E. N. S. T
المدرسة الوطنية العليا للتكنولوجيا
Ecole nationale supérieure de technologie

GEii
Génie Électrique et Informatique Industrielle

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Département : Génie électrique Et Informatique Industrielle

Mémoire de fin d'étude en vue de l'obtention du diplôme de

Ingénieur d'état

Spécialité

Systèmes Embarqués

- Thème -

Conception et implémentation sur FPGA d'un système de détection des menaces sur les objets IOT basé sur l'architecture Zynq

Réalisé par

Billal MENZOU

Présenté et soutenu publiquement le (02/07/2023)

Composition du Jury :

Président	M Abdelhalim ZAOUI	Pr	ENST
Promoteur	Mme Faiza BELHACHAT	MCA	ENST
Examinateur	M Belkacem ABBADI	MCB	ENST
Examinateur	Mme Khadidja ZELLAT	MCB	ENST

Année Universitaire 2022-2023

Dédicaces

Je dédie ce travail à ;

La mémoire de ma grand-mère ;

Mes chers parents à qui je dois tout ;

Mes deux petites sœurs ;

Ainsi qu'à tous mes amis.

Billy

Remerciements

Je tiens à remercier mon encadrante Madame **F. Belhachat** pour avoir accepté de diriger ce travail, son aide scientifique et psychologique était d'une très grande importance durant toute la période de la réalisation de ce mémoire.

Je remercie les honorables membres du jury pour m'avoir fait l'honneur de bien vouloir évaluer mon travail.

Je remercie également toute l'équipe pédagogique de l'ENST de tout son soutien et accompagnement durant tout mon cursus universitaire.

Enfin, je tiens à remercier les membres de ma famille pour tous leurs soutiens inconditionnels et leurs encouragements, durant tout mon cursus universitaire et en particulier durant la période de réalisation de ce projet de fin d'étude.

Table des matières

Liste des figures I
Liste des abréviations III
Introduction générale 1

Chapitre I : Généralités sur les FPGA et les SoC

I.1 Introduction 4
I.2 Les PLD (Programmable Logic Devices) 4
I.2.1 Les PAL (Programmable Array Logics) 5
I.2.2 Les EPLD (Erasable Programmable Logic Devices) 5
I.2.3 Les GAL (Generic Array Logic) 5
I.2.4 Les CPLD (Complex Programmable Logic Devices) 5
I.2.5 Les FPGA (Field Programmable Gate Array) 5
I.3 Architecture d'un FPGA 6
I.4 Systèmes embarqués 7
I.5 Les systèmes sur puce SOC 8
I.5.1 Définition 8
I.5.2 Les caractéristiques des SOC 9
I.6 Les composants virtuels IP 9
I.7 Flot de conception 10
I.7.1 Flot de conception classique 10
I.7.2 Flot de conception haut niveau 11
I.9 Présentation de la carte de développement Zybo Z-7 12
I.10 Conclusion 12

Chapitre II : Généralités sur les objets connectés

II.1 Introduction 15
II.2 Historique de l'Internet des Objets 15
II.3 Généralité sur l'internet des objets 15
II.3.1 Définition de l'Internet des Objets 15
II.3.2 Applications et utilisation de l'IOT 16
II.3.3 Architecture standard d'un système IOT 17
II.3.4 Composant de l'IOT 18
II.4 Modèle OSI 18
II.5 Internet des Objets et FPGA 20
II.6 Les menaces possibles 20
II.6.1 Couche application 20
II.6.2 Couches intermédiaires 20
II.6.3 Couche physique 21

II.7 Menaces retenues 21
II.8 Détails sur les menaces retenues 21
II.8.1 Déni de service 21
II.8.2 Brouillage 22
II.8.3 Blocage de mise en veille 22
II.9 Conclusion 23

Chapitre III : Conception, simulation et implémentation du système de détection des menaces sur les objets IOT sur FPGA

III.1 Introduction 25
III.2 Structure matérielle et logicielle de notre système 25
III.3 Etapes de création du projet dans l'outil Vivado 25
III.3.1 Création des modules 26
III.3.2 Création des IP 29
III.4 Les modules de notre système 30
III.4.1 Menace de déni de service 30
III.4.2 Menace de brouillage 33
III.4.2.1 Test du bloc responsable de la détection de la menace de brouillage .. 36
III.4.3 Menace de blocage de la mise en veille du système 37
III.4.3.1 Test du bloc responsable de la détection de la menace de blocage de mise en veille 39
III.4.3.2 Implémentation sur FPGA du bloc de détection du blocage de mise en veille 40
III.5.1 Le schéma du module 44
III.5.2 Génération du wrapper 45
III.5.2 Synthèse et implémentation 46
III.6 Le logiciel Xilinx SDK 48
III.7 Conclusion 48
Conclusion générale et perspectives 49
Annexes 51
Bibliographie 62

Liste des figures

Figure I.1 : Architecture d'un PLD basique. 13
Figure I.2: Architecture d'un FPGA. 15
Figure I.3: Système embarqué. 17
Figure I.4: Les types d'IP. 18
Figure I.5: Flot de conception classique. 19
Figure I.6 : Flot de conception haut niveau. 20
Figure I.7 : Architecture de la famille Zynq. 20
Figure I.8 : La carte de développement Zybo Z-7. 22
Figure II.9 : Internet des Objets. 25
Figure II.10 : Les applications de l'IOT. 26
Figure II.11 : Les couches d'un système IOT. 27
Figure II.12 : Le modèle OSI. 28
Figure II.13 : Le modèle TCP-IP. 28
Figure III.14: Structure matérielle et logicielle de notre système. 34
Figure III.15 : L'interface de l'outil Vivado. 35
Figure III.16 : Etape 1 de la création du projet. 36
Figure III.17 : Etape 2 de la création du projet. 37
Figure III.18 : Etape 3 de la création du projet. 37
Figure III.19 : Etape 4 de la création du projet. 38
Figure III.20 : Etape 5 de la création du projet. 38
Figure III.21 : Etape 1 de la création d'un IP. 39
Figure III.22 : Etape 2 de la création d'un IP. 39
Figure III.23 : Code de détection de la menace de déni de service. 40
Figure III.24 : Création de l'IP de détection de la menace de déni de service. 41
Figure III.25 : Fonctionnement de l'IP de détection de déni de service. 42
Figure III.26 : Schéma de l'IP de détection de la menace de déni de service. 41
Figure III.27 : Code de détection de la menace de brouillage. 43
Figure III.28 : Fonctionnement de l'IP de détection de brouillage. 43
Figure III.29 : Création de l'IP de détection de la menace de brouillage. 43
Figure III.30 : Schéma de l'IP de détection de la menace de brouillage. 44
Figure III.31 : Banc de test du bloc de détection de la menace de brouillage. 45
Figure III.32 : Les valeurs obtenues après la simulation de la menace de brouillage. 45
Figure III.33 : Code de détection de la menace de blocage de mise en veille. 46
Figure III.34 : Création de l'IP de détection de la menace de blocage de mise en veille. 46
Figure III.35 : Création de l'IP de détection de la menace de blocage de mise en veille. 43
Figure III.36 : Schéma de l'IP de détection de la menace de blocage de mise en veille. 47
Figure III.37 : Le banc de test du bloc de détection de la menace de blocage de mise en veille. 47
Figure III.38 : Les signaux obtenus après la simulation de la menace de blocage de mise en veille. 48
Figure III.39 : Fichier de contrainte. 49
Figure III.40 : Fin de l'étape de synthèse 49

Figure III.41 : Rapport énergétique généré après l'étape de synthèse. 49
Figure III.42 : Fin de l'étape d'implémentation 49
Figure III.43 : Les ressources matérielles utilisées. 50
Figure III.44 : Génération du fichier binaire (bitstream) 49
Figure III.45 : Implémentation sur la carte FPGA. 51
Figure III.46 : Architecture matérielle complète du système de détection de la menace de blocage de mise en veille. 52
Figure III.47 : Le wrapper 53
Figure III.48 : Les ressources matérielles utilisées. 54
Figure III.49 : Les CLB et les interconnections dans le circuit FPGA. 54
Figure III.50 : Rapport énergétique. 55

Liste des abréviations

ARP : Adress Resolution Protocol.

ASIC : Application Specific Integrated Circuits.

AXI : Advanced eXtended Interface.

CLB : Cell Logic Block.

CPLD : Complex Programmable Logic Devices.

DDOS : Distributed Denial Of Services.

DOS : Denial Of Services

EMIO : Extended Multiplexed Input Output.

EPLD : Erasable Programmable Logic Devices.

FFT : Fast Fourier Transform.

FPGA : Field Programmable Gate Array.

GAL : Generic Array Logic.

HDL : Hardware Description Language.

HLS : High Level Synthesis.

IOB : Input Output Block

IOT : Internet Of Things..

IP : Intellectual Propertie.

LUT : Look-Up Table.

MIO : Multiplexed Input Output.

MITM : Men In The Middle.

OSI : Open Systems Interconnection.

PAL : Programmable Array Logic.

PL : Programmable Logic.

PLD : Programmable Logic Devices.

PS : Processing System.

SDK : Software Development Kit.

SoC : System on Chip.

SoPC : System on Programmable Chip.

TCP-IP : Transmission Control Protocol-Internet Protocol.

Trame-IP : Internet Protocol.

VHDL : VHSIC (Very High Speed Integrated Circuits) Hardware Description Language.

Wi-Fi : Wireless Fidelity

Introduction générale

L'internet des objets ou Internet of Things (IOT) en anglais est une technologie qui a pris une place importante dans notre quotidien. C'est un concept dans lequel plusieurs objets intelligents et autonomes sont connectés entre eux et avec internet, échangent des données sans l'intervention de l'être humain et qui ont une certaine capacité de prise de décision, de mémorisation et de calcul. La puissance de l'IOT repose sur les avancées dans les systèmes embarqués.

Les systèmes embarqués sont des systèmes autonomes qui effectuent une tâche bien précise ce qui les rend très performants, ils jouent un grand rôle dans le domaine de l'IOT notamment dans la collecte et le traitement des données émises par les objets connectés.

Afin de répondre aux exigences croissantes dans le domaine de l'IOT, telles que la sécurité et le traitement des données en temps réel, les systèmes embarqués nécessitent une puissance de calculs élevée avec une grande flexibilité.

Les circuits FPGA sont largement utilisés pour le développement des applications IOT car ils offrent une grande puissance de calcul, une grande capacité de reprogrammation et une liaison entre le monde virtuel et le monde réel.

L'importance de l'IOT réside dans la connectivité des appareils, ce qui améliore les services pour les utilisateurs. Cependant, un réseau qui utilise internet pour la transmission des données est vulnérable aux problèmes de sécurité. En effet, les objets IOT peuvent être soumis à diverses attaques qui dégradent leurs performances ou même les mettre hors service.

L'objectif principal de ce projet de fin d'études est de concevoir un système qui permet de détecter certaines menaces sur les objets connectés, ce système traitera la menace de déni de service, de brouillage et de blocage de la mise en veille. Le système est composé d'une partie matérielle qui s'implémente sur FPGA, elle consiste en les trois blocs qui traitent les menaces sous forme de composants virtuels (IP), et d'une partie logicielle gérée par le processeur ARM A9.

Ce manuscrit est organisé en trois chapitres comme suit :

- Dans le premier chapitre nous introduisons la famille des circuits programmables, en particulier les circuits FPGA, et la notion des systèmes sur puce (SoC System on Chip), puis, nous présentons le concept de composants virtuels (IP Intellectual Properties). Nous concluons par la description de l'architecture zynq et de la carte Zybo z-7 utilisée dans notre travail.
- Dans le deuxième chapitre, nous commençons par un historique sur la technologie de l'internet des objets, puis, nous introduisons les aspects généraux de l'IOT à travers sa définition, ses domaines d'utilisation, son architecture et ses équipements, nous présentons également les intérêts découlant de l'utilisation des circuits FPGA pour la technologie IOT, les

menaces possibles qui peuvent exister pour les objets IOT ainsi qu'une explication détaillée des menaces retenues dans notre travail.

- Dans le dernier chapitre nous présentons la structure matérielle et logicielle du système, puis nous décrivons l'outil de développement Vivado et les étapes nécessaires pour la création de notre projet et la création des blocs d'IP de détection des menaces. Ensuite, nous présentons le principe de fonctionnement des modules de détection de menaces ainsi que les résultats des simulations obtenus. Nous présentons également les résultats de la synthèse et de l'implémentation sur FPGA. Enfin, nous décrivons l'architecture complète du système de blocage de la mise en veille incluant l'IP de détection de cette menace connectée au processeur ARM A9 qui se charge de la gestion la partie logicielle ainsi que d'autres bloc IP développés par Xilinx.
- Enfin, nous terminons ce mémoire par une conclusion générale ainsi que des perspectives pour l'amélioration des performances de notre système.

Chapitre I : Généralités sur les FPGA et les SoC

I.1 Introduction

Depuis l'invention du circuit intégré par Jack Kilbey en 1958, les recherches dans ce domaine on permit l'amélioration des caractéristiques des circuits intégrés par une augmentation de la fréquence de fonctionnement, une consommation d'énergie de plus en plus réduite et une surface de silicium occupée optimisée. Cette avancée technologique a permis l'apparition des circuits ASIC qui sont des circuits performants mais qui n'est pas flexible et donc spécifiques aux applications pour lesquelles ils ont été développés. Puis, l'apparition des circuits logiques programmable (PLD) a permet de régler le problème de flexibilité puisqu'ils pouvaient être reprogrammés en fonction de l'application souhaitées mais avec de faibles capacités. Enfin, l'apparition des circuits FPGA qui combinent les avantages des deux familles mentionnées, la capacité de reprogrammation (flexibilité) des circuits logiques programmables et les performances des circuits ASIC [1].

Dans ce chapitre, nous exposons l'évolution des circuits à logiques programmables, depuis les PAL en passant par les CPLD jusqu'à les circuits FPGA qui seront présentés en détail. Puis, nous introduisons le système embarqué, le concept de SOC, et les composants virtuels (IP). Les flots de conception d'un circuit FPGA (classique et de haut niveau), l'architecture zynq ainsi que la carte Zybo Z-7 seront également présentés.

I.2 Les PLD (Programmable Logic Devices)

Les réseaux logiques programmables sont des circuits logiques qui peuvent remplacer plusieurs circuits en TTL ou CMOS et on peut les reprogrammer selon la fonction logique qu'on souhaite implémenter [2].

