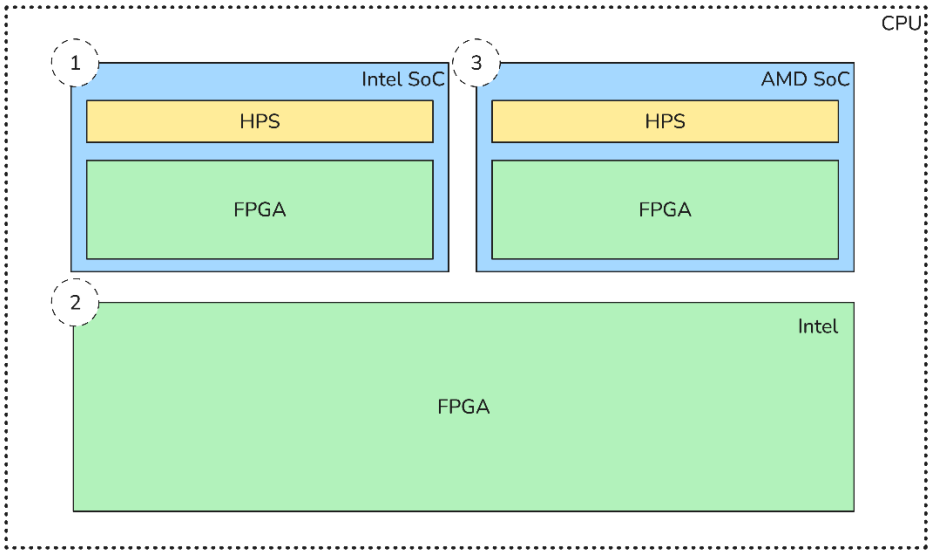
Le calculateur de tranche est un dispositif embarqué destiné à assurer en temps réel les fonctions de contrôle-commande, de protection et de surveillance de la distribution du réseau électrique. Il joue un rôle essentiel dans la régulation du réseau, en permettant de prévenir les surcharges, les pannes et d'assurer la continuité de service.

Ce système est composé de plusieurs cartes électroniques, chacune remplissant un rôle spécifique :

- **Carte d'alimentation** : assure la distribution électrique interne du calculateur.
- **Carte CPU** : cœur du système, elle intègre un ou plusieurs FPGA pour le traitement des données critiques.
- **Interface graphique utilisateur (HMI)** : permet la visualisation et la configuration des paramètres du système.
- **Cartes additionnelles** : selon les besoins, elles peuvent inclure des interfaces de communication, des modules d'entrées/sorties, etc.

Dans le cadre de ce projet, l'attention est portée spécifiquement sur la **carte CPU**, qui constitue la plateforme de développement. Elle intègre plusieurs composants critiques, notamment le **FPGA Intel Cyclone V SX**, qui héberge les modules de traitement des trames **GOOSE**. L'analyse débutera par une présentation de l'**architecture matérielle** de cette carte, avant de se concentrer sur les **composants logiques** qui y sont implémentés.

4.2 Composition de la carte CPU



HPS[1] : Hard Processor System, SoC : System on Chip, PS : Processor System

Figure 4.2 Architecture simplifiée de la carte CPU, sans représentation des bus de communication.

La carte CPU embarque trois puces, chacun jouant un rôle spécifique :

1. **Intel SoC** : Cœur du projet. Contient toute l'intelligence métier, notamment le module d'interprétation des trames GOOSE.
2. **Intel FPGA** : Contribue à la gestion des communications réseau au sein du système.
3. **AMD SoC** : Héberge un environnement logiciel dédié à la connectivité et à la sécurité.

4.3 Cible du projet

Le développement s'est concentré sur le FPGA Intel SoC, et plus précisément sur le module d'interprétation GOOSE. Ce module a pour rôle de traiter les trames reçues via Ethernet, en extrayant les informations critiques nécessaires aux fonctions de protection et de commande. L'objectif principal est de surveiller et analyser son comportement afin d'améliorer la fiabilité et la traçabilité du système.

1. Le Hard Processor System est une unité de traitement intégrée dans une puce FPGA, généralement basée sur un processeur ARM. Contrairement à la logique programmable du FPGA, le HPS est fixe (non reconfigurable) et offre des performances élevées pour les tâches de calcul ou de gestion système.

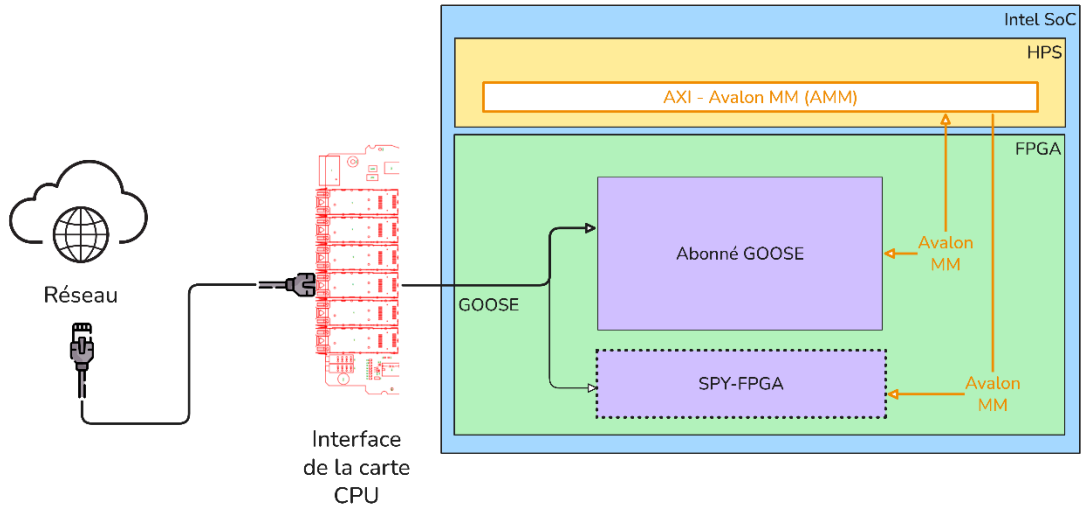


Figure 4.3 Architecture simplifiée du Cyclone V SX avec intégration théorique du module SPY FPGA.

La figure ci-dessus présente une vue simplifiée de l'architecture du FPGA Intel Cyclone V SX, illustrant l'implémentation théorique du module de monitoring SPY FPGA.

Les protocoles utilisés dans cette architecture sont les suivants :

- **AXI (Advanced eXtensible Interface) :** protocole d'interconnexion haute performance développé par ARM, utilisé pour la communication interne entre les composants du système sur puce (SoC).
- **Avalon Mapped Memory :** protocole simplifié développé par Intel, dérivé de l'AXI, facilitant les échanges entre le FPGA et le HPS (Hard Processor System).
- **Avalon Streaming (Avalon ST) :** protocole de communication développé par Intel pour les systèmes FPGA, permettant le transfert continu de flux de données entre blocs IP.
- **IEC 61850-GOOSE (Generic Object Oriented Substation Event) :** protocole de communication normalisé pour les systèmes électriques intelligents, permettant l'échange rapide et fiable de messages critiques entre les équipements de protection.

5 Flux de Conception Assisté par Ordinateur pour FPGA

Le flux de conception FPGA comprend les étapes suivantes :

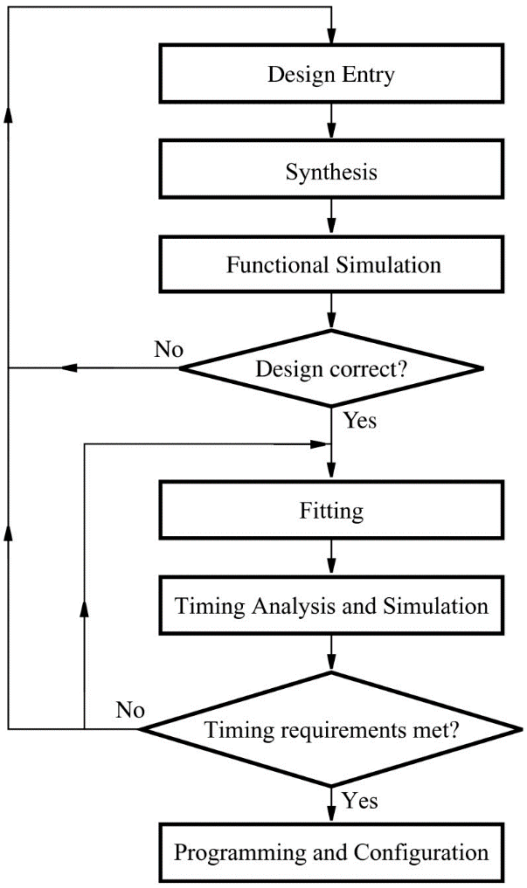


Figure 5.1 illustre ce flux de conception utilisé dans les outils de CAO pour FPGA.