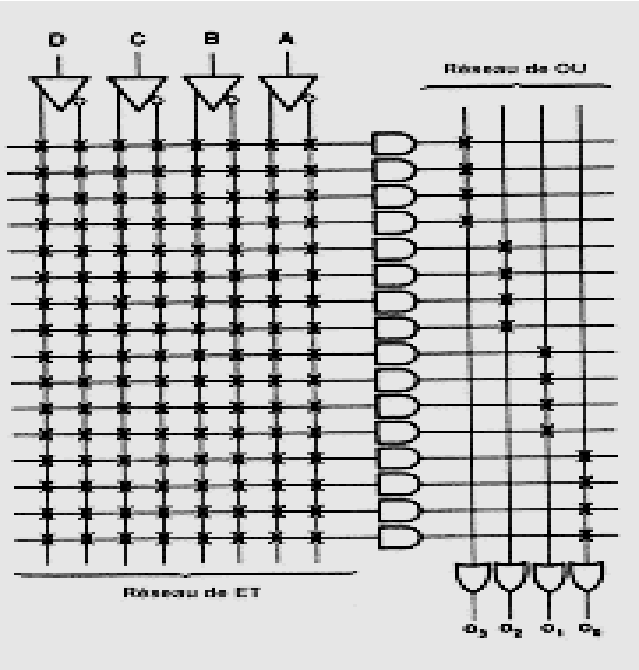


Figure I.1 : Architecture d'un PLD basique [3].

Il existe plusieurs familles de PLD :

I.2.1 Les PAL (Programmable Array Logics)

C'est un type de circuit programmable par grillage des fusibles, et une fois programmé on ne peut plus effacer. Les PAL sont constitués de :

- Ensembles de portes AND sur lesquelles les entrées sont connectées.
- Ensembles de portes OR sur lesquelles les sorties des portes AND sont connectées.

Il existe deux variantes :

- Les PAL : seules les entrées sont programmables (les matrices AND), les matrices Or sont fixées.
- Les PLA : les deux matrices (Or et AND) sont programmables.

I.2.2 Les EPLD (Erasable Programmable Logic Devices)

Ils sont des circuits en technologie CMOS avec les avantages par rapports aux PAL d'être effaçables électriquement ou par ultra-violets (UV) et une basse consommation d'énergies.

I.2.3 Les GAL (Generic Array Logic)

Ils ont un fonctionnement similaire aux EPLD et ils sont effaçables électriquement seulement.

I.2.4 Les CPLD (Complex Programmable Logic Devices)

Sont des circuits composés de plusieurs PAL reliés par des matrices d'interconnexion, Ils arrivent à atteindre des grandes vitesses de fonctionnement grâce à cette architecture.

I.2.5 Les FPGA (Field Programmable Gate Array)

C'est une famille de circuits intégrés inventés par la société XILINX en 1985, elle combine la capacité de reprogrammation des PLD (Programmable Logic Devices) et la complexité des circuits ASIC (Application Specific Integrated Circuits) , donc, un FPGA permet d'avoir une architecture à haute densité avec la possibilité de la reprogrammer pour de nouvelles applications, qui se fait grâce un langage de description matérielle HDL (Hardware Description Language) tel que : VHDL ou verilog qui sera traduit dans un fichier de configuration pour le composant cible [2].

I.3 Architecture d'un FPGA

Les FPGA sont composés principalement de : bloc de configuration logique (CLB), bloc d'entrées-sorties (IOB) et les différentes interconnexions :

- **Les CLB :** ils sont les principaux composants des FPGA, dans les FPGA de la société XILINX on les appelle slice, composés des LUT (Look Up Table) qui sont simulés à un multiplexeur et des bascules. Lors de la configuration, les mémoires des LUT sont écrites, et la logique permet l'implémentation des fonctions logiques.
- **Les IOB :** ils permettent l'interconnexion de la logique interne aux entrées/sorties, ils sont présents sur toute la périphérie du circuit et ils peuvent être définis en entrées, en sortie ou en entrées-sorties.
- **Les interconnexions :** leur rôle est de relier les blocs logiques configurables et les blocs d'entrées-sorties avec un maximum d'efficacité pour une utilisation plus optimale. Il existe trois types d'interconnexions :
    - **Les interconnexions directes** : ce type d'interconnexion permet de relier les CLB aux IOB et de relier directement certains CLB entre eux.
    - **Les longues lignes** : elles parcourent toutes la longueur du circuit, elles permettent une transmission sans retard des signaux et permettent d'éviter l'encombrement des points d'interconnexions.
    - **Les matrices d'interconnexions** : leur rôle est de raccorder les grandes lignes, elles sont à chaque intersection. Des buffers sont implantés dans les matrices pour éviter l'affaiblissement des signaux [4].

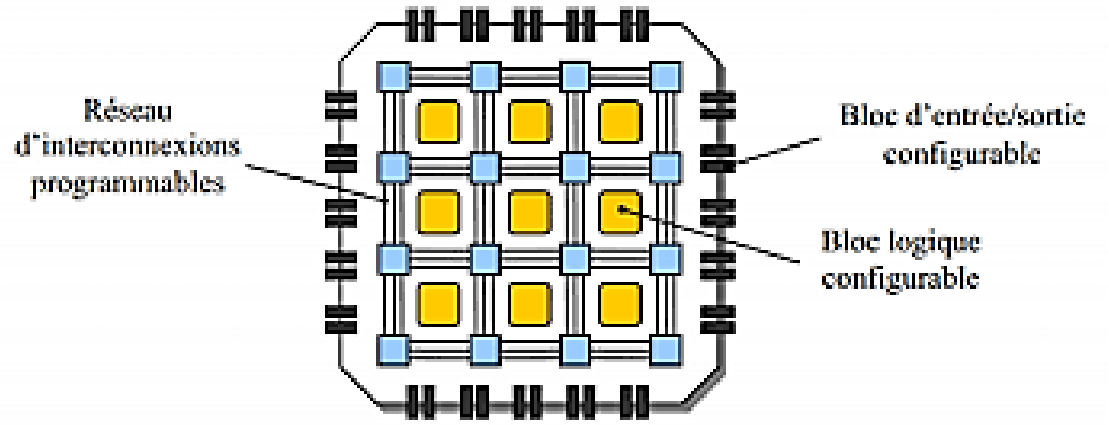


Figure I.2 : Architecture d'un FPGA [5].

Les trois blocs qu'on vient de décrire sont les blocs principaux, mais un FPGA possède des ressources additionnelles :

- Les blocs RAM : sont des mémoires de type double ports qui permettent l'écriture et la lecture avec des fréquences d'horloge différentes, elles sont configurables en blocs de 16 ou 32 KO selon la catégorie du circuit FPGA.
- Les DSP (Digital Signal Processing) : ils sont des blocs qui permettent de faire du traitement de signal numérique ou réaliser des opérations arithmétiques en virgule flottante ce qui permet de faire des traitements plus rapides et plus précis.
- Les processeurs embarqués : ces processeurs peuvent être des processeurs hardcore ou des processeurs softcore :
    1. Processeurs hardcore : sont des processeurs décrits au niveau physique (layout sur silicium), les performances du système en termes de vitesse et de surface de silicium sont optimisées et elles sont connues et ne varient pas d'un système à un autre mais les processeurs hardcore souffrent d'un grand manque de flexibilité et donc ils sont spécifiques aux applications pour lesquelles ils sont conçus. Par exemple : PowerPC et ARM.
    2. Processeurs softcore : sont des processeurs décrits par un langage de description matérielle (HDL), avec des performances réduites par rapport aux processeurs hardcore, mais jouis d'une grande flexibilité ce qui les rend reprogrammables selon l'application souhaitée et s'adaptent aux besoins des développeurs. Par exemple : microblaze et NIOS.
- Les DCM (Digital Clock Manager) : il permet d'obtenir des périodes d'horloges différentes à partir d'une fréquence d'horloge externe fixe [4].

I.4 Systèmes embarqués

Un système embarqué est un système électronique et informatique autonome qui réalise une tâche bien précise, souvent en temps réel qui peut être strict où le non-respect des contraintes temporelles est intolérable et peut causer des dégâts très graves, ou souple où le non-respect des contraintes temporelles est parfois tolérable, cela dépend de l'application, avec différentes contraintes physiques :

- Consommation : un système embarqué ne doit pas consommer beaucoup d'énergie car il pilote un système portable avec des ressources énergétiques limitées donc, il doit assurer une bonne autonomie et ne doit pas influencer sur le caractère portable du système.
- Poids et dimension : le système embarqué ne doit pas avoir un poids très important et ne doit pas être très encombrant dans le système qu'il pilote, d'où

la caractéristique de limiter les tâches que peut réaliser un système embarqué ce qui permet d'avoir un nombre optimal de périphériques.

Un système embarqué doit être également sûr de fonctionnement, en garantissant ces paramètres :

- Fiabilité : le système fonctionne correctement sous certaines conditions pendant un intervalle de temps.
- Maintenabilité : le système est réparable dans un certain délai.
- Disponibilité : le système peut fonctionner correctement à l'instant où il est sollicité.
- Sécurité : le système ne cause pas de dégâts matériels ou humains pendant son fonctionnement.

Le coût d'un système embarqué est une caractéristique très importante, il doit être le plus bas possible car la majorité vise des applications publiques donc, doit avoir un prix abordable [6].

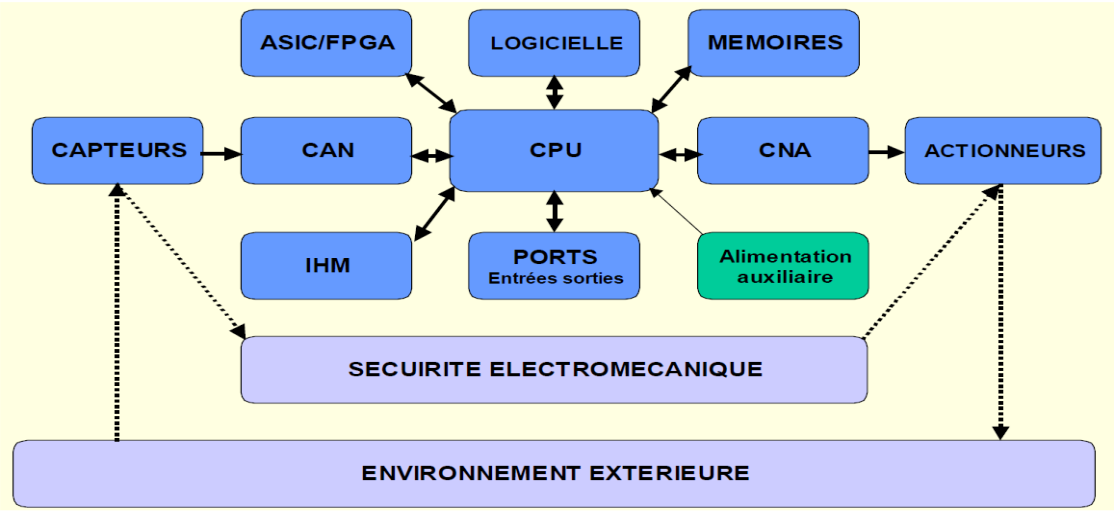


Figure I.3 : Système embarqué [7].

I.5 Les systèmes sur puce SOC

I.5.1 Définition

Les SOC (System On Chip) désignent l'intégration d'un système complet sur une seul puce, ils sont omniprésents dans nos vie, l'exemple le plus simple est le téléphone portable avec un processeur qui gère différentes choses : l'appareil photo (traitement des images), navigateur internet, les appels, etc.

Les SOC ont été créés pour le développement des ASIC et avec l'apparition des circuits FPGA, on a créé l'approche SOPC (System On Programmable Chip), qui est une approche plus compatible avec la caractéristique reprogrammable des circuits FPGA.

- **SOC :** ils sont destinés pour le développement des ASIC, plus performants en termes de vitesse et de consommation, utilisés pour les applications grand public et produits en grande quantités mais ils sont figés et ne sont utilisés que pour les applications pour lesquelles ils sont développés.
- **SOPC :** ils sont destinés pour le développement des FPGA, moins performants qu'un ASIC, utilisés pour le prototypage, ils sont plus flexibles : on peut les réutiliser selon l'application souhaitée par le programmeur [4].

I.5.2 Les caractéristiques des SOC

- **Hétérogénéité des SOC :** un SOC a une partie mécanique ainsi qu'une partie électronique, qui peut contenir une partie analogique et une partie numérique, et la partie numérique peut se décomposer aussi en une partie logicielle et une partie matérielle.
- **Un SOC réalise une tâche bien précise :** un système sur puce cible toujours une application bien spécifique, contrairement aux systèmes d'usage général qui sont plutôt polyvalents et réalisent diverses tâches à la fois. Cela ne signifie pas qu'un système sur puce est à chaque fois conçu de zéro, il y a donc, la notion de réutilisation qui ne contredit pas la spécificité d'un système sur puce
- **Intégration du SOC dans un système plus complexe :** lorsqu'un SOC est conçu, il va être intégré dans un système plus complexe qui va piloter. Le SOC va communiquer avec les différents périphériques d'E/S en utilisant généralement des CAN (Convertisseur Analogique Numérique) ou CNA (Convertisseur Numérique Analogique).
- **Réduction de latence :** dans un SOC, tout le système est dans une puce (processeur et mémoires), donc le temps d'accès du processeur aux données en mémoires est réduit.
- **Réduction de coût :** puisque la logique et la mémoire se trouvent à la même place donc on peut réduire les broches ainsi que le câblage ce qui va permettre de réduire le coût de conception [4].

I.6 Les composants virtuels IP

Les IP (Intellectual Proprieties) sont des circuits qui réalisent une fonction donnée qui peut être très simple (compteur) ou très complexe (un processeur) pouvant être réutilisé par un autre utilisateur dans l'objectif de réduire le temps de développement des systèmes.

Il existe trois types de composant virtuels :

- **IP softcore (logiciel) :** c'est un IP décrit par un langage HDL, il a l'avantage d'être synthétisable donc portable et peut être reprogrammé selon le besoin,

mais son inconvénient est qu'on ne peut prédire ses caractéristiques en termes de vitesse, de consommation d'énergie et de la surface de silicium.

- **IP hardcore (matériel) :** c'est un IP décrit par des portes logiques et englobe la Netlist toute entière, il n'est pas portable car il dépend de la technologie pour laquelle il a été développé, son principal avantage est qu'il est prédictible en termes de vitesse, de consommation d'énergie et de la surface
- **IP firmcore :** ce type d'IP est un compromis entre les IP hardcore et softcore, il est prédictible comme un IP matériel et synthétisable (donc portable) comme un IP logiciel. Il est décrit par une combinaison de code synthétisable, une Netlist et des portes logiques [4], [7].

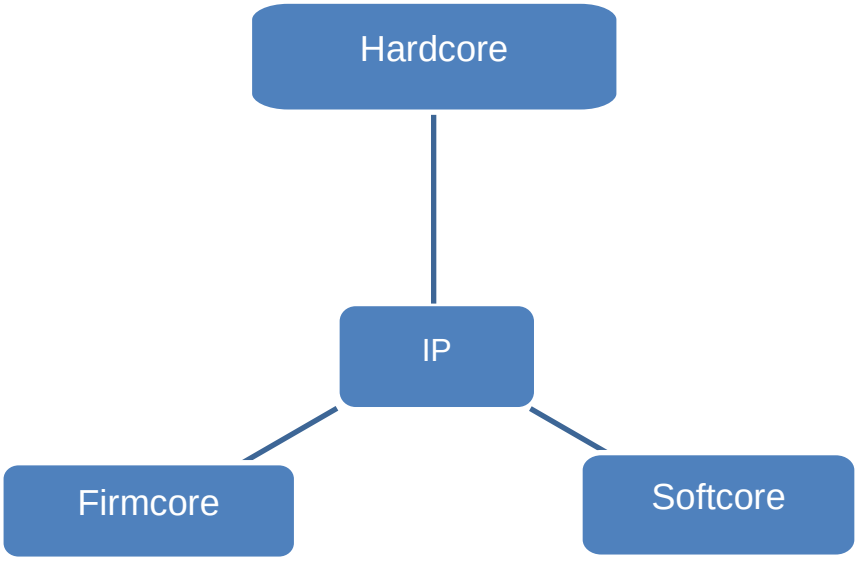


Figure I.4 : Les types d'IP.

I.7 Flot de conception

Il s'agit d'un ensemble d'étapes qu'un développeur doit suivre afin de mettre en œuvre une application sur une carte FPGA. Il existe deux flots de conception : un classique et l'autre de haut niveau.

I.7.1 Flot de conception classique

On commence avec un langage de description de matériel (HDL) comme VHDL ou verilog qui représente l'architecture du circuit. La description matérielle subit une étape de synthèse logique qui engendre un fichier Netlist, ce fichier met en lien les différents éléments logiques du circuit, la prochaine étape est la correspondance qui s'agit d'une traduction du fichier Netlist à des composants qui existent sur la carte FPGA (les LUT et les bascules dans les CLB), ensuite l'étape de placement routage, cette étape s'agit de connecter les éléments de la description matérielle et les éléments dans le FPGA cible. Enfin, on obtient le fichier binaire (bitstream) qui va être chargé dans le FPGA [4].

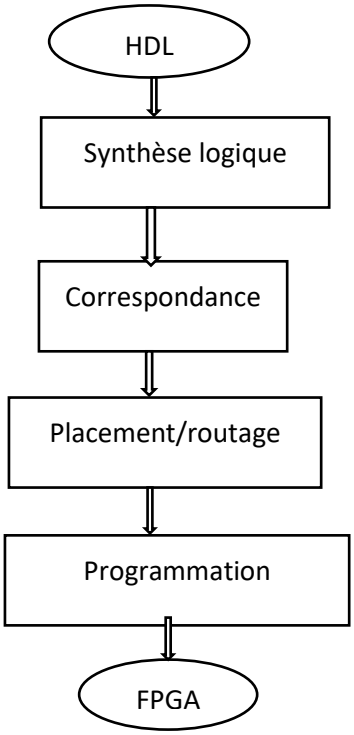


Figure I.5 : Flot de conception classique [9].

I.7.2 Flot de conception haut niveau

On commence avec un langage de haut niveau (c, c++) qui représente une description algorithmique qui facilite et réduit le temps de développement d'un circuit, à la fin de cette étape on obtient la description du circuit dans un langage de description de matériel HDL. La suite du flot est similaire au flot de conception classique [4].

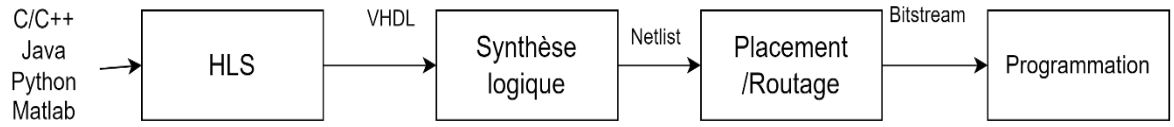


Figure I.6 : Flot de conception haut niveau [10].

I.8 Présentation de la famille de SOC ZYNQ

La Zynq est une famille de FPGA de la série 7 qui contient en plus de la partie logique programmable (Programmable Logic PL) un système à microprocesseur ARM doubles cœurs avec ses contrôleurs mémoire et ses périphériques (Processing System PS). Ils existent différents modèles de ZYNQ.

La partie PS et la partie PL communiquent entre elles via le bus EMIO (Extended Multiplexed Input Output) et le bus AXI (Advanced eXtensible Interface).

La partie PS communique avec les périphériques d'entrées/sorties via le bus MIO (Multiplexed Input Output), elle utilise aussi le bus EMIO lorsque les 54 broches du bus MIO sont occupées [4].

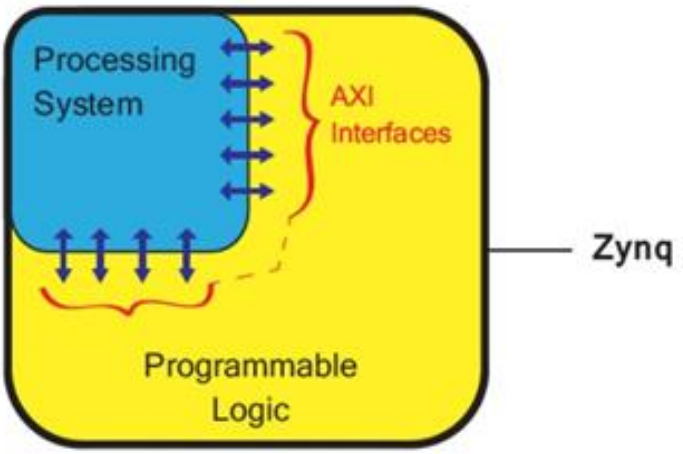


Figure I.7 : Architecture de la famille Zynq [12].

I.9 Présentation de la carte de développement Zybo Z-7

La carte de développement utilisée dans notre projet est la carte ZYBO-Z7. C'est une carte à faible coût. Elle est basée sur un SOC du type Z-7 (XC7Z010-1CLG400C).

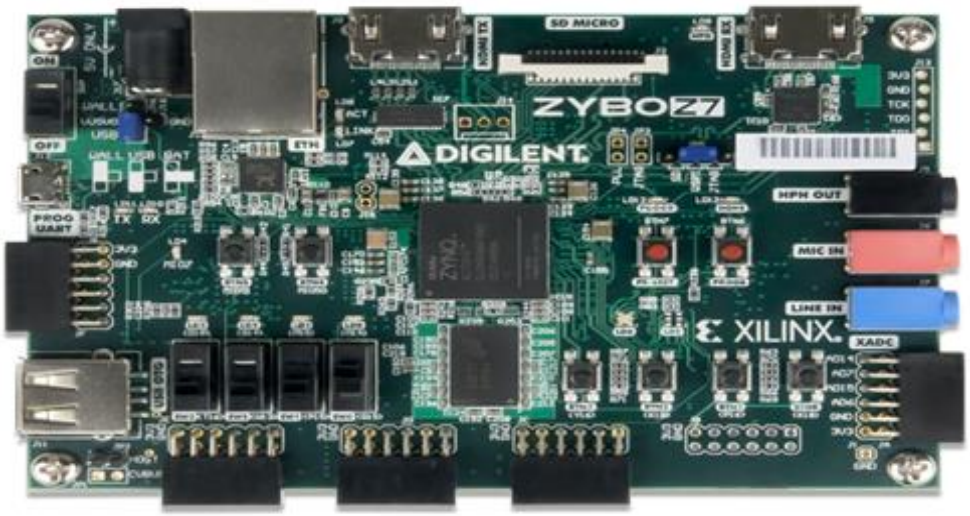


Figure I.8 : La carte de développement Zybo Z-7 [11].

I.10 Conclusion

Dans ce chapitre, nous avons vu l'évolution des circuits logiques programmables qui ont conduit à l'apparition des circuits FPGA, nous avons défini aussi le système

embarqué avec toutes ses spécificités, nous avons également introduit la notion de SoC, les composants virtuels, les flots de conception des circuits FPGA, l'architecture Zynq ainsi que la carte cible Zybo Z-7 utilisée.

Dans le prochain chapitre nous allons aborder les concepts de l'internet des objets ainsi que les menaces pour ces objets.

Chapitre II : Généralités sur l'Internet des Objets.

II.1 Introduction

L'internet des objets est un domaine qui a connu une avancée majeure durant ces dernières années, il désigne plusieurs objets physiques qui sont connectés à internet et échangent des données entre eux sans l'intervention de l'être humain.

Les objets IOT ont la particularité d'embarquer peu de puissance de calcul avec des ressources énergétiques limitées, ce qui les rend vulnérables aux attaques. En effet, si les appareils n'ont pas une grande puissance de calcul, ils ne peuvent pas se protéger contre les attaques ou du moins ne peuvent pas les détecter. Il est donc intéressant de mettre en place des dispositifs qui permettent de détecter les menaces et de les contrer.

Dans ce chapitre, nous introduisons l'historique de l'Internet des Objets. Ensuite, nous présentons des généralités sur les l'IOT en donnant une définition de l'internet des objets, les domaines de son utilisation, l'architecture des systèmes IOT, les équipements de l'IOT ainsi que le protocole OS. Nous présentons aussi l'intérêt de l'utilisation des cartes FPGA pour les systèmes IOT, puis nous présentons les menaces les plus connues sur les objets IOT en général, et nous détaillons les menaces retenues.

II.2 Historique de l'Internet des Objets

- L'apparition du terme Internet des Objets en 1999 par Kevin Ashton, qui désigne des machines physiques connectées à internet et qui échangent des données. Mais le concept était là bien avant, en 1970, un système de surveillance à distance des compteurs électriques ont été implémentés ; et dans les années 90, la technologie sans fil avancée a permis la surveillance des équipements.

- En 2015, deux chercheurs (Sang Don Kim and Seung Eun Lee) ont réussi à implémenter un environnement IOT dans un FPGA Altera, ils ont implémenté un système de surveillance de la température et de l'humidité en utilisant des capteurs qui communiquent avec FPGA via un port USART.

- En 2016, la mise en œuvre d'un système de contrôle multi-capteurs connecté à un serveur internet avec une application serveur qui s'exécute sur une adresse IP dédiée.

- En 2016, deux scientifiques (Gasim Alandjani et et.al) proposent un design d'une machine à ECG sur FPGA en utilisant la technique de mise à l'échelle de la capacité quand l'appareil fonctionne sur des fréquences spécifiques [13].

II.3 Généralité sur l'internet des objets

II.3.1 Définition de l'Internet des Objets

Le terme Internet des Objets (IdO) a été introduit pour la première fois par l'ingénieur britannique Kevin Ashton en 1999, pour désigner des objets physiques connectés à

internet qui ont la capacité de créer et transmettre les données. En 2012, une définition a été donnée par l'UIT (Union International de Télécommunication) : " *une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution*". Pour ce travail on va définir l'Internet des Objets comme étant un ensemble d'appareils physiques connectés entre eux et avec internet, avec la capacité d'échanger des données sans l'intervention de l'être humain [14].



Figure II.9 : Internet des Objets [15].

II.3.2 Applications et utilisation de l'IOT

Le domaine de l'IOT est en expansion continue et devient de plus en plus présent dans notre vie quotidienne.

On cite quelques domaines où on trouve l'internet des objets :

- Les objets portables : divers objets ont été implémentés tels que : les dispositifs de soins portables et les objets Google Glass qui permettent de contrôler des applications sur internet.
- Les véhicules connectés : par exemples les systèmes de navigations qui aident les personnes à trouver leurs chemins.
- Les maisons intelligentes : des systèmes qui permettent le contrôle des lumières, de la climatisation et divers objets électriques depuis la même interface (smart phone par exemple), l'avancée dans l'IOT a conduit à l'apparition des villes intelligentes.

- Villes intelligentes : grâce à l'IOT, on peut utiliser les infrastructures d'une ville (les réseaux électriques par exemple) d'une manière plus optimisée, et ainsi améliorer la qualité de vie.
- Services de localisation : plusieurs services de localisation sont fournis par l'IOT : localisation des enfants, soins aux personnes âgées et le commerce électronique.
- Agriculture : en utilisant les réseaux de capteurs connectés à internet, on peut mieux surveiller l'état des implantations pour améliorer les récoltes et mieux prévenir contre les dégâts.
- Santé : dans ce domaine, l'IOT peut permettre par exemple de surveiller les constantes biologiques des patients à distance à l'aide des capteurs de température et de saturation en oxygène.
- L'industrie : la technologie IOT permet un suivi plus minutieux des processus de production, de la logistique des matières premières jusqu'à l'obtention du produit final et elle permet plus de sécurité vis-à-vis des dégâts ainsi qu'une meilleure qualité des produits [13].

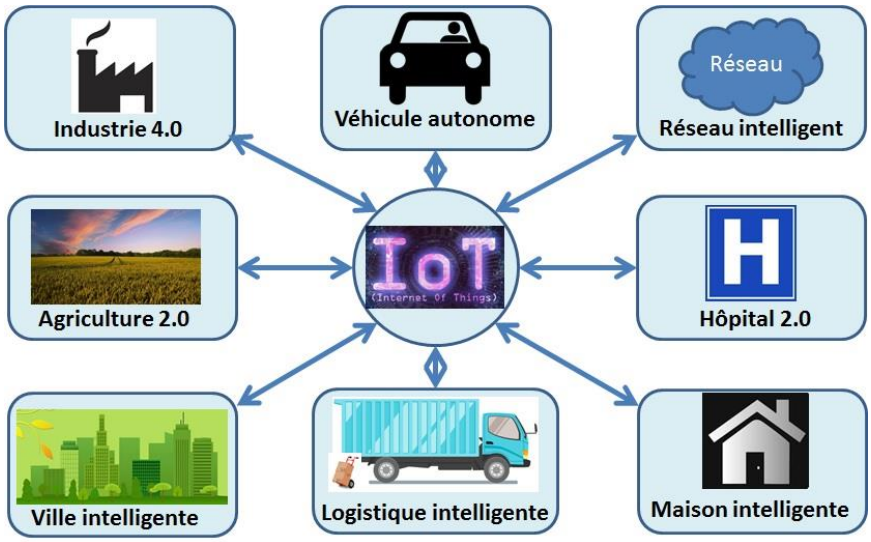


Figure II.10 : Les applications de l'IOT [16].

II.3.3 Architecture standard d'un système IOT

Les systèmes IOT utilisent plusieurs objets qui sont répartis selon trois couches qui sont : la couche application, la couche réseau et la couche de perception.

- La couche perception : les objets de cette couche ont pour rôle la détection des propriétés physiques externes telles que la température ou l'humidité et les convertir en signaux numériques, ils peuvent être des dispositifs de détection tels que les capteurs ou des actionneurs.
- La couche application : les objets de cette couche offrent aux utilisateurs une interface pour le contrôle des objets et la surveillance de leurs fonctionnements, ils sont généralement des logiciels.