- **Saisie du design (Design Entry)** : modélisation du système à l'aide d'un langage de description matériel (VHDL ou Verilog).
- **Synthèse** : transformation du code HDL en un netlist logique compatible avec l'architecture cible du FPGA.
- **Simulation fonctionnelle** : vérification du comportement logique du design avant l'implémentation physique.
- **Placement et routage (Place & Route)** : attribution des ressources physiques du FPGA aux éléments logiques du design.
- **Analyse temporelle (Timing Analysis)** : validation des contraintes temporelles pour garantir le bon fonctionnement en temps réel.

- **Programmation et configuration** : génération du fichier binaire et téléchargement sur la cible FPGA.

Dans le cadre du projet SPY-FPGA, ce flux a été respecté afin d'assurer une intégration fiable de la solution de monitoring au sein du système existant.

6 Cahier des charges

6.1 Objectif général

Développer une solution de monitoring embarquée dans un FPGA, permettant l'acquisition, le filtrage, le stockage et l'accessibilité des données internes, tout en respectant les contraintes de performance en temps réel et de compacité.

6.2 Exigences fonctionnelles obligatoires (Necessary)

Exigence	Description
Compacité du design	Le design doit minimiser l'utilisation des ressources logiques disponibles, en limitant chaque module à environ 1 % des **ALMs**. Étant donné que le système est déjà fortement chargé en fonctionnalités, il est essentiel d'éviter toute surcharge supplémentaire.
Acquisition et stockage en temps réel	Le système doit assurer une acquisition et un enregistrement des données en temps réel, sans aucune perte, tout en garantissant la non-altération du fonctionnement actuel.
Stockage interne	Les données doivent être stockées dans la mémoire interne du FPGA (B-RAM).
Accessibilité via bus AXI-Avalon MM	Les données stockées doivent être accessibles via le bus mémoire-mappé AXI-Avalon.
Compatibilité protocolaire	Le système doit être compatible avec les protocoles suivants : IEC61850 GOOSE, Avalon MM, Avalon ST.
Configurable via mémoire	La configuration du système (paramètres de capture, filtrage, etc.) doit être possible via la B-RAM, accessible par le bus AXI-Avalon MM.

6.3 Exigences fonctionnelles souhaitables (facultatif)

Exigence	Description
Filtrage des données	Le système devrait permettre de sélectionner quelles parties des données acquises doivent être stockées.
Flexibilité du stockage	Le système devrait offrir une flexibilité sur la quantité de données à stocker.

7 Développement du POC (Proof Of Concept) GOOSE-SUB

Le module SPY-FPGA a été conçu dans le but de surveiller, analyser et filtrer les trames GOOSE échangées au sein d'un système embarqué conforme à la norme IEC 61850. Son objectif principal est de renforcer la traçabilité et la fiabilité des communications critiques dans un environnement industriel, tout en respectant des contraintes strictes de compacité, de configurabilité et de faible consommation de ressources.

L'architecture modulaire développée dans ce chapitre répond précisément à cette ambition : chaque sous-module a été pensé pour remplir une fonction spécifique (conversion, gestion, filtrage, statistiques, stockage), tout en s'intégrant harmonieusement dans un flux de traitement optimisé. L'ensemble forme une solution légère, adaptable et facilement intégrable dans les calculateurs de tranche utilisés dans les postes électriques.

7.1 Résumé :

e module SPY-FPGA a été conçu comme une solution de monitoring embarquée, optimisée pour le traitement des trames GOOSE selon la norme IEC 61850. Il repose sur une architecture modulaire composée de six entités principales :

- **L'interface de conversion :** transforme les trames du format IEC 61850 vers le protocole Avalon Streaming, facilitant leur traitement interne.
- **Le gestionnaire :** pilote la réception des trames via une machine à états, assurant leur séquencement et leur conformité structurelle.
- **Le filtre :** compare les données reçues aux paramètres de configuration et déclenche des actions en cas de non-conformité.
- **Le module de statistiques :** comptabilise les trames valides et les erreurs détectées.
- **Le système de stockage :** enregistre uniquement les trames conformes, avec une gestion intelligente de la mémoire.
- **Le bloc de configuration :** centralise les paramètres de filtrage et de validation utilisés par les autres modules.

L'implémentation en VHDL a été guidée par une exigence de compacité et de performance. Les résultats de synthèse (tableau 7.5) confirment une occupation inférieure à 1 % des ressources logiques et mémoire du FPGA, validant ainsi la légèreté, la configurabilité et la facilité d'intégration du module dans un environnement industriel.

7.2 Architecture

Description de l’architecture du système

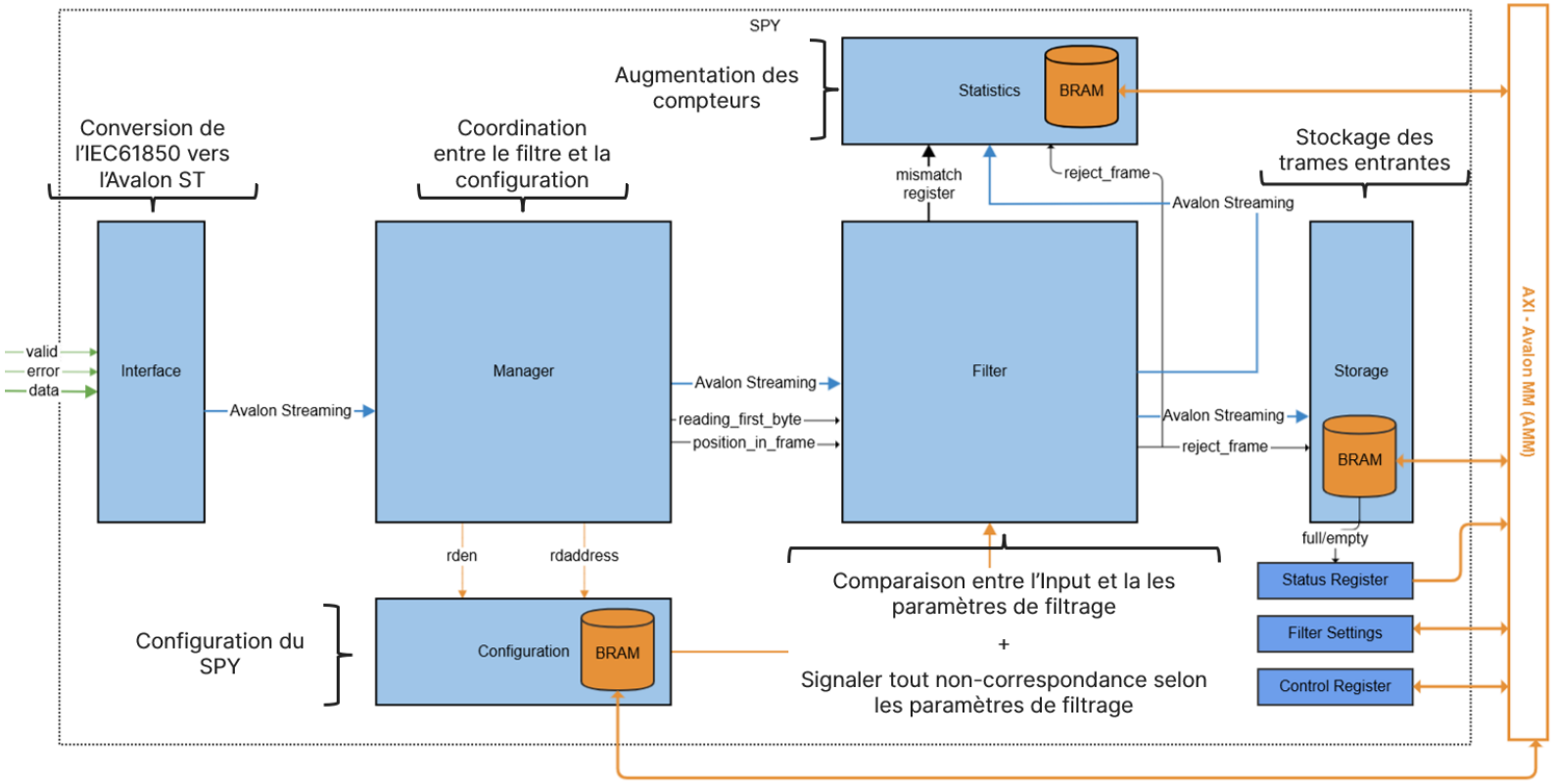


Figure 7.1 illustration de l’architecture de SPY-FPGA

Le système est composé de six entités principales, chacune jouant un rôle spécifique dans le traitement, la validation et le stockage des trames GOOSE selon la norme IEC 61850. Ces entités interagissent de manière coordonnée afin d'assurer la conformité des trames reçues avec la configuration attendue.

7.2.1 Module 1 : Interface (Spécifique aux protocoles IEC-61850)

- **Rôle :** Point d’entrée du flux de données dans le système.
- **Fonction :** Convertit les trames GOOSE reçues au format IEC 61850 vers le protocole Avalon Streaming, facilitant ainsi leur traitement par les modules internes.