- La couche réseau : cette couche contient les objets responsables de la transmission des signaux envoyés par les dispositifs de la couche de perception, les technologies cellulaires sont utilisées pour les objets de cette couche telles que : Wi-Fi, Bluetooth, Ethernet, etc. [17].

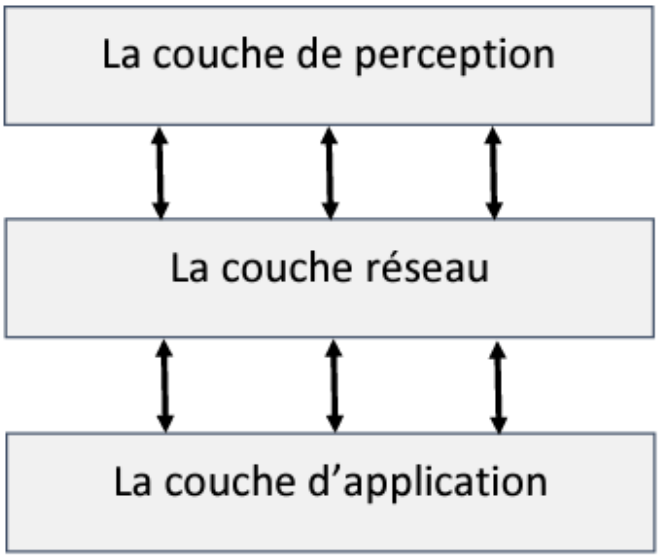


Figure II.11 : Les couches d'un système IOT [17].

II.3.4 Composant de l'IOT

Selon les couches du système IOT présenté ci-dessus, un objet IOT a pour fonction la collecte des données externes, transmettre les données sous forme de signal numérique et recevoir des instructions à exécuter en fonction des signaux en entrées, donc, selon ce découpage, on a d'une manière générale cinq composants :

- Capteurs : sont des dispositifs qui transforment un phénomène physique (vitesse, chaleurs, etc.) en un signal électrique utilisable.
- Réseaux de capteurs : un ensemble de capteurs dotés de la capacité de transferts les données sans fil.
- Actionneurs : des dispositifs avec la capacité de transformer un signal en une action, par exemple : les interrupteurs, vannes, serrures.
- Connectivité : elle est assurée par une antenne radio fréquence, elle permet la connexion d'un objet à un ou à plusieurs réseaux.
- Energie : il s'agit de la contrainte la plus importante dans un système IOT [17].

II.4 Modèle OSI

Le modèle OSI est un modèle en couche qui est apparu au début des années 1980, il est apparu pour régler le problème d'interconnexion des systèmes hétérogènes. Ce modèle a pour principe de découper les fonctions réalisées par un système de

télécommunication en couche superposées, où chaque couche réalise des tâches bien précises [18].

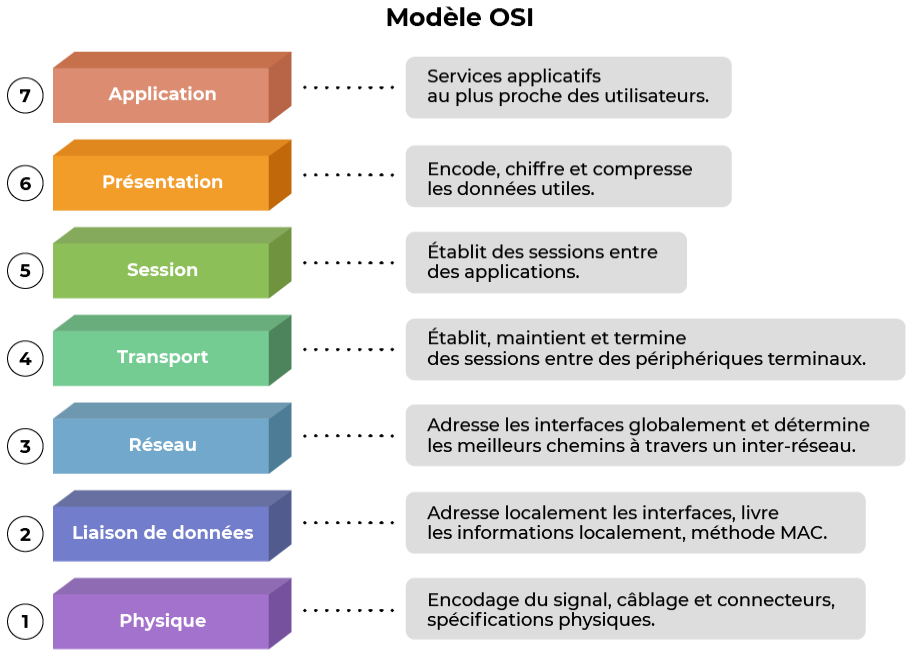


Figure II.12 : Le modèle OSI [19].

Le modèle OSI est un modèle purement expérimental, dans les applications réelles on retrouve plutôt le modèle TCP-IP, qui est un protocole en couche et se base sur l'adressage IP où chaque couche réalise des tâches bien définies [18].

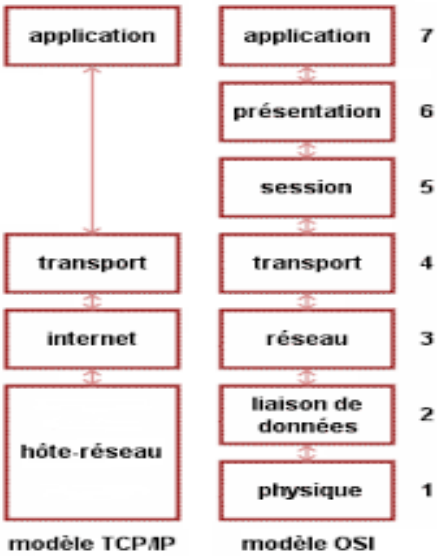


Figure II.13 : Le modèle TCP-IP [20].

II.5 Internet des Objets et FPGA

L'utilisation des FPGA dans les applications IOT permet de lier le hardware et le software et offre : la flexibilité, la fiabilité, un faible coût de développement et un time to market rapide.

Les FPGA offrent plusieurs avantages tels que :

- Un FPGA est reconfigurable donc offre la possibilité de réaliser des modifications selon l'application souhaitée plus facilement et plus rapidement.
- Un FPGA n'utilise pas un système d'exploitation mais utilise plutôt son hardware pour le traitement logique, donc les opérations ne rivalisent pas pour la même ressource ce qui donne des vitesses de traitement plus élevées.

Les plateformes FPGA sont utilisées pour des applications IOT pour deux raisons :

- On peut implémenter sur la plateforme FPGA le prototype de l'application SOC, qui offre une vérification des fonctions et des délais en temps réel.
- Le prototype FPGA ne peut pas être validé directement, l'automatisation de la conception est obligatoire pour la modélisation et le choix des composants, cette étape est essentielle pour produire un prototype efficace, et simplifier pour la traduction du prototype en un SOC final [13],[21].

II.6 Les menaces possibles

La liste qu'on va fournir ne contient pas toutes les menaces possibles, mais les plus connues dans un réseau IOT :

II.6.1 Couche application

- Injection d'un code malicieux par buffer overflow : cette menace consiste à écrire un processus dans une zone mémoire qui ne lui est pas destinée, donc l'attaquant peut exécuter des instructions non désirées.
- Bloquer la mise en veille : cette menace consiste à garder le système en marche pour épuiser son énergie plus rapidement, en envoyant des signaux.
- Envoie des données corrompues : elle consiste à corrompre les données envoyées dans le réseau.

II.6.2 Couches intermédiaires

- Déni de service (Denial Of Service DOS) : cette attaque consiste à submerger le réseau en envoyant un grand nombre de requêtes, c'est une attaque très utilisée.
- L'Homme du milieu (Man In The Middle MITM) : elle consiste à intercepter les données envoyées par le client au serveur tout en continuant à les transmettre, ce qui les rend invisibles des deux côtés (serveur et l'utilisateur). Cette attaque est inefficace si les données sont cryptées.

- Spoofing : l'attaque consiste à obtenir l'identité d'une machine (son adresse MAC) pour accéder au réseau, et l'attaquent peut, ensuite, faire une attaque de déni de service.
- Utilisation de la table ARP : l'ARP est un protocole qui associe l'adresse IP de la couche réseau (couche 3 du modèle OSI) et l'adresse MAC de la couche liaison de données (couche 2 du modèle OSI), pour simplifier, il permet à une machine de se connecter à un routeur, l'attaque consiste à utiliser les données de la table ARP pour accéder au réseau.
- Sniffing : il s'agit d'écouter et analyser le canal pour avoir des informations, et réaliser une des attaques citées ci-dessus.

II.6.3 Couche physique

Son plus grand inconvénient est que l'attaquant peut avoir un accès physique à l'objet :

- Brouillage : il s'agit d'émettre un signal dans une plage de fréquence qui perturbe les signaux utiles, ils peuvent être des signaux déterministes (de laboratoire) ou des signaux aléatoires.
- Sniffing : écouter et analyser le canal de la couche physique [22].

II.7 Menaces retenues

Notre système va se concentrer principalement sur l'attaque de déni de service (DOS) qui se produit dans les couches intermédiaires.

Pour la menace de blocage de mise en veille peut être détectée en vérifiant le niveau de la batterie de l'objet, et la vérification des données permet de détecter l'attaque d'envoie des données corrompues.

L'attaque de brouillage sur la couche physique peut être détectée en analysant les signaux reçus

II.8 Détails sur les menaces retenues

II.8.1 Déni de service

Le DOS est une attaque qui a pour objectif de rendre inaccessible les services d'une entreprise en s'attaquant à leurs serveurs. L'objectif de cette attaque n'est pas la collecte ou la suppression des données d'une entreprise qui offre un service internet mais plutôt de nuire à sa réputation en rendant inaccessible ces services aux clients.

Il existe deux types de méthodes pour faire une attaque de dénis de service :

- Déni de service par saturation : en saturant le serveur de requêtes et le rendre incapable de répondre aux requêtes des clients.
- Déni de service par exploitation des vulnérabilités : en exploitant une faille du système et le rendre inaccessible.

Le principe de la menace de dénis de service est d'envoyer un grand nombre de requêtes avec des constituants inhabituels pour altérer un service réseau ou dans des cas extrêmes provoquer le crash du serveur.

Cette attaque peut avoir des répercussions économiques assez conséquentes sur l'entreprise victime, en effet un arrêt de quelques minutes seulement peut arrêter plusieurs transactions.

Une autre variante de la menace de dénis de service, elle s'agit de la menace de déni de service distribué (DDOS) qui a le même principe qu'une attaque DOS mais avec plusieurs machines à la fois appelé zombie (botnet). Cette attaque se base sur un seul maitre et plusieurs esclaves, l'attaquant se connecte à la machine maitresse et ordonne aux hôtes esclaves d'émettre une attaque [24].

II.8.2 Brouillage

Le brouillage est une attaque très efficace contre les objets de l'IOT, elle consiste à émettre des signaux dans une gamme de fréquences qui perturbe ou même bloque les signaux utiles de communication.

Le brouillage des communications des objets connectés peut avoir des conséquences très graves, car les services offerts par les objets IOT sont basés sur la connectivité et les échanges sans fil. On cite quelques menaces du brouillage sur les objets connectés :

- Perturber la connectivité : le brouillage peut perturber les signaux de connexion utilisés par les objets IOT tels que la Wi-Fi, le Bluetooth et les réseaux cellulaires (3G, 4G, etc.), ce qui va empêcher les objets d'envoyer ou de recevoir les données.
- Compromettre la sécurité : en brouillant les communications entre les objets, l'attaquant peut manipuler l'objet et ainsi faire une action prohibée.
- Impact sur les applications délicates : le brouillage des objets qui sont utilisés dans des applications sensibles peut avoir des répercussions très graves, par exemple les objets IOT dans le domaine de la santé peuvent entrainer des pertes de vies humaines.

II.8.3 Blocage de mise en veille

La plupart des objets connectés dispose d'un mécanisme de mise en veille dans le but d'économiser de l'énergie et prolonger la durée de vie de la batterie ainsi que d'assurer une utilisation plus optimale de l'objet, cette menace consiste en l'empêchement du processus de mise en veille en envoyant des signaux externes, ce qui le garde toujours en fonctionnement et consomme de l'énergie.

Le fait que l'objet reste en marche, signifie qu'il continue à traiter, à envoyer et à recevoir des données, augmente le risque de s'exposer à d'autres attaques.

II.9 Conclusion

Dans ce chapitre, nous avons introduit l'historique de l'internet des objets, en suite, nous avons introduit des généralités sur l'IOT avec une définition de l'Internet des Objets, les domaines d'application, l'architecture et les équipements de l'IOT, nous avons introduit le modèle OSI et le modèle TCP-IP .Ensuite, nous avons présenté l'utilisation des FPGA pour les systèmes IOT et leurs avantages pour leur développement, puis, nous avons cité les menaces les plus connues sur les objets connectés, et nous avons terminé avec quelques détails sur les menaces retenues.

Le prochain chapitre est consacré la conception, la simulation et l'implémentation sur FPGA du système de détection des menaces sur les objets IOT.

Chapitre III : *Conception, simulation et implémentation du système de détection des menaces sur les objets IOT sur FPGA*

III.1 Introduction

Les objets de l'IOT sont vulnérables aux attaques, et comme qu'ils embarquent peu de puissance de calcul et ont une autonomie d'énergie plus au moins faible, ils sont dans la plupart des cas incapables de contrer ou même de détecter ces attaques. Il est donc intéressant de créer des mécanismes robustes avec ressources optimisées (espace mémoire, consommation d'énergie) pour la détection des menaces.

L'objectif de ce chapitre est de concevoir un système qui permet de détecter certaines menaces sur les objets connectés, les menaces considérées sont : la menace de déni de service, la menace de brouillage et la menace de blocage de la mise en veille. Le système est constitué d'une partie matérielle qui s'implémente sur FPGA, elle consiste en les trois blocs qui traitent les menaces sous forme de composants virtuels (IP), et d'une partie logicielle gérée par le processeur ARM A9. La structure du système, les étapes de création des différents blocs d'IP, les résultats de simulation obtenus ainsi que l'implémentation sur FPGA sont présentés en détail.

III.2 Structure matérielle et logicielle de notre système

Le système sur puce que nous allons concevoir est constitué d'une partie matérielle et une partie logicielle. La partie matérielle se compose des trois IP conçus pour la détection des menaces qui vont être implémentés sur la carte Zybo-Z7. La partie logicielle sert à gérer la partie matérielle et est exécuté par le processeur ARM A9 du circuit FPGA.