7.2.2 Module 2 : Gestionnaire (Spécifique au protocole IEC-61850 GOOSE)

- **Rôle :** Ordonne et synchronise les trames entrantes.
- **Fonctionnalités :**
  - Lit le flux de données converti.
  - Utilise une machine à états inspirée de la structure des trames GOOSE pour assurer leur séquencement correct.
  - Synchronise les trames transmis au Filtre avec le bloc de Configuration afin de garantir leur conformité structurelle.

7.2.3 Module 3 : Configuration (Générique)

- **Rôle :** Référence de configuration pour l'interprétation des trames.
- **Fonction :**
  - Stocke les paramètres définissant les règles d'interprétation et de filtrage des trames.
  - Sert de base de comparaison pour les modules de validation.

7.2.4 Module 4 : Filtre (Spécifique au protocole IEC-61850 GOOSE)

- **Rôle :** Valide les trames et signale les anomalies.
- **Fonctionnalités :**
  - Compare les trames entrantes avec les paramètres de la Configuration.
  - Déclenche un signal de rejet si une trame est malformée ou hors séquence.
  - Déclenche un signal de non-conformité si la trame est valide mais ne correspond pas à la configuration attendue.

7.2.5 Module 5 : Statistiques (Générique)

- **Rôle :** génération des statistiques à partir des données reçues.
- **Fonctionnalités :**
  - A la fin de chaque trame conforme reçue augmentation des compteurs situé dans une BRAM en fonction du signal de non-conformité.

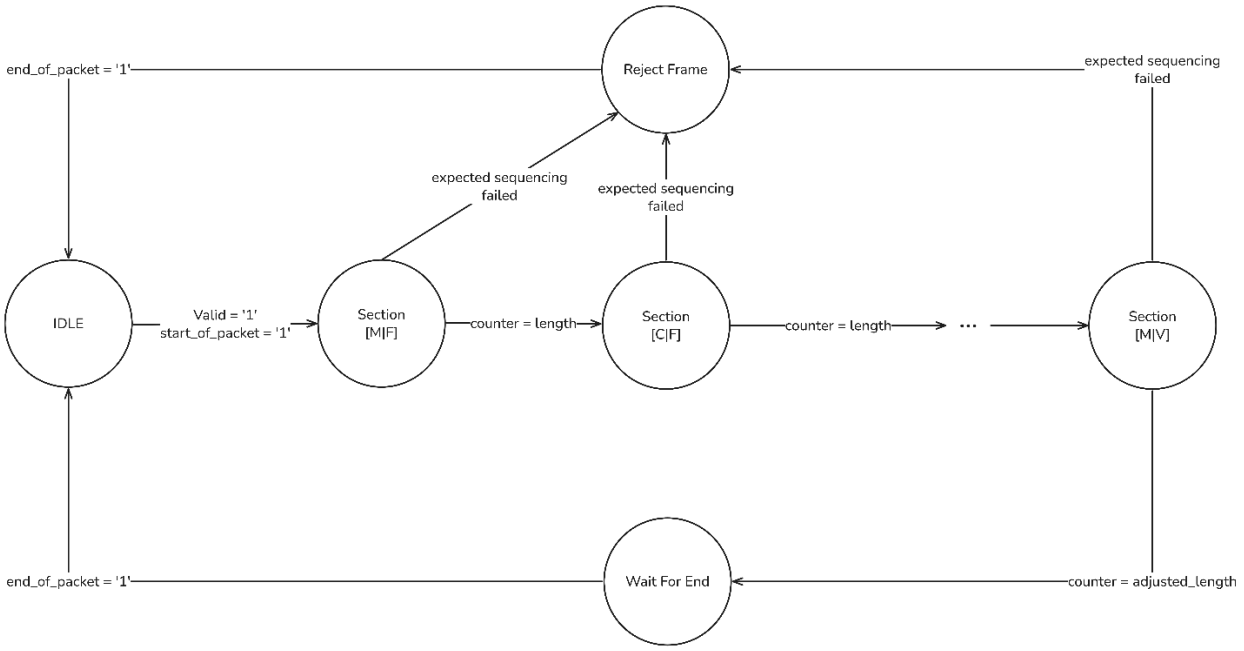
7.2.6 Module 6 **:** Stockage (Générique)

- **Rôle :** Conserve uniquement les trames valides pour une analyse ou un traitement ultérieur.
- **Fonctionnalités :**
  - Enregistre les trames acceptées.
  - En cas de rejet, l'adresse d'écriture est réinitialisée au début de la trame courante, permettant son écrasement par la suivante.
  - Lorsque la capacité de stockage est atteinte, le système cesse d'accepter de nouvelles trames, sauf si un signal de réinitialisation est émis.

7.3 Implémentation du POC (Proof Of Concept)

Cette section décrit les éléments essentiels du code VHDL utilisés pour la gestion du système (**Manager**) ainsi que pour le filtrage des trames **GOOSE** (**Filter**).

7.3.1 Gestionnaire (Manager)



Section	Présence	Taille
Section de la trame GOOSE	M : Obligatoire (Must Have)	V : Taille variable
	C : Peut être présente (Can Have)	F : Taille fixe

Figure 7.2 Représentation simplifié de la machine d'états du module gestionnaire.

Le module Gestionnaire joue un rôle central dans le traitement des trames GOOSE. Il assure l'ordonnancement, la synchronisation et le séquencement correct des données entrantes, en s'appuyant sur une machine à états inspirée de la structure normalisée des trames GOOSE (IEC 61850).

Grâce à ses fonctionnalités avancées, le gestionnaire :

- Lit le flux de données converti en provenance du réseau.
- Identifie et suit la position exacte dans la trame à l'aide d'un système d'états.
- Synchronise les trames avec le bloc de Configuration, garantissant ainsi leur conformité structurelle avant transmission au module Filtre.

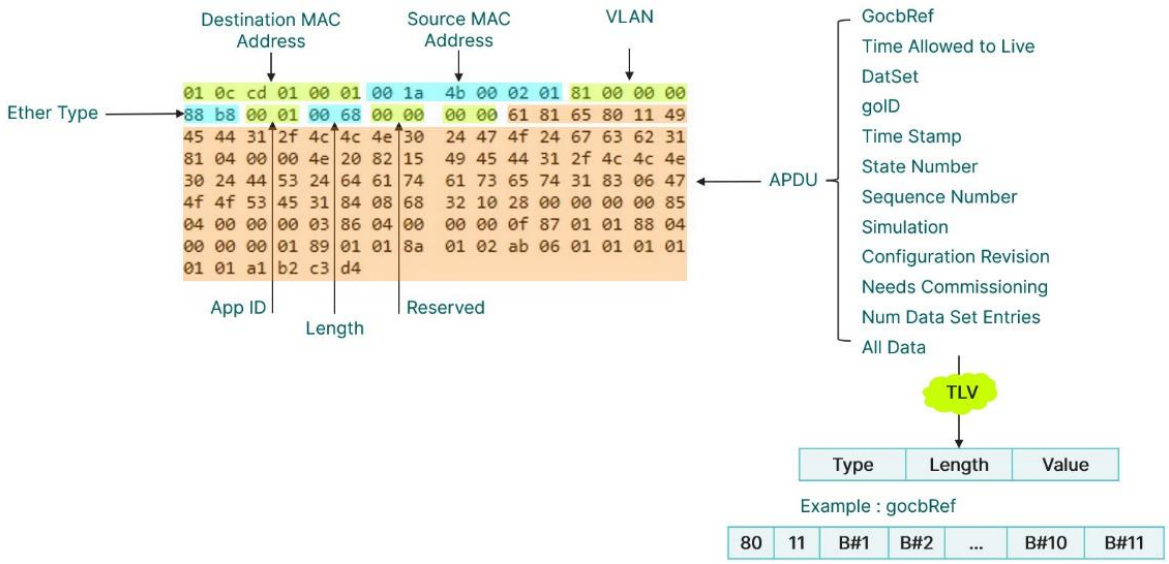


Figure 7.3 Example de trame GOOSE

Chaque section de la trame GOOSE (voir figure 7.3), possède son état dédié dans le gestionnaire. À chaque état, le système décompte le nombre d'octets entrants jusqu'à atteindre zéro, ce qui déclenche le passage à l'état suivant.

Dans les sections où la taille doit être interprétée dynamiquement (TLV), le système initialise un compteur à 2. Une fois ce compteur arrivé à zéro, le champ décrivant la longueur est interprété, puis ajouté au compteur avec un offset de 2. Lorsque le compteur atteint la valeur 3, cela indique la fin de la section, et le gestionnaire passe à l'état suivant.

Par ailleurs, dans chaque état, le gestionnaire associe une adresse de configuration à la valeur correspondante de la section, permettant une comparaison efficace au niveau du filtre. En complément de cette coordination entre les paramètres de configuration et les données d'entrée, le gestionnaire émet un signal d'identification, position_in_frame, indiquant la position actuelle dans la trame.

Par exemple, une valeur de 0x00200 signifie que le filtre est en train de recevoir les octets correspondant au champ GOCBREF. Ce mécanisme permet un traitement séquentiel et structuré des différentes sections de la trame.

Dans l'état « Reject Frame », le gestionnaire informe le filtre que la trame a été rejetée en utilisant le signal d'identification position_in_frame, auquel il attribue la valeur 0xFFFFF.

En revanche, si le gestionnaire parvient à séquencer correctement la réception de la trame, le système passe à l'état « Wait for End ». Dans cet état, les données continuent d'être transmises sans que les champs spécifiques du GOOSE ne soient pris en compte. Le système attend simplement la fin du paquet pour passer à la trame suivante.

7.3.2 Filtre

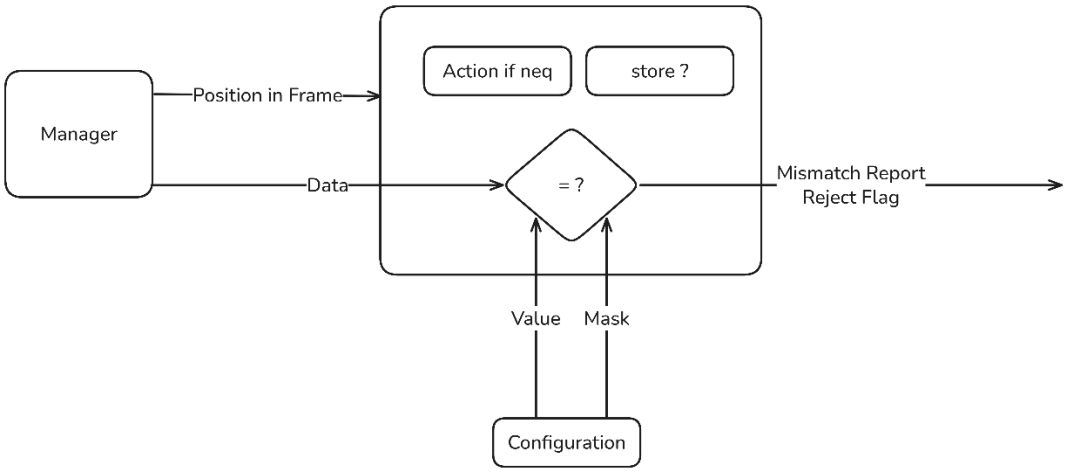


Figure 7.3 Représentation simplifié du fonctionnement du système de filtrage du SPY-FPGA.

Le filtre reçoit, de la part du gestionnaire, les données ainsi que leur position dans la trame GOOSE. En parallèle, il reçoit une valeur et un masque provenant de la configuration.

Le filtre applique le masque à la valeur reçue et aux données de la trame. Si les deux résultats ne correspondent pas, cela indique un problème potentiel au niveau de la trame.

Avant le lancement du monitoring en temps réel sur la carte FPGA, l'utilisateur peut configurer plusieurs options selon ses besoins spécifiques :

Option	Description
Ignorer	Ignorer complètement le mismatch (une non-correspondance entre les données entrante et la configuration)
Signaler	Signaler le mismatch
Rejeter	Rejeter la trame (ne pas la stocker)
Rejeter et Signaler	Rejeter la trame et signaler le mismatch

Table 7.1 Structure du registre de control

De plus, l'utilisateur peut choisir de stocker ou non cette partie de la trame concernée.

Toutes ces décisions sont prises via des registres de paramétrage et de contrôle, accessibles par le biais de la mémoire mappée Avalon (Avalon Mapped Memory).

7.3.2.1 Registre de contrôle

Bit 31	Bit 30	…	Bit 3	Bit 2	Bit 1	Bit 0
/	/	…	/	/	Preconfigure	Start/End

Table 7.2 Structure du registre de control

- Start/End : Bit de commande permettant d'activer ou de désactiver le module espion (SPY-FPGA).
- Preconfigure : Bit de commande utilisé pour charger les paramètres de filtrage dans le filtre, selon la configuration définie.

7.3.2.2 Paramètres de Filtrage

Chaque section ci-dessous représente une partie spécifique des données d'entrée de la trame GOOSE. Par exemple, une section peut correspondre au champ GOCBREF, à l'adresse MAC de destination, ou à tout autre champ structurant de la trame GOOSE.

Pour chaque section des données d'entrée, les options suivantes sont disponibles :

Paramètres de Stockage

Bit 31	Bit 30	…	Bit 1	Bit 0
Section#31	Section#30	…	MAC Source	MAC Destination

Table 7.3 Structure du registre de paramétrage pour le filtre (stockage)

- '0' : Ne pas stocker la trame.
- '1' : Stocker la trame.

Paramètres de Filtrage

Bit 63-62	Bit 61-60	…	Bit 3-2	Bit 1-0
Section#31	Section#30	…	MAC Source	MAC Destination

Table 7.4 Structure du registre de paramétrage pour le filtre ()

- '00' : Ignorer le mismatch détecté.
- '01' : Signaler le mismatch sans rejeter la trame.
- '10' : Rejeter la trame en cas de mismatch.
- '11' : Rejeter la trame et signaler le mismatch.

7.4 Synthèse

La synthèse du code VHDL a été réalisée à l'aide de l'outil Quartus Prime (voir Annexe A pour plus de détails). Le rapport généré fournit une estimation des ressources utilisées, permettant de vérifier la conformité du design avec les spécifications du cahier des charges.

Ressource	Occupation
Logique utilisée (in ALMs)	625 / 113,560 **(< 1 %)**
Mémoire utilisée	12,288 / 12,492,800 **(< 1 %)**

Tableau 7.5 Résultats de la Synthèse.

Le rapport de synthèse (voir tableau 7.5) met en évidence une utilisation inférieure à 1 % des ressources critiques du FPGA, tant en logique qu'en mémoire. Ce résultat valide la compacité du système, un critère essentiel avant son intégration dans l'architecture cible.

7.5 Conclusion Intermédiaire

Après avoir conçu et implémenté le module SPY-FPGA selon une architecture modulaire et optimisée, il est essentiel de valider son bon fonctionnement dans un environnement représentatif. La prochaine étape consiste donc à simuler le système, à vérifier sa conformité aux spécifications, et à évaluer sa robustesse face à différents scénarios de trames GOOSE.

Le chapitre suivant présente la méthodologie de simulation automatisée mise en place, les outils utilisés, ainsi que les résultats obtenus. Cette phase est déterminante pour garantir la fiabilité du module avant son intégration dans un système industriel réel.

8 Simulation et Vérification

Dans le cadre du développement de systèmes embarqués critiques, la conception matérielle ne constitue qu'une première étape. Il est impératif de démontrer que le système fonctionne conformément aux spécifications, qu'il est fiable, robuste et capable de réagir correctement dans des conditions réalistes. C'est précisément l'objectif de la phase de simulation et de vérification.

8.1 Résumé

Cette section décrit la mise en place d'une méthode de simulation automatisée pour tester le module SPY-FPGA. L'approche repose sur trois scripts Python principaux :

- Générateur d'input SPY : transforme des trames réseau (PCAP) en fichiers exploitables par le test-bench VHDL.
- Générateur de test-bench : crée un environnement de simulation adapté à l'entité VHDL.
- Générateur de rapport : analyse les résultats de simulation et produit un rapport lisible.

Le tout est orchestré via un Makefile, outil pour séquencer et automatiser la compilation (voir l'Annexe A pour plus de détails), permettant de lancer des scénarios de test complets sans intervention manuelle. L'utilisateur peut configurer les filtres, injecter des trames, arrêter le module, et récupérer les statistiques automatiquement.

Les résultats sont ensuite validés à l'aide de Wireshark (analyse réseau, voir Annexe A pour plus de détails), confirmant la détection correcte des trames GOOSE et des anomalies (mismatches). Cette méthode garantit une vérification fiable, rapide et reproductible du module SPY-FPGA dans des conditions proches du réel.

8.2 Méthodologie de Simulation Automatisée

Afin de simuler mon système et de vérifier son bon fonctionnement, j'ai adopté une approche automatisée visant à gagner du temps et à faciliter les tests sur le long terme. Cette méthode repose sur l'utilisation de Python pour générer automatiquement les fichiers d'entrée nécessaires à la simulation dans ModelSim, un outil de simulation et de vérification (voir Annexe A pour plus de détails).