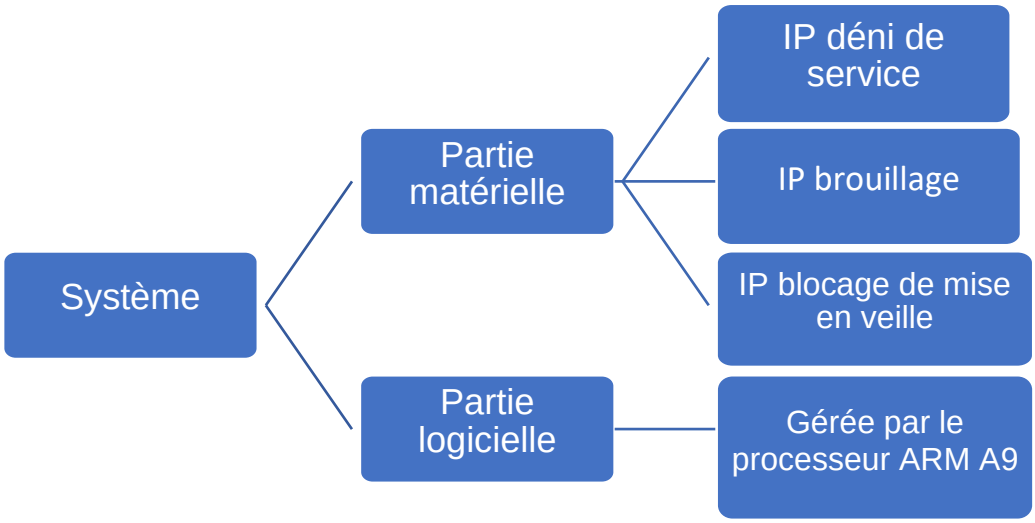


Figure III.14 : Structure matérielle et logicielle de notre système.

III.3 Etapes de création du projet dans l'outil Vivado

Dans cette section, nous présentons l'outil de développement Vivado. C'est un outil très puissant qui nous permet de décrire le fonctionnement des systèmes à l'aide d'un langage de description matérielle, dans notre cas le VHDL. Il nous permet également de créer des blocs de composants virtuels qui ont l'avantage d'être réutilisables, de gagner du temps de développement et de ne pas avoir à construire un système à partir de zéro. En outre, il permet aussi d'exécuter l'outil SDK qui permet de créer des

applications à l'aide du langage C ou C++ qui seront implémentées sur le processeur ARM A9 du circuit FPGA.

Les étapes de création de notre projet sont les suivantes :

III.3.1 Création des modules

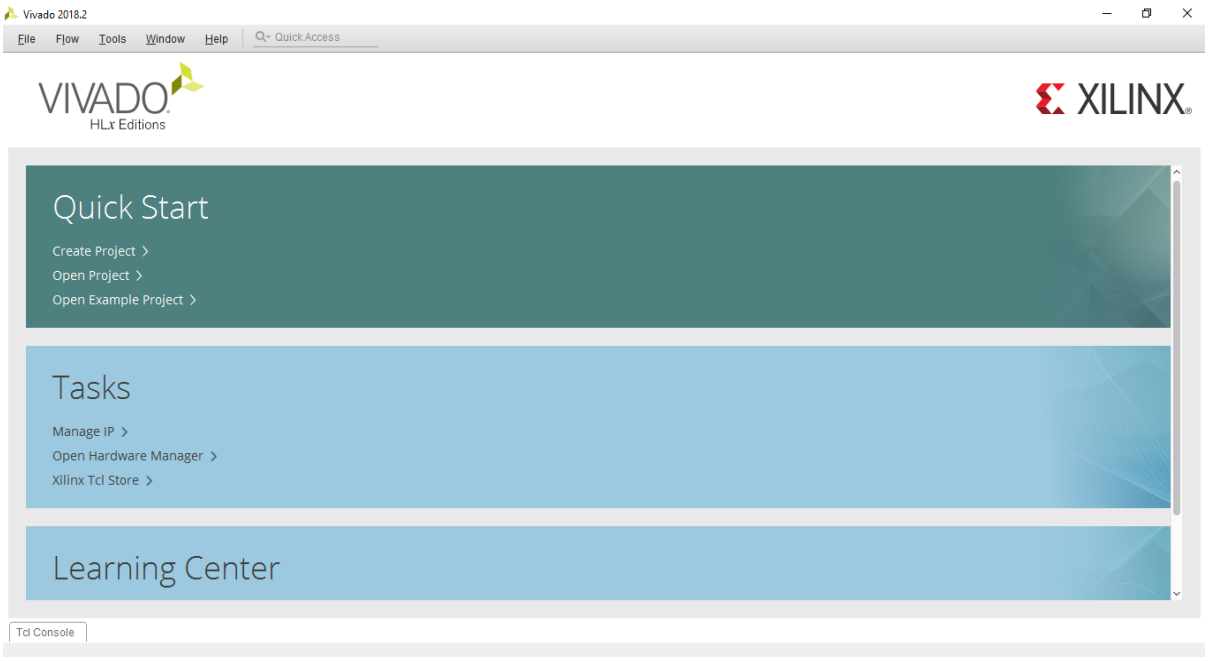


Figure III.15 : L'interface de l'outil Vivado.

Sous "Quick Start" nous sélectionnons "Create Project", ce qui fait apparaitre plusieurs fenêtres pour pouvoir créer le projet. Nous cliquons sur "Next" pour créer un projet.

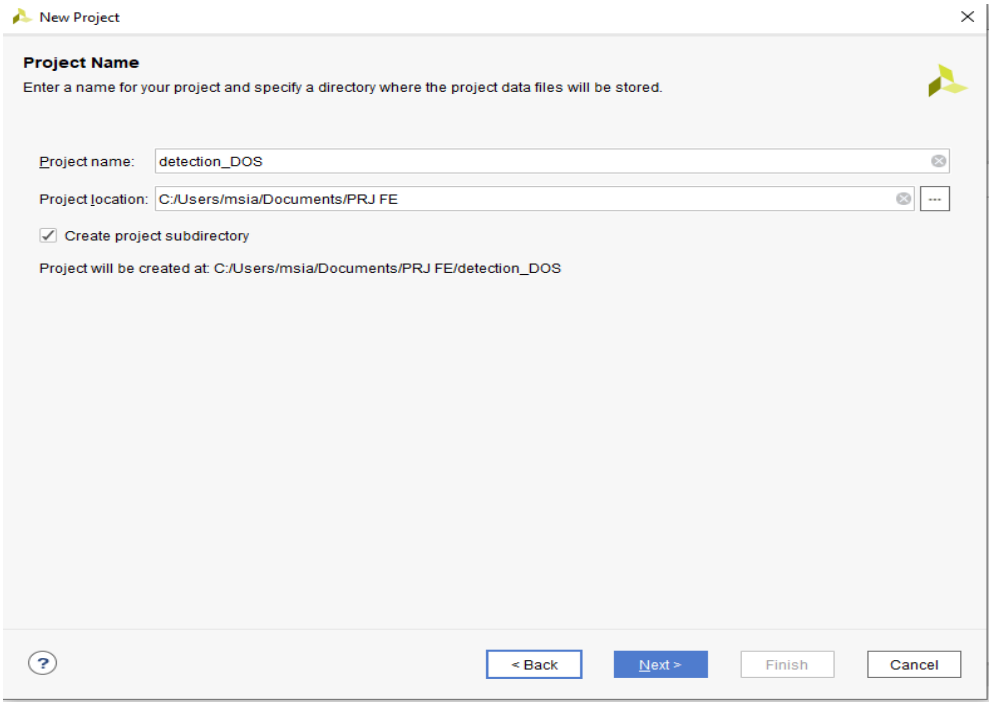


Figure III.16 : Etape 1 de la création du projet.

La fenêtre suivante permet de déterminer le type du projet créé. Pour notre projet, nous choisissons RTL (Registre Time Level).

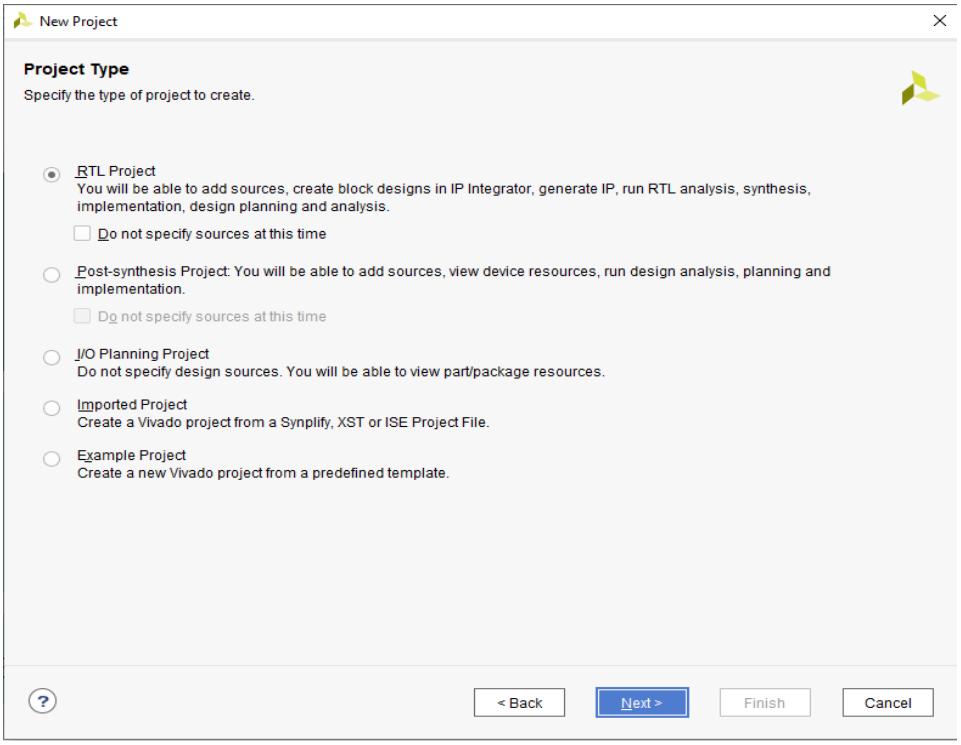


Figure III.17 : Etape 2 de la création du projet.

Les deux prochaines fenêtres sont pour l'ajout des ressources déjà existantes telles que les IP ou les codes sources.

Dans cette fenêtre, nous pouvons sélectionner la carte cible que nous voulons utiliser. Dans notre projet, c'est la carte Zybo-Z7 qui utilise un FPGA de la famille Zynq.

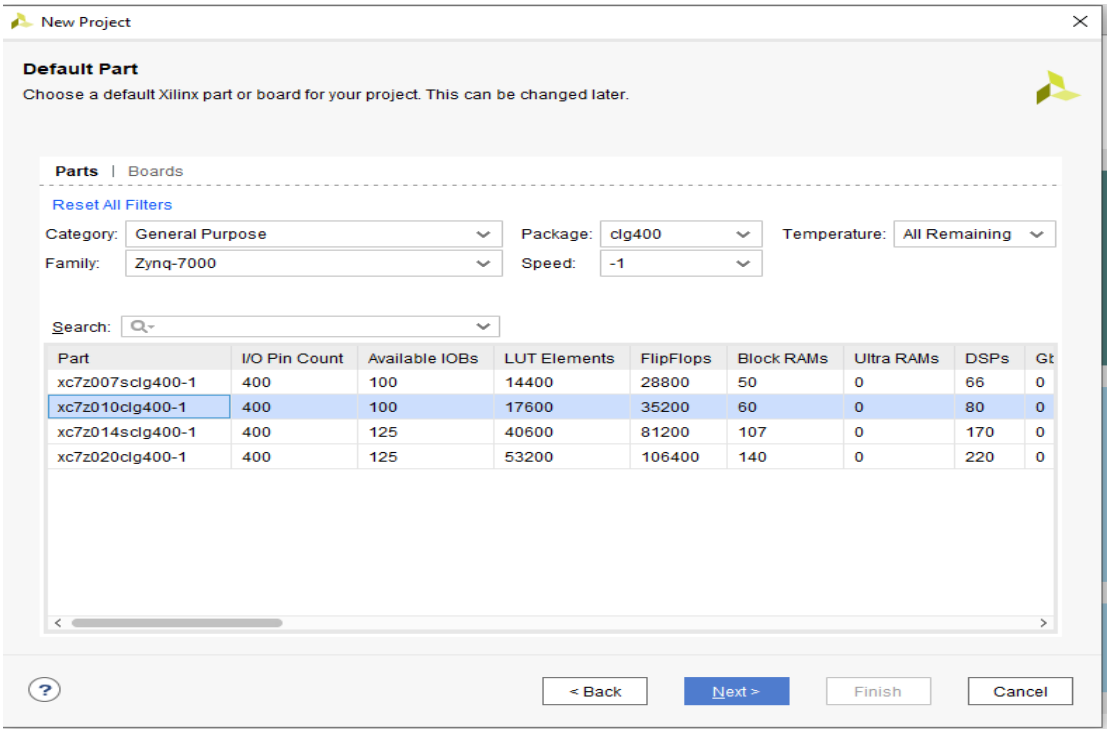


Figure III.18 : Etape 3 de la création du projet..

Dans la prochaine fenêtre, nous finalisons la création de notre projet.

L'interface de Vivado est affichée comme suit, elle contient quatre parties :

- Flow Navigator : cette section affiche la liste de tous les processus qu'on peut exécuter dans notre projet.
- Sources : dans cette section, tous les fichiers du projet sont répertoriés.
- Fenêtre de l'éditeur : cette section affiche le fichier qui est en cours de modification.
- Console : cette section est très importante, elle affiche les messages d'erreurs ou d'avertissements.