8.2.1 Fonctionnement du point de vue de l'utilisateur

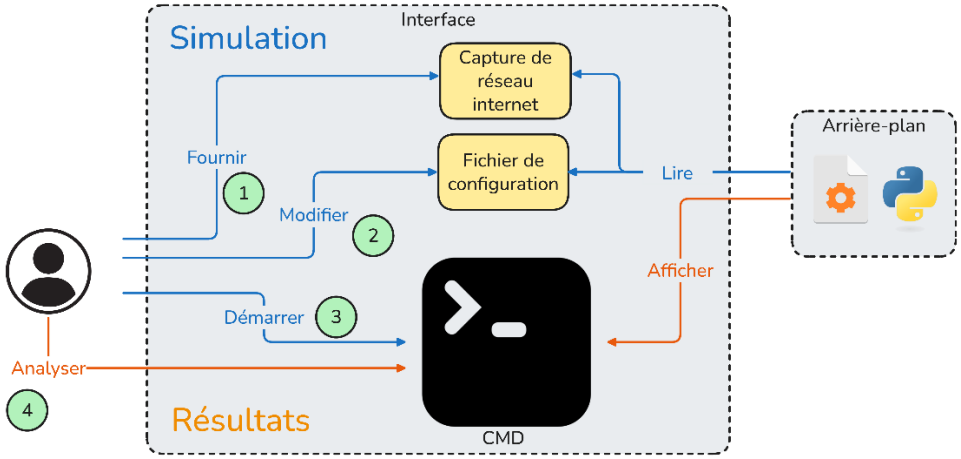


Figure 8.1 Illustration simplifiée du système d'automatisation de la simulation et vérification des modules en VHDL (Interface)

La méthode se compose de deux éléments principaux, illustrés dans la Figure 8.1 :

- Interface utilisateur (partie gauche de la Figure 8.1) :

Cette interface constitue la partie visible et accessible par l'utilisateur. Elle permet de configurer et de lancer la simulation, puis d'afficher les résultats obtenus. Conçue pour être intuitive et facile à prendre en main, elle simplifie considérablement le processus de test en masquant la complexité technique sous-jacente.

L'utilisateur n'a qu'à suivre quatre étapes principales (numérotées en vert dans la Figure 8.1) :

1. Fournir une capture du réseau pour simuler la partie GOOSE
2. Modifier le fichier de configuration du système
3. Lancer la simulation via une ligne de commande ('make simulation')
4. Analyser les résultats affichés dans la console

- Moteur automatisé (partie droite de la Figure 8.1) :

Fonctionnant en arrière-plan, cette partie prend en charge l'ensemble des opérations techniques. Elle interprète les fichiers fournis par l'utilisateur, génère automatiquement les fichiers nécessaires à la simulation (tels que le test-bench, les entrées), exécute l'outil de simulation via la ligne de commande, puis collecte les résultats. Ces derniers sont ensuite traités et présentés sous forme de rapport clair et structuré, facilitant l'analyse et la validation du système.

8.2.2 Fonctionnement de l'arrière-plan

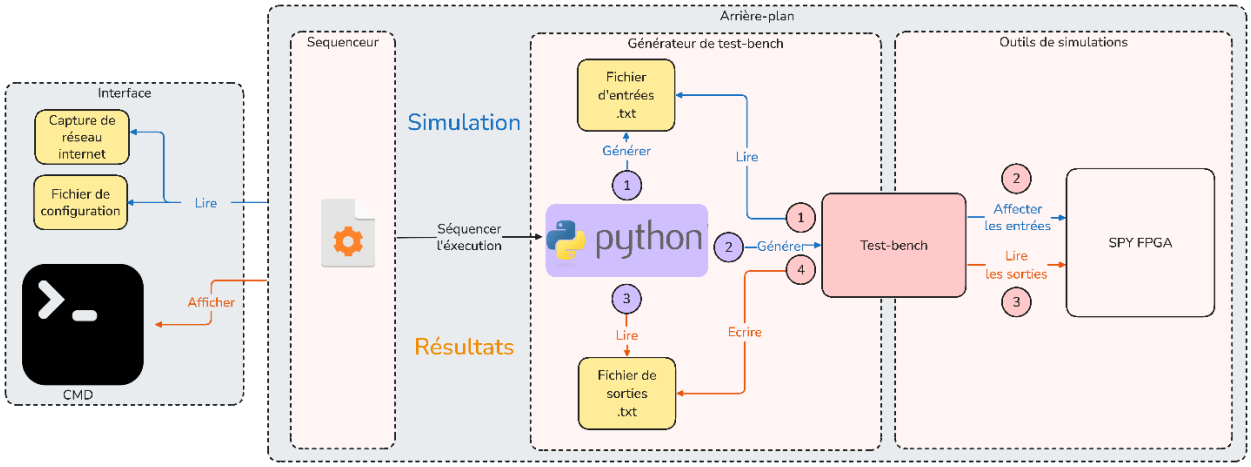


Figure 8.2 Illustration simplifiée du système d'automatisation de la simulation et vérification des modules en VHDL (arrière-plan)

Les scripts Python jouent un rôle essentiel dans l'interprétation et la génération des fichiers nécessaires à la simulation et à la vérification du module SPY FPGA. Trois scripts principaux assurent le bon fonctionnement de cette chaîne (numérotées en violet dans la Figure 8.2) :

1. Génération du fichier d'entrées.
2. Génération du test-bench.
3. Génération du rapport de simulation.

8.2.2.1 Générateur d'input SPY (en violet)

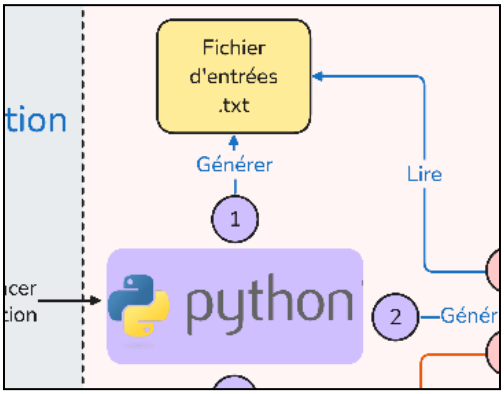


Figure 8.5 Illustration du système d'automatisation (génération du fichier d'entrées)

Ce script utilise un fichier PCAP (contenant des trames réseau capturées) pour générer un fichier texte. Ce fichier est ensuite interprété par le code VHDL afin de simuler le comportement du module SPY.

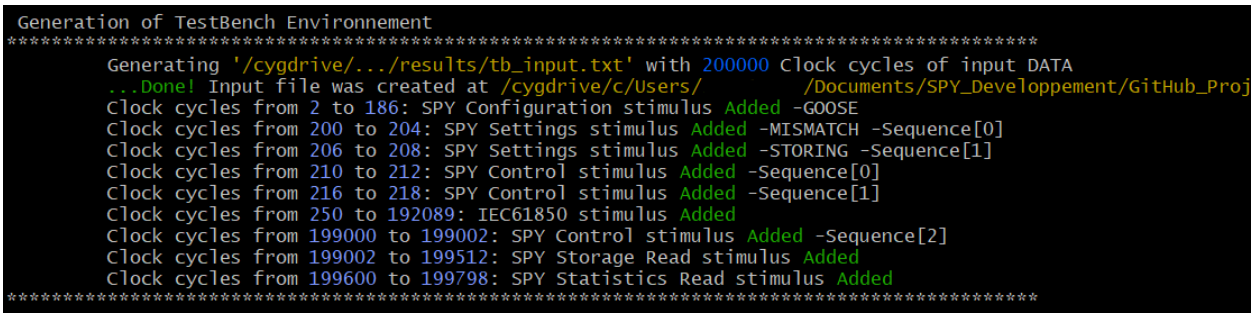


Figure 8.3 Capture du Shell Cygwin lors de la génération du fichier d'entrée.

La Figure 8.3 illustre le processus de génération du fichier d'entrées destiné au système FPGA, en suivant une chronologie précise basée sur les cycles d'horloge :

- **Injection de la configuration initiale (cycle 2 → cycle 186) :** Les paramètres de base nécessaires au fonctionnement du système sont injectés, notamment ceux liés au filtrage et à la configuration du protocole GOOSE.
- **Configuration du filtrage et du stockage (cycle 200 → cycle 212) :** Les paramètres spécifiques au filtrage (sections du message GOOSE à surveiller ou ignorer) ainsi que ceux relatifs au stockage sont injectés dans le fichier d'entrées. Ces données seront utilisées par le module SPY pour effectuer la capture et l'analyse.
- **Démarrage du module SPY (cycle 216 → cycle 218) :** Une commande de démarrage est insérée dans le fichier d'entrées, déclenchant l'activation du module SPY après la configuration.
- **Injection des paquets réseau (cycle 250 → cycle 192,089)** : Une fois le module SPY opérationnel, les paquets réseau extraits et filtrés à partir d'un fichier PCAP sont injectés dans le système. Cette étape simule un trafic réseau réel pour tester le comportement du module.
- **Arrêt du module SPY (cycle 199,000 → cycle 199,002) :** Une commande d'arrêt est injectée pour mettre fin à la session de capture du module SPY, une fois tous les paquets transmis.
- **Lecture des données stockées et des statistiques (cycle 199,002 → cycle 199,798) :** Le système procède à la récupération des données enregistrées ainsi qu'aux statistiques de traitement. Ces informations permettent une analyse approfondie du comportement du module SPY et de la validité du filtrage.