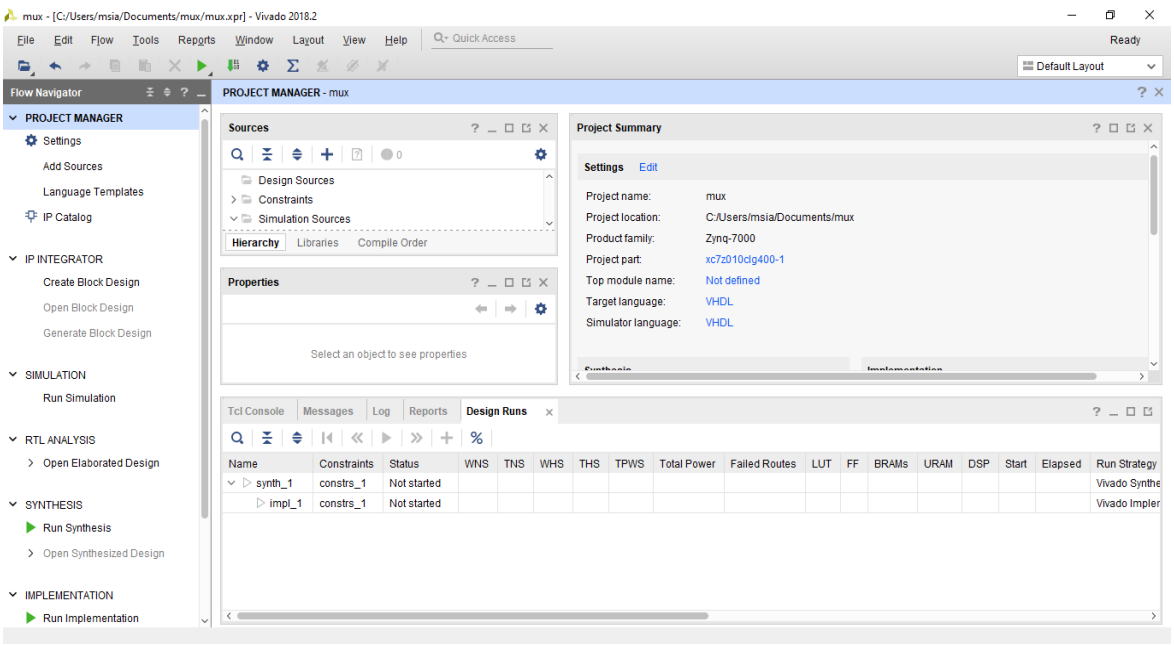


Figure III.19 : Etape 4 de la création du projet.

Dans la section Flow Navigator, nous sélectionnons "Add Sources" pour créer un module en VHDL ou ajouter un module déjà existant. Lorsque nous créons le module, nous définissons les entrées et les sorties :

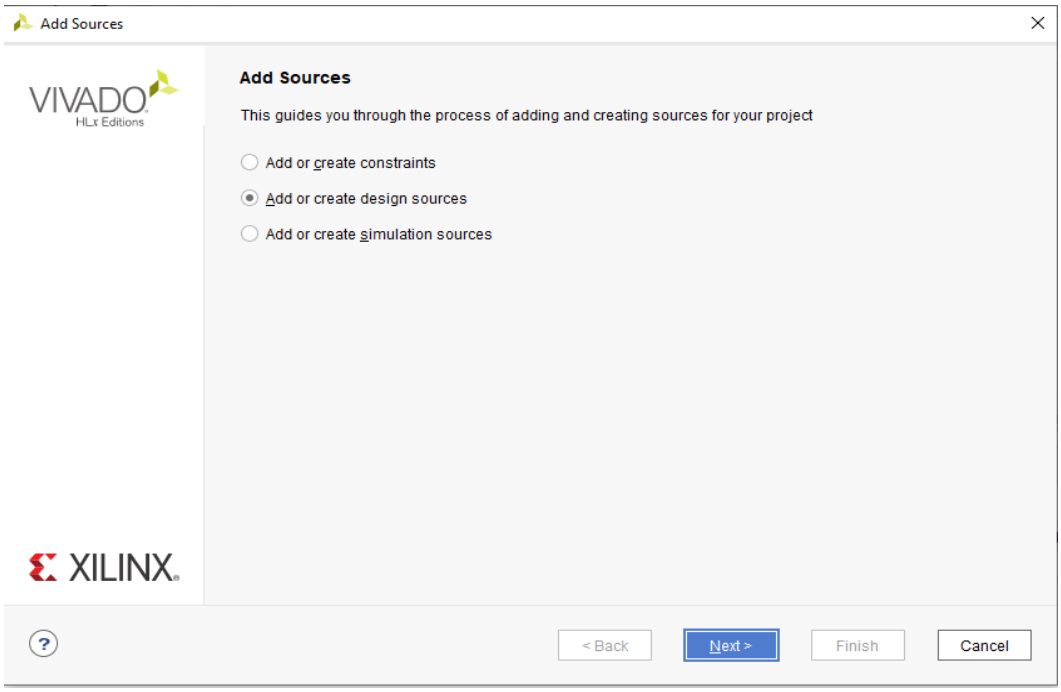


Figure III.20 : Etape 5 de la création du projet.

III.3.2 Création des IP

L'outil Vivado nous permet de créer des IP pour chaque module de notre système qui peuvent ensuite être réutilisés dans d'autres projets ce qui permet de gagner du temps lors du développement d'applications.

Nous sélectionnons "Tools" en suite "Create and Package new IP". Cette fenêtre apparait :

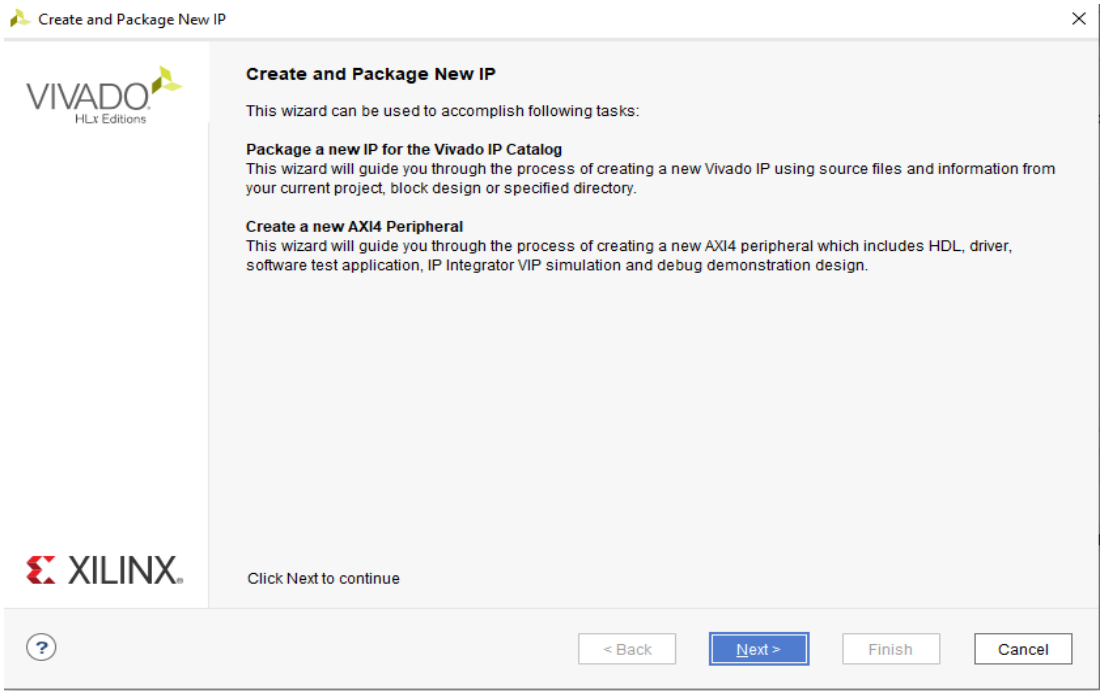


Figure III.21 : Etape 1 de la création d'un IP.

La prochaine étape permet de créer le package pour le module courant :

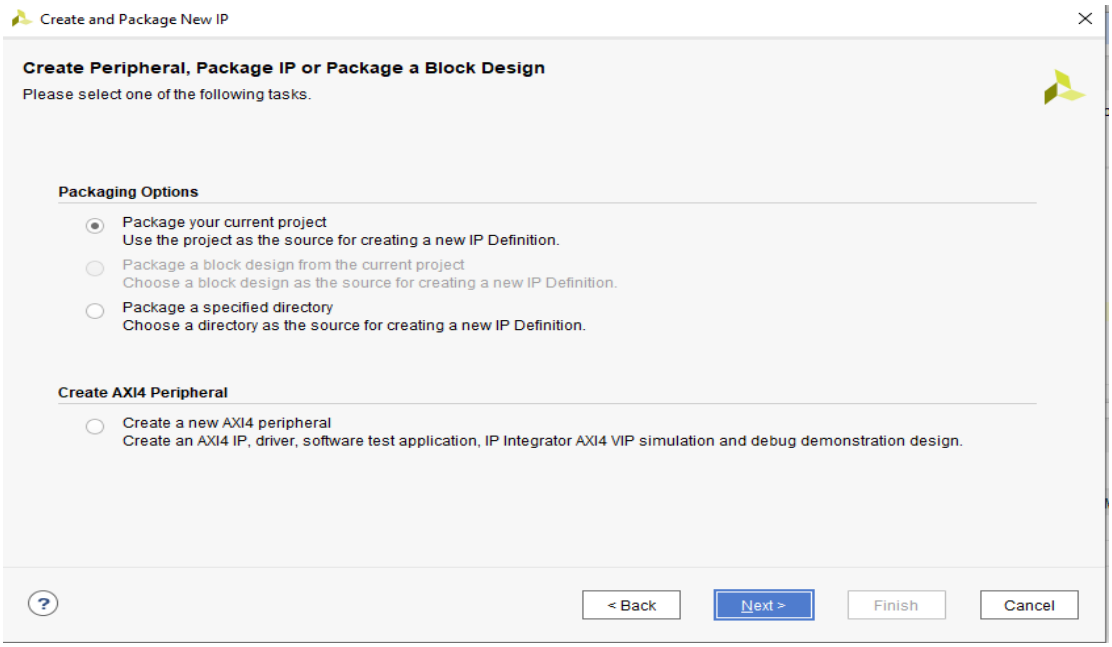


Figure III.22 : Etape 2 de la création d'un IP.

Les prochaines étapes consistent à terminer l'enregistrement de notre IP.

III.4 Les modules de notre système

Nous allons créer trois modules dans notre système, un module pour chaque menace ciblée :

- Un module pour la détection de la menace déni de service (DOS), en analysant la trame IP (Internet Protocol) afin de détecter des anomalies dans les paquets transmis.
- Un module pour la détection de la menace de brouillage. Ce module effectue une FFT (Fast Fourier Transform) sur les signaux afin de détecter les anomalies.
- Un dernier module pour détecter la menace de blocage de mise en veille. Ce module vérifie le niveau de batterie afin de détecter une consommation anormale de la batterie. Cependant, cette technique de détection doit être combinée avec une autre technique, car elle n'est pas fiable à 100%.

III.4.1 Menace de déni de service

Cet IP analyse de la trame IP (Internet Protocol), et plus précisément le champ TTL (Time To Live). Ce champ est composé de 8 bits, et contient un nombre qui se décrémente à chaque fois qu'un paquet passe par un routeur, s'il atteint 0 et n'atteint pas la destination, il se perd sur le réseau. Certain type d'attaque de dénis de service rend ce champ inchangeable (fixe) ce qui fait que le paquet reste sur le réseau et au bout d'un moment le réseau est submergé et devient lent ou même il se plante dans les cas extrêmes.

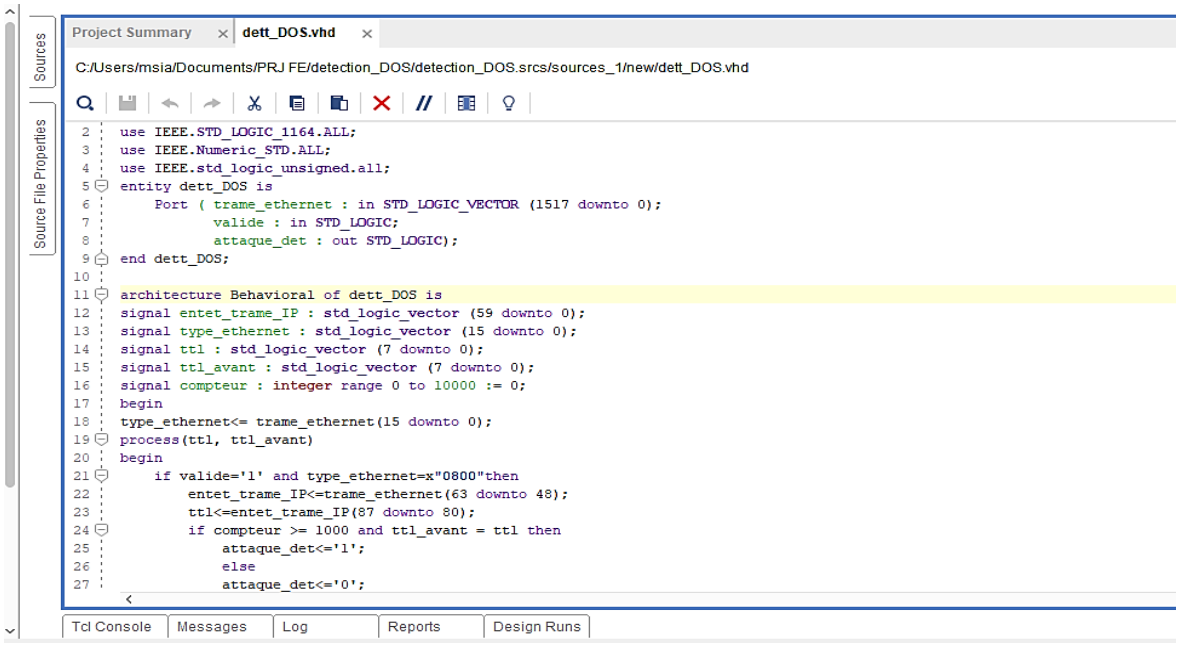


Figure III.23 : Code de détection de la menace de déni de service.

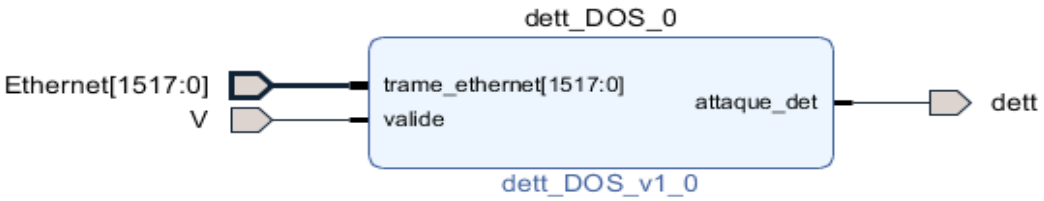


Figure III.24 : Création de l'IP de détection de la menace de déni de service.