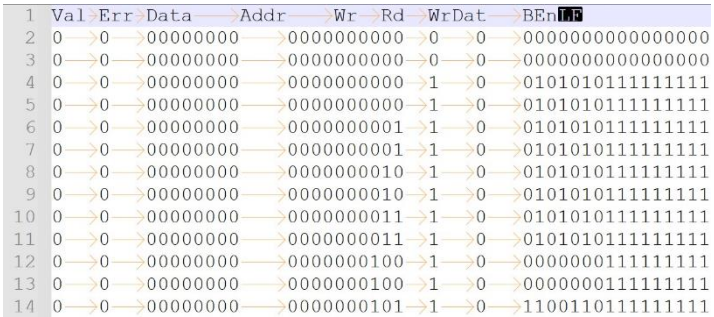


Figure 8.4 Capture d'écran des premières lignes du fichier d'entrée généré automatiquement.

La figure 8.4 montre le résultat de cette génération de fichier d'entrées. Elle représente sous format binaire les différentes valeurs qui seront affectés au module SPY. Chaque ligne représente un cycle d'horloge.

8.2.2.2 Générateur de test-bench (en violet)

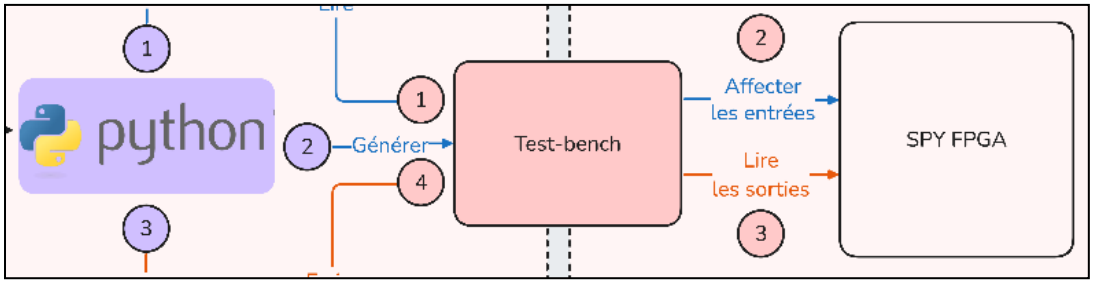


Figure 8.5 Illustration du système d'automatisation (génération du test-bench)

À partir de l'entité VHDL développée, ce script génère un banc de test (test-bench) en VHDL capable de lire les entrées depuis un fichier texte et d'écrire les sorties dans un autre fichier texte. Cela permet de simuler le module dans un environnement contrôlé et reproductible.

La simulation suit les étapes suivantes (numérotées en rouge clair dans la Figure 8.2) :

1. **Lecture des entrées** : Le script lit les valeurs d'entrée depuis un fichier texte préalablement défini.
2. **Injection des entrées** : Ces valeurs sont ensuite appliquées au module FPGA via un outil de simulation (quel que soit l'outil utilisé).
3. **Observation des sorties** : Les sorties du module sont surveillées en temps réel pendant la simulation.
4. **Écriture des résultats** : Les valeurs de sortie sont enregistrées dans un fichier texte, permettant une analyse post-simulation, voir la figure 8.6.

8.2.2.3 Générateur de rapport SPY (en violet)

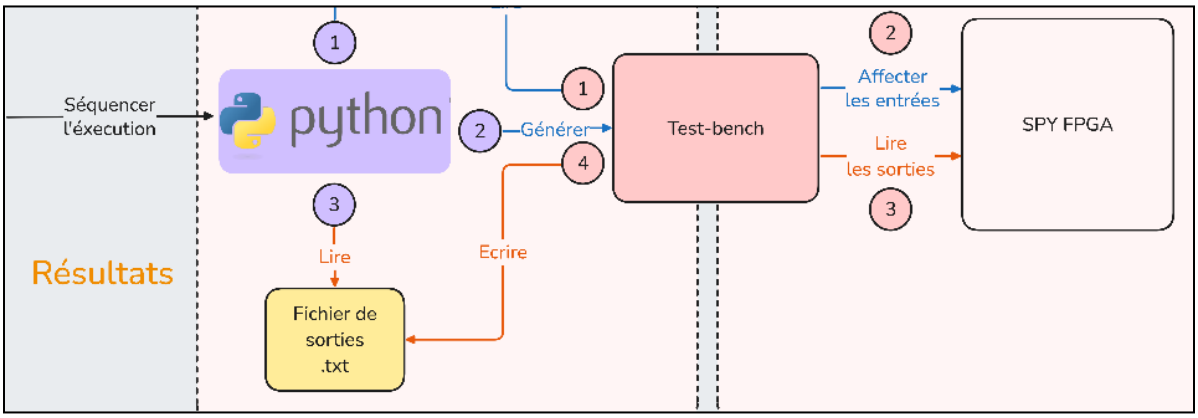


Figure 8.6 Illustration du système d'automatisation (génération du rapport de simulation)

Ce script analyse les résultats de la simulation, enregistrés dans un fichier texte par le test-bench, pour produire un rapport détaillé sur le comportement du module SPY, facilitant ainsi la vérification et le débogage.

Le rapport détaillé est composé de quatre sections :

1. **Affichage de la configuration GOOSE (Figure 8.7)**

Cette section présente les paramètres initiaux utilisés pour configurer le module SPY-FPGA. Elle inclut les champs spécifiques de la trame GOOSE (comme GOCBREF, MAC, etc.) qui serviront de référence pour le filtrage et la validation. Cela permet de visualiser les critères sur lesquels le module va se baser pour détecter les anomalies.

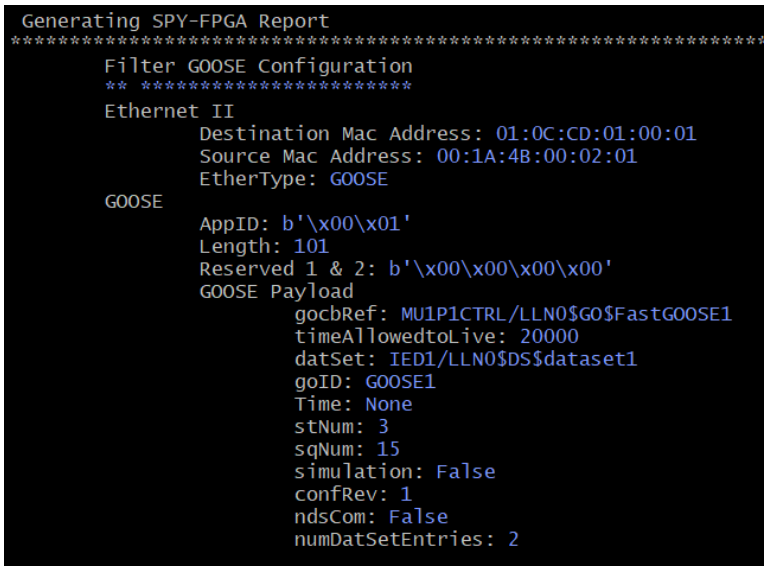


Figure 8.7 Capture du Shell Cygwin lors de la génération du rapport (Configuration).

2. **Affichage des paramètres de filtrage (Figure 8.8)**

Ici sont détaillées les règles appliquées à chaque champ de la trame GOOSE. Pour chaque section, le système indique si une non-conformité doit être ignorée, signalée, rejetée, ou les deux. Ces paramètres permettent de personnaliser le comportement du module face aux erreurs ou incohérences dans les trames reçues.

Le système possède 4 modes de filtrage pour chaque champ de la trame GOOSE :

- Ignorer (en vert) la non-correspondance.
- Signaler (en jaune) la non-correspondance.
- Rejeter (en rouge) la non-correspondance.
- Rejeter et signaler (en magenta) la non-correspondance.

Comme illustré dans la figure 8.8, le système est configuré pour signaler (en jaune) chaque non-correspondance avec le champ GOCBREF. Les autres mismatches sont ignorés (en vert).

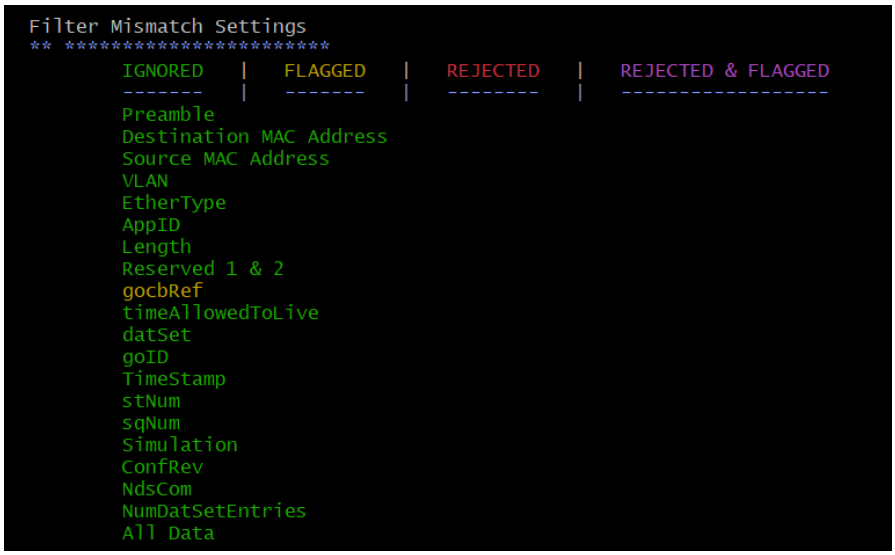


Figure 8.8 Capture du Shell Cygwin lors de la génération du rapport (Mismatch).

3. Affichage des paramètres de stockage (Figure 8.9)

Cette partie montre quelles sections des trames GOOSE seront conservées en mémoire. Chaque champ peut être activé ou désactivé pour le stockage, selon les besoins de l'analyse. Cela permet d'optimiser l'utilisation de la mémoire interne du FPGA en ne conservant que les données pertinentes.

Comme illustré dans la figure 8.9. Le système est paramétré pour stocker uniquement la partie GOCBREF des trames GOOSE entrantes.

Figure 8.9 Capture du Shell Cygwin lors de la génération du rapport (Storing).

4. Affichage du résultat de la simulation (Figure 8.10)

Cette dernière section résume les résultats obtenus après exécution du test. Elle indique le nombre de trames GOOSE détectées, celles qui ont été rejetées ou signalées, et les statistiques associées. Ces résultats permettent de valider le bon fonctionnement du module SPY-FPGA et de confirmer que les paramètres de filtrage et de stockage ont été correctement appliqués.

Les résultats indiquent que 3 trames GOOSE ont été identifiés, dont 2 avec un GOCBREF incorrect.

Figure 8.10 Capture du Shell Cygwin lors de la génération du rapport (Results).

Une vérification via Wireshark permet de confirmer que le fichier d'entrée contient bien trois trames GOOSE et que deux parmi ces GOOSE possède le mauvais GOCBREF.

Analyse des trames envoyées vers le module SPY lors de la simulation (issue du PCAP filtré, figure 8.11) sur WireShark :

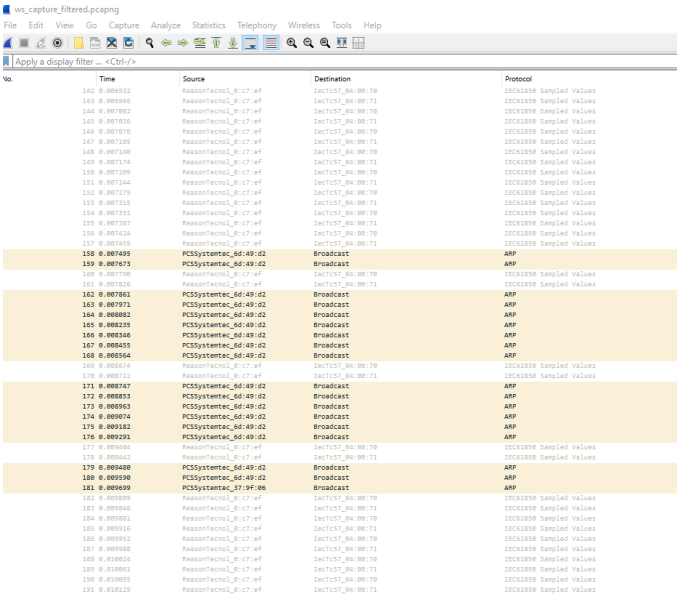


Figure 8.11 Capture de WireShark montrant une section des trames envoyées vers le SPY.

En appliquant le filtre (ws.col.protocol == "GOOSE"), on retrouve ces 3 trames que le module SPY a détectées (voir la figure 8.12).

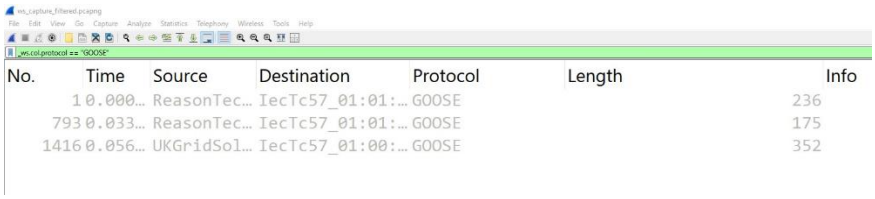


Figure 8.12 Capture de WireShark montrant une section des trames envoyées vers le SPY avec filtrage.

En appliquant le filtre sur les GOOSEs pour exclure ceux avec un GOCBREF égale à MU1P1CTRL/LLN0GOFastGOOSE1, on identifie 2 trames avec un GOCBREF incorrect, confirmant ainsi les résultats observés avec le module SPY (voir la figure 8.13).

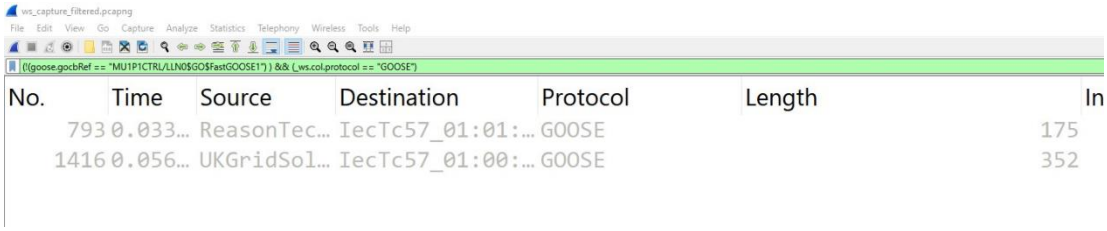


Figure 8.13 Capture de WireShark montrant une section des trames envoyées vers le SPY avec filtrage.

8.3 Conclusion Intermédiaire

L'ensemble du processus de simulation et de vérification du module SPY-FPGA repose sur une chaîne automatisée, robuste et flexible, construite autour de scripts Python, et de l'outil ModelSim. Grâce à cette approche, il est possible de :

- Filtrer et préparer les données d'entrée à partir de fichiers PCAP et JSON,
- Générer automatiquement les fichiers nécessaires à la simulation,
- Exécuter des scénarios de test complets sans intervention manuelle,
- Produire des rapports détaillés facilitant l'analyse et le débogage.

Cette méthode permet non seulement de valider le bon fonctionnement du module SPY dans des conditions réalistes, mais aussi d'assurer une reproductibilité et une scalabilité des tests pour des cas d'usage futurs. Les résultats obtenus, confirmés par des outils comme Wireshark, démontrent la fiabilité du système dans la détection, et l'analyse des trames GOOSE.

Perspectives

Le module SPY-FPGA développé dans le cadre de ce projet constitue une première étape vers une solution de monitoring embarquée, compacte et configurable. Plusieurs axes d'évolution peuvent être envisagés pour enrichir et étendre cette solution :

- **Intégration industrielle**
  - Intégration dans le calculateur de tranche pour une exploitation en conditions réelles.
  - Interopérabilité avec d'autres modules FPGA présents sur la carte Cyclone V SX.
- **Extension fonctionnelle**
  - Support d'autres protocoles normalisés comme IEC 61850 SV (Sampled Values) ou MMS (Manufacturing Message Specification).
  - Amélioration de la configurabilité via une interface logicielle ou une API.
- **Évolutions côté Python**
  - Développement d'une interface graphique (GUI) pour piloter les simulations sans passer par le terminal.
  - Ajout de tests unitaires pour fiabiliser les scripts et faciliter leur maintenance.
- **Évolutions côté HDL**
  - Mise en place de tests de robustesse (fuzzing, trames corrompues) pour renforcer la fiabilité.

Conclusion

Ce stage chez GE Vernova m'a offert une expérience à la fois riche sur le plan technique et formatrice sur le plan humain.

Sur le plan technique, j'ai pu :

- Approfondir mes compétences en conception VHDL et en architecture FPGA.
- Mettre en œuvre une chaîne de simulation automatisée, alliant Python, Makefile et ModelSim.
- Travailler sur des protocoles industriels normalisés (IEC 61850, GOOSE), dans un contexte exigeant en termes de performance et de fiabilité.

Sur le plan relationnel, j'ai eu l'opportunité de :

- Collaborer avec une équipe R&D expérimentée, dans un environnement stimulant et bienveillant.
- Participer à des réunions techniques, échanger avec des experts, et apprendre à structurer mes idées pour les présenter clairement.
- Développer mon autonomie, ma rigueur et ma capacité à m'adapter à des contraintes industrielles réelles.

Ce projet m'a permis de contribuer concrètement à une solution innovante, tout en consolidant les bases nécessaires à une future carrière dans les systèmes embarqués et les architectures numériques. Il m'a également permis de prendre conscience de l'importance de la rigueur, de la méthode et de la collaboration dans un environnement industriel exigeant. Cette expérience a renforcé ma motivation à poursuivre dans le domaine de la conception de systèmes embarqués, avec l'ambition de développer des solutions toujours plus performantes, fiables et adaptées aux enjeux de la transition énergétique.