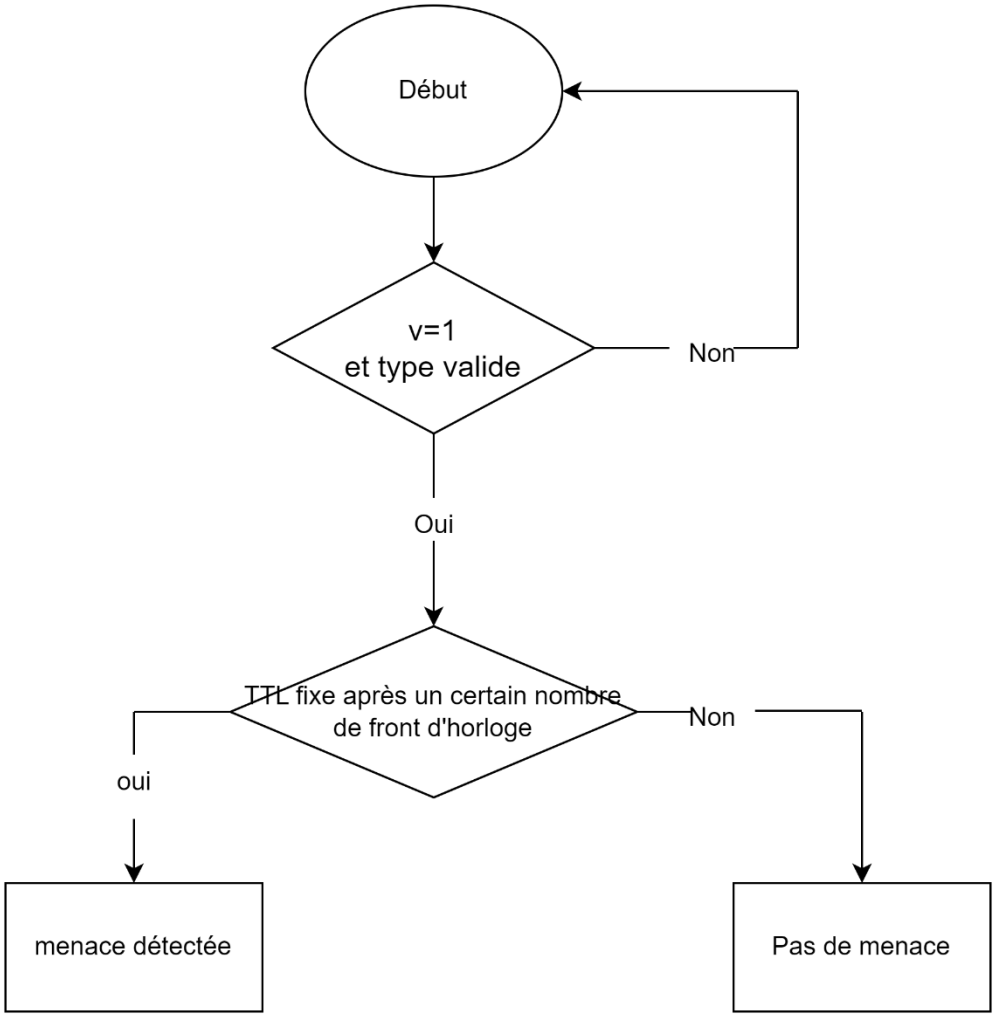


Figure III.25 : Fonctionnement de l'IP de détection de déni de service.

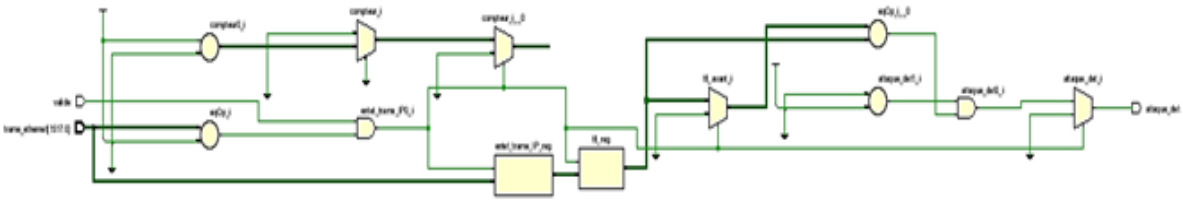


Figure III.26 : Schéma de l'IP de détection de la menace de déni de service.

Après la génération du schéma logique du bloc de détection de la menace de déni de service, Vivado nous donne la consommation des ressources sur le FPGA, notre bloc a consommé :

- 12 cellules logiques.
- 1523 ports d'entrées/sorties.
- 235 interconnexions.

Nous n'avons pas pu faire un banc de test pour le bloc de détection de la menace de dénis de service car nous avons en entrée une trame Ethernet qui est assez longue que nous ne pouvons pas l'initialiser à la main.

III.4.2 Menace de brouillage

Cet IP effectue l'opération de la FFT (Fast Fourier Transform), cette opération consiste à décomposer le signal en une série de fréquence, chacune avec une amplitude et une phase. La FFT est une variante de la TFD (Transformer de Fourier Discrète), mais plus rapide, elle permet donc un traitement en temps réel. La fonction mathématique de la FFT est :

$$x(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j*2*\pi*n*k}{N}} \qquad (1)$$

n : L'indice de l'échantillon (temporelle).

k : L'indice de l'échantillon (fréquentielle).

N : La taille de l'échantillon.

Nous aurons une fonction qui réalise la fonction de tournant qui divise les données en d'entrée en sous-groupes, la TFD est effectuée pour toutes les données. Cette opération exploite la propriété de la symétrie des séquences et permet d'optimiser le temps de calcul. La fonction mathématique de l'opération de tournant est :

$$w = e^{\frac{-2*\pi*i}{N}} \qquad (2)$$

w : La fonction de tournant.

Nous aurons une fonction papillon (Butterfly) qui combine les résultats des TFD effectuées sur des échantillons situés à des positions symétriques. Ces deux opérations sont répétées jusqu'à l'obtention de la FFT finale. La fonction mathématique de la fonction papillon est :

$$A' = A + w * B$$

$$B' = A - w * B \qquad (3)$$

Avec :

A et B : Les échantillons à combiner

A' et B' : Les échantillons après combinaison

Nous aurons donc, une fonction qui vérifie la présence ou l'absence d'un brouillage en comparant la dernière amplitude, car la dernière correspond à la fréquence la plus élevée et est considérée pour détecter des caractéristiques spécifiques dans un signal, car les signaux de brouillage sont caractérisés par des amplitudes très élevées.

Dans notre programme, nous avons eu besoin des fonctions cosinus, sinus et la fonction d'arrondissement, qui ne sont pas disponibles dans les packages standards de IEEE. Donc, nous avons encapsulé dans un package une fonction qui réalise le

cosinus et une autre fonction qui réalise le sinus que nous avons utilisé dans le programme de détection de la menace de brouillage grâce à l'instruction "Use".

- Le package pour le calcul du cosinus : la fonction permet de calculer une approximation de la valeur de cosinus de X grâce aux séries de Taylor selon un nombre d'itération prédéfini. La formule mathématique du développement limité de cosinus est :

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots \qquad (4)$$

- Le package pour le calcul du sinus : la fonction permet de calculer une approximation de la valeur de sinus de X grâce aux séries de Taylor selon un nombre d'itération prédéfini. La formule mathématique du développement limité de sinus est :

$$\sin(x) = 1 - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots \qquad (5)$$

- Le package pour le calcul de l'arrondissement : la fonction permet de faire l'arrondissement d'un nombre, en ajoutant ou en enlevant 0.5 (la fonction prend en compte si le nombre est positif ou négatif).

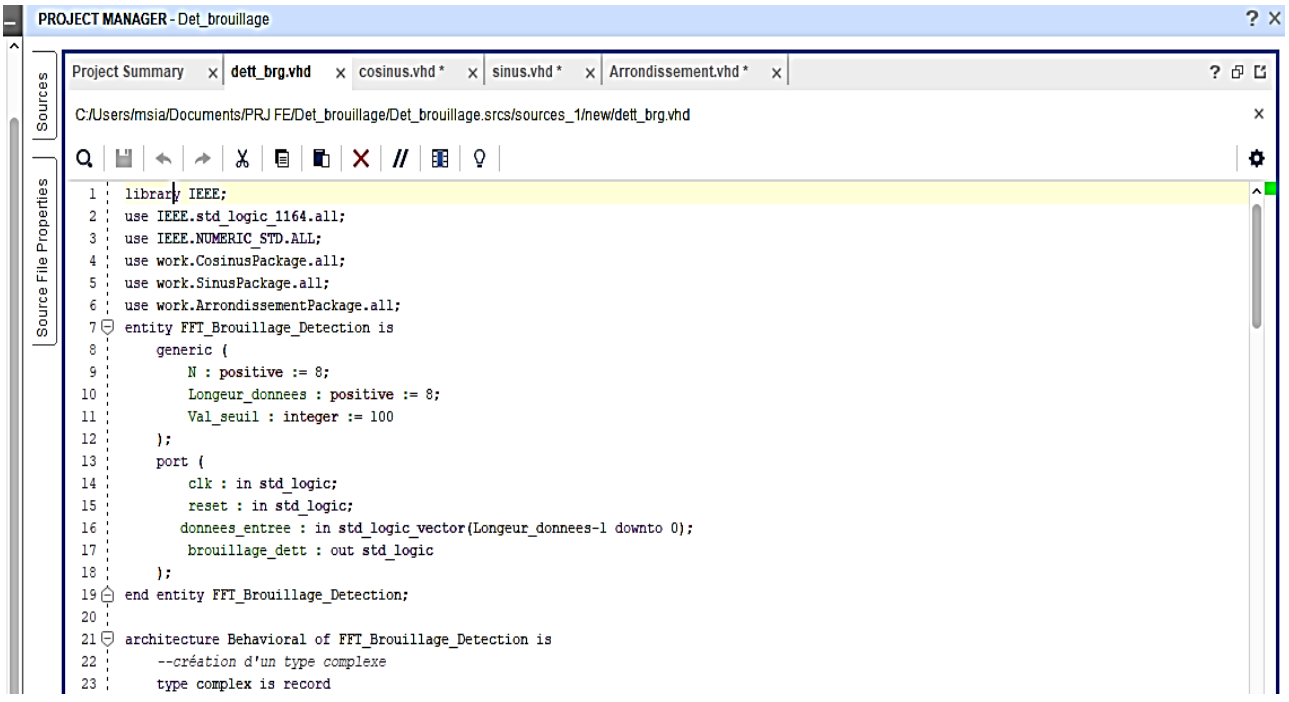


Figure III.27 : Code de détection de la menace de brouillage.

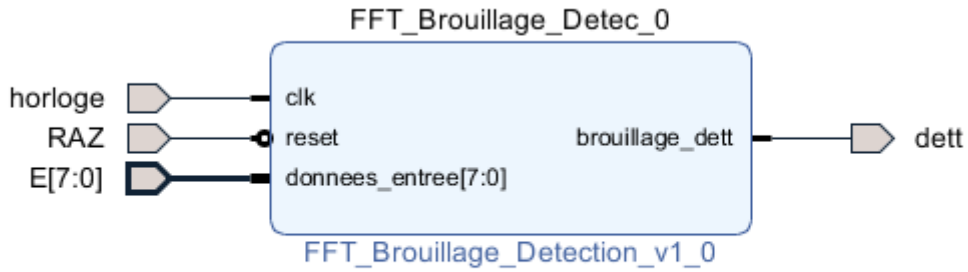


Figure III.28 : Création de l'IP de détection de la menace de brouillage.

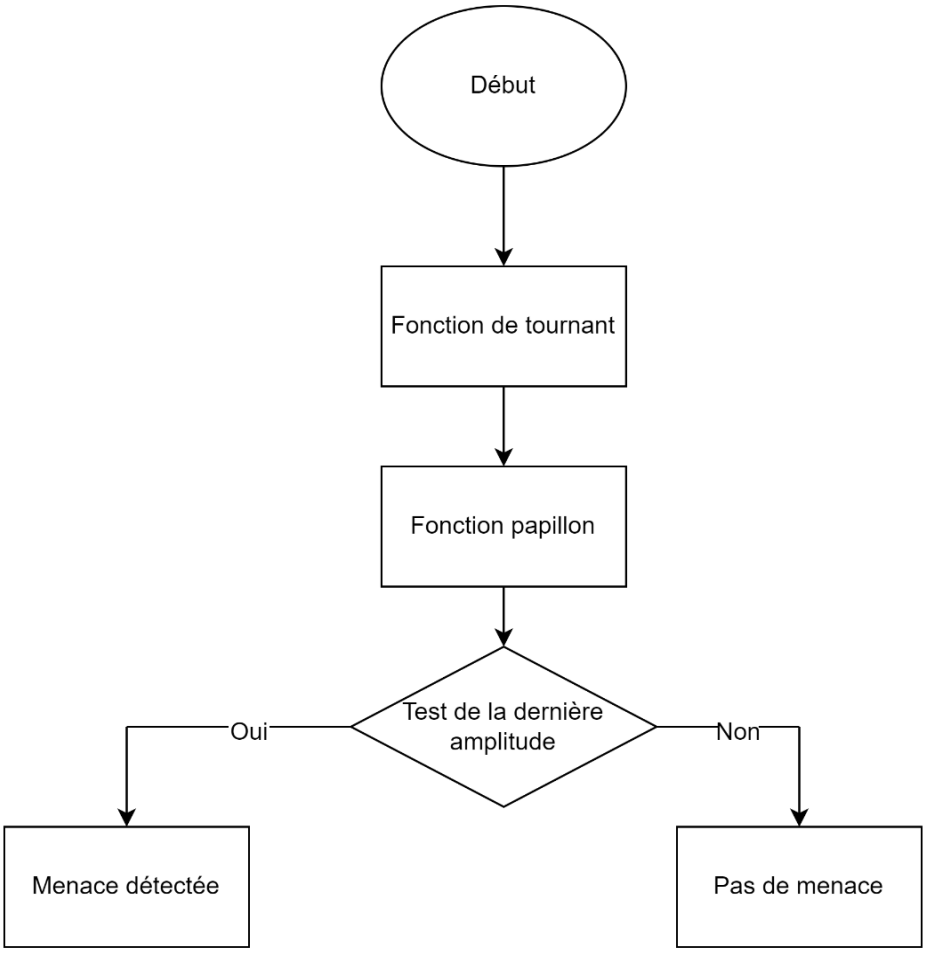


Figure III.29 : Fonctionnement de l'IP de détection de la menace de brouillage.