Annexes

A : Outils de développement

1. Quartus Prime

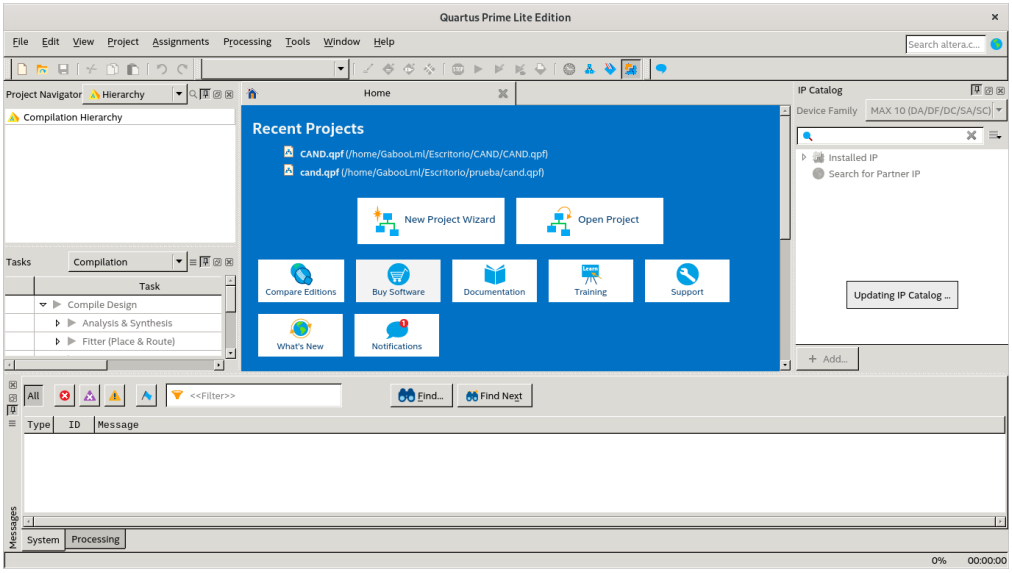


Figure A.1 : Interface graphique de Quartus Prime

Quartus Prime est un logiciel de conception développé par Intel (anciennement Altera) pour la programmation de circuits logiques programmables, tels que les FPGA et CPLD. Il permet aux ingénieurs de créer, simuler, analyser et synthétiser des designs numériques. Quartus Prime offre une interface conviviale, des outils de simulation, de vérification temporelle, ainsi que la prise en charge de langages de description matérielle comme VHDL et Verilog.

2. ModelSim

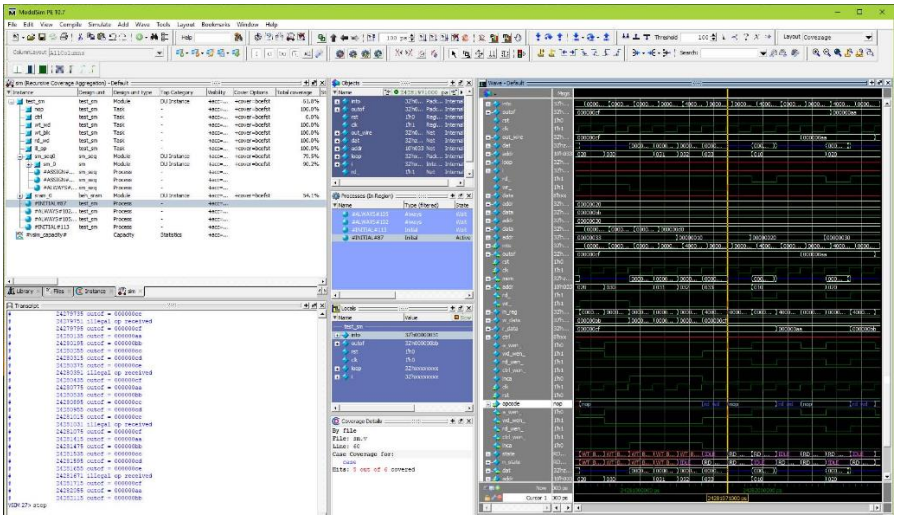


Figure A.1 : Interface graphique de ModelSim

ModelSim est un environnement de simulation développé par Siemens (anciennement Mentor Graphics), utilisé pour la vérification fonctionnelle de circuits numériques. Il prend en charge les langages de description matérielle comme VHDL, Verilog et SystemVerilog. ModelSim permet aux concepteurs de simuler, déboguer et valider leurs conceptions avant la synthèse, ce qui en fait un outil essentiel dans le flux de développement FPGA ou ASIC.

3. Cygwin

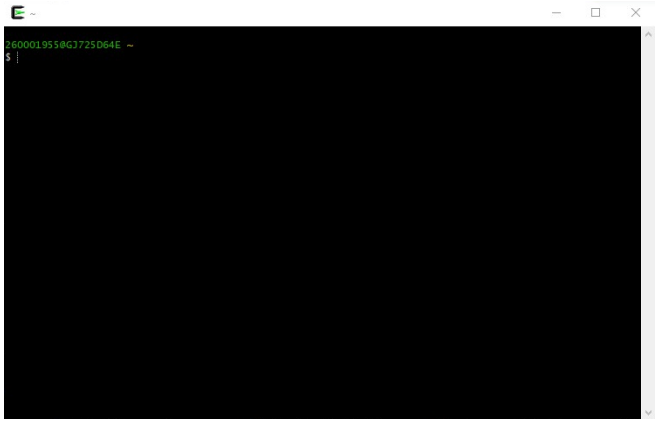


Figure A.3 : Terminal Cygwin.

Cygwin est un environnement logiciel qui permet d'exécuter des applications Linux/Unix sur des systèmes Windows. Il fournit une collection d'outils GNU et une couche d'émulation POSIX, permettant aux utilisateurs de bénéficier d'une interface en ligne de commande semblable à celle d'un système Unix. Cygwin est particulièrement utile pour les développeurs qui souhaitent utiliser des scripts Bash, des utilitaires Unix ou compiler des programmes open source sur Windows.

4. WireShark

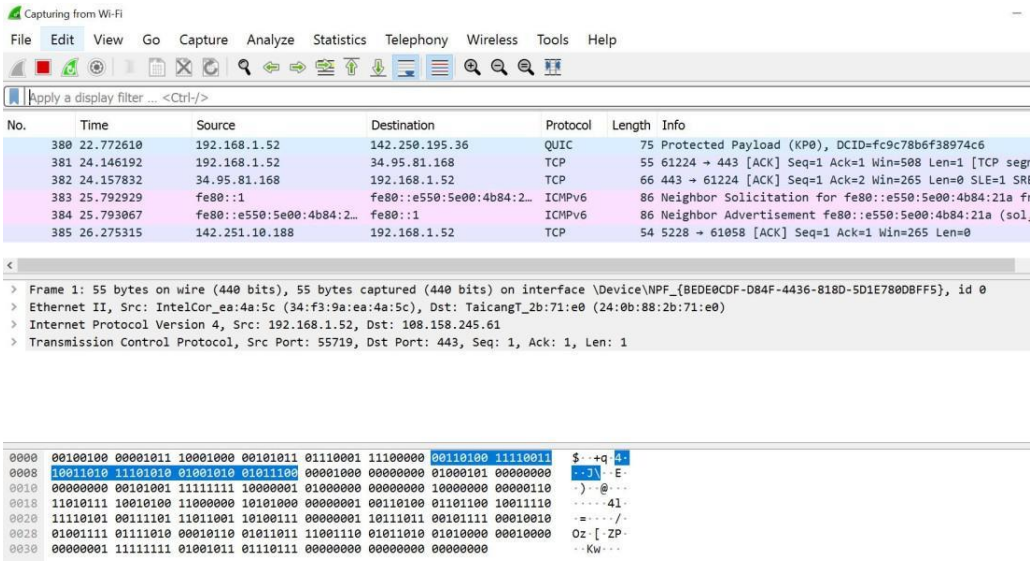


Figure A.4 : Interface graphique de WireShark.

Wireshark est un analyseur de paquets réseau open source, utilisé pour capturer et inspecter le trafic circulant sur un réseau informatique en temps réel. Il permet aux administrateurs réseau, développeurs et ingénieurs en cybersécurité de diagnostiquer des problèmes, analyser des protocoles, et détecter des comportements suspects. Grâce à son interface graphique intuitive, Wireshark offre une visualisation détaillée des paquets, avec la possibilité de filtrer, rechercher et décoder une grande variété de protocoles réseau.

5. Makefile

Un Makefile est un fichier utilisé par l'outil make pour automatiser la compilation et la gestion de projets logiciels, notamment en C/C++. Il contient des règles qui définissent comment générer des fichiers cibles à partir de fichiers sources, en précisant les dépendances et les commandes à exécuter. Cela permet de simplifier et d'accélérer le processus de compilation, en particulier dans les projets complexes.

Dans le cadre du projet SPY, il a été utilisé pour séquencer le déroulement de la simulation, en commençant par la génération des données d'entrée jusqu'à la production d'un rapport sur les résultats de la simulation.