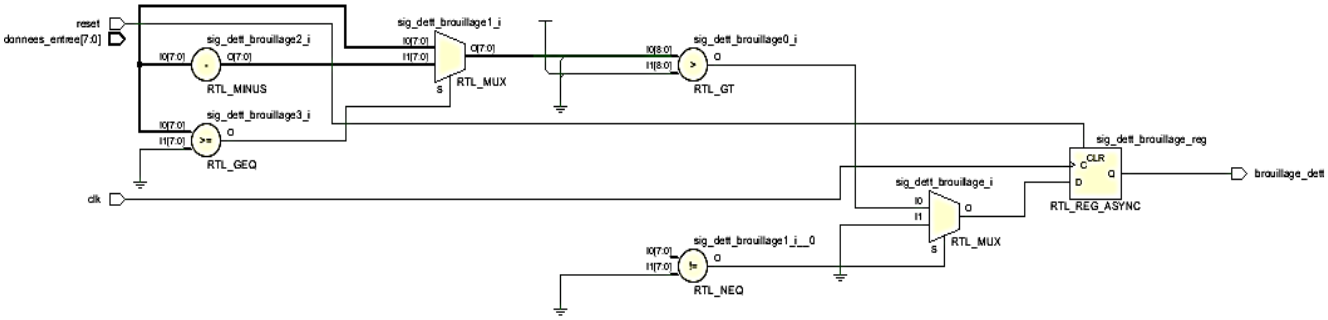


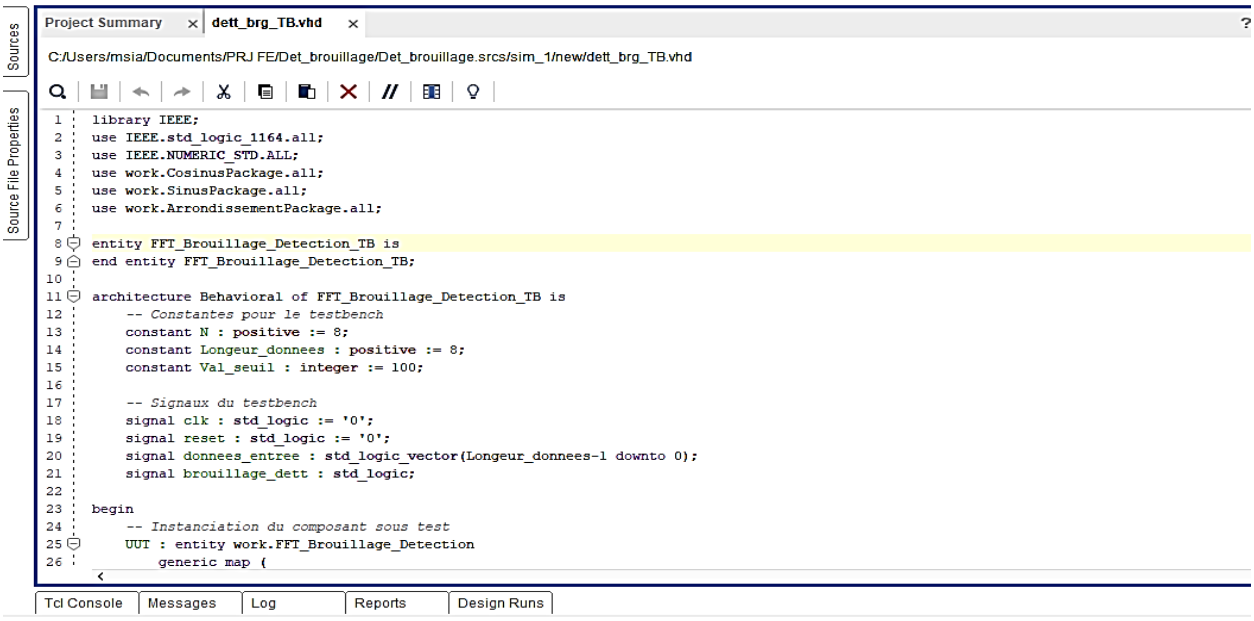
Figure III.30 : Schéma de l'IP de détection de la menace de brouillage.

Après la génération du schéma logique du bloc de détection de la menace de brouillage, Vivado nous donne la consommation de ressources sur le FPGA, notre bloc a consommé :

- 7 cellules logiques.
- 11 ports d'entrées/sorties.
- 33 interconnexions.

III.4.2.1 Test du bloc responsable de la détection de la menace de brouillage

Pour effectuer le test, nous écrivons le test bench en langage VHDL en attribuant des valeurs aux vecteurs de données d'entrée, un traitement se fait sur ces valeurs, et le signal de sortie indiquant la présence ou non d'une menace de brouillage affiche le résultat.




```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.NUMERIC_STD.ALL;
4  use work.CosinusPackage.all;
5  use work.SinusPackage.all;
6  use work.ArrondissementPackage.all;
7
8  entity FFT_Brouillage_Detection_TB is
9  end entity FFT_Brouillage_Detection_TB;
10
11 architecture Behavioral of FFT_Brouillage_Detection_TB is
12     -- Constantes pour le testbench
13     constant N : positive := 8;
14     constant Longeur_donnees : positive := 8;
15     constant Val_seuil : integer := 100;
16
17     -- Signaux du testbench
18     signal clk : std_logic := '0';
19     signal reset : std_logic := '0';
20     signal donnees_entree : std_logic_vector(Longeur_donnees-1 downto 0);
21     signal brouillage_dett : std_logic;
22
23 begin
24     -- Instanciation du composant sous test
25     UUT : entity work.FFT_Brouillage_Detection
26         generic map (
```

Figure III.31 : Banc de test du bloc de détection de la menace de brouillage.

Après la simulation du banc de test, nous obtenons un tableau avec les valeurs suivantes :

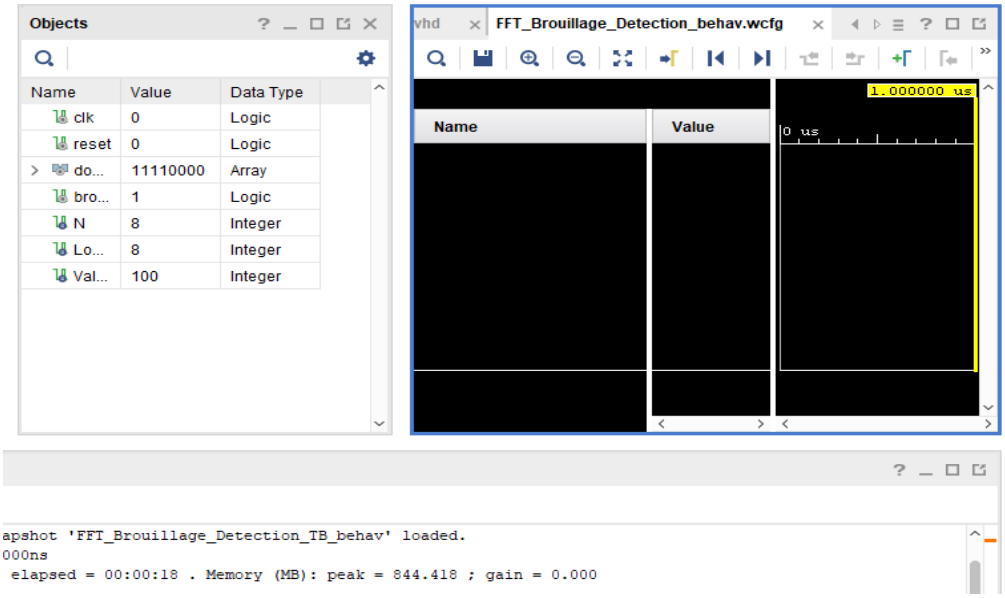


Figure III.32 : Les valeurs obtenues après la simulation de la menace de brouillage.

Les valeurs obtenues après l'exécution du banc de test nous montrent le fonctionnement du bloc IP. Un vecteur de 8 bits est donné en entrée, il passe par deux traitements, une fonction de tournant, ensuite, une fonction papillons. A l'issue des deux traitements, les amplitudes aux différentes fréquences sont déterminées en comparant la dernière valeur à un seuil prédéfini. Par conséquent, la sortie prend 1 s'il y a une menace et 0 si non.

III.4.3 Menace de blocage de la mise en veille du système

Cet IP réalise une opération de traitement du niveau de batterie de notre objet. En effet, les attaques épuisent la batterie des objets IOT et donc les mettent hors service. Nous définissons un seuil auquel nous comparerons le niveau de la batterie de notre objet, s'il est en dessous du seuil, une menace sera détectée.

Le programme fonctionne comme suit :

- Nous définissons une valeur indiquant le seuil de la batterie.
- Nous recevons le niveau de batterie de l'objet.
- Nous le comparons à une valeur seuil prédéfinie.
- Si le niveau de la batterie est inférieur à cette valeur (seuil), une menace est détectée.
- S'il est supérieur, une menace n'est pas détectée.

Project Summary | blockage_miseenveille.vhd

C:/Users/msia/Documents/TP/PRJ FE/project_1/project_1.srcs/sources_1/new/blockage_miseenveille.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity dett_blockage is
4      Port ( nv_batterie :in STD_LOGIC_VECTOR (6 downto 0);
5             clk : in STD_LOGIC;
6             blocage : out STD_LOGIC);
7  end dett_blockage;
8  architecture Behavioral of dett_blockage is
9  constant seuil_batterie: std_logic_vector(6 downto 0):="0110010";
10 signal blc: STD_LOGIC;
11 begin
12 process (clk, nv_batterie)
13 begin
14 if rising_edge(clk) then
15     if nv_batterie<seuil_batterie  then
16         blc<='1';
17     else
18         blc<='0';
19     end if;
20 end if;
21 end process;
22 blocage<= blc;
23 end Behavioral;
```

Figure III.33 : Code de détection de la menace de blocage de mise en veille.

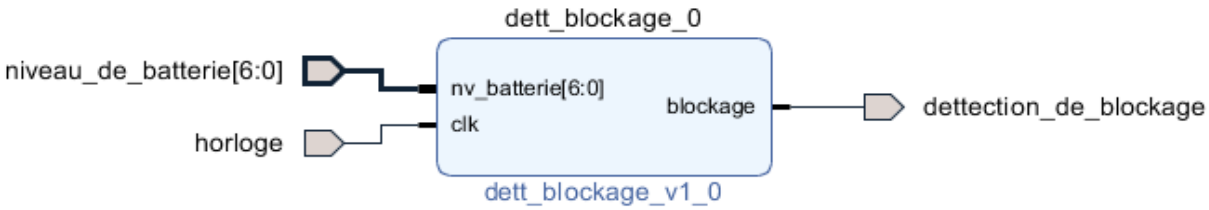


Figure 34 : Création de l'IP de détection de la menace de blocage de mise en veille.

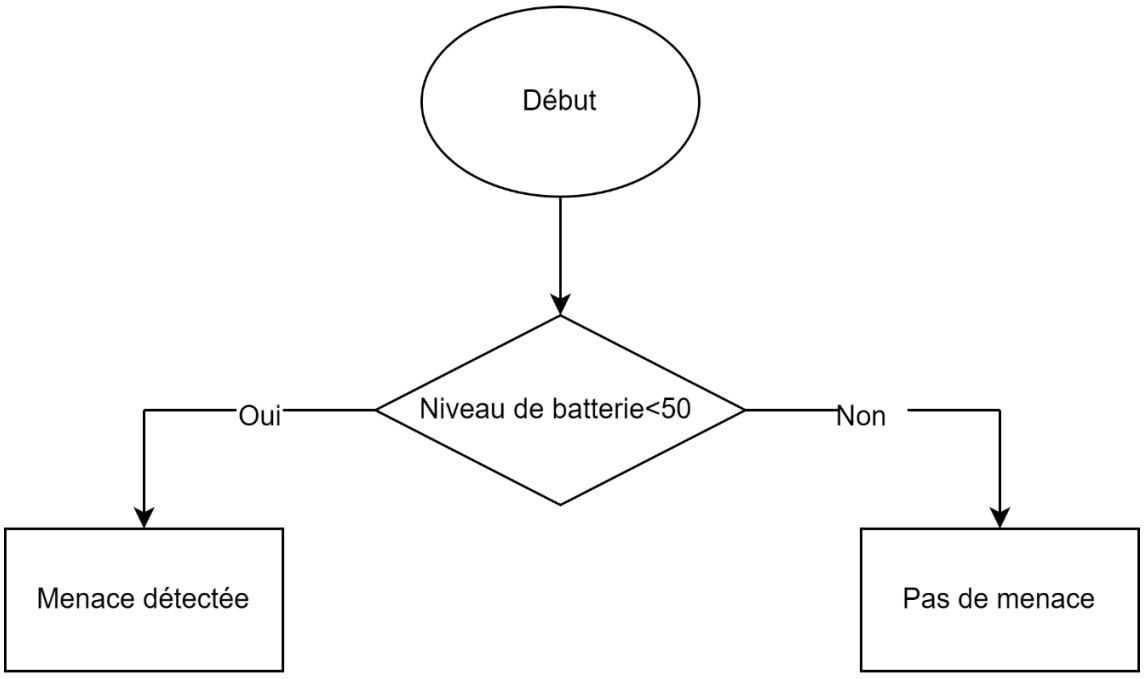


Figure III.35 : Fonctionnement de l'IP de blocage de mise en veille

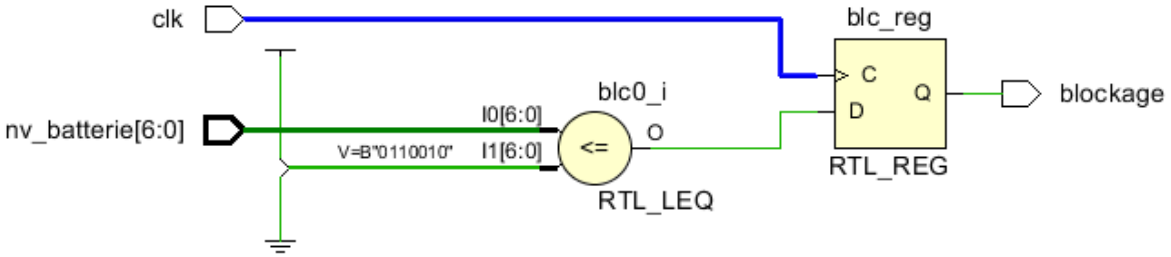


Figure III.36 : Schéma de l'IP de détection de la menace de blocage de mise en veille.

Après avoir généré le schéma logique du bloc de détection de la menace de blocage de mise en veille, Vivado nous donne la consommation de ressources sur le FPGA, qui est consommée par note bloc :

- 2 cellules logiques.
- 9 ports d'entrées/sorties.
- 12 interconnexions.

III.4.3.1 Test du bloc responsable de la détection de la menace de blocage de mise en veille

Pour effectuer le test, nous écrivons un banc de test en langage VHDL en insérant des valeurs aléatoires pour le niveau de batterie et en observant les changements sur le signal indiquant le blocage en sortie, le tout est synchronisé par un signal horloge.

blockage_miseenveille.vhd | brouillage_TB.vhd | dett_blockage_tb_behav.wcfg

C:/Users/msia/Documents/PRJ FE/project_1/project_1.srcs/sim_1/new/brouillage_TB.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity dett_blockage_tb is
5   end dett_blockage_tb;
6
7   architecture Behavioral of dett_blockage_tb is
8       component dett_blockage is
9       Port (
10          nv_batterie : in STD_LOGIC_VECTOR (6 downto 0);
11          clk : in STD_LOGIC;
12          blockage : out STD_LOGIC
13      );
14      end component;
15
16      signal nv_batterie : STD_LOGIC_VECTOR (6 downto 0):="1100100";
17      signal clk : STD_LOGIC := '0';
18      signal blockage : STD_LOGIC := '0';
19  begin
20      -- Instanciation de l'unité sous test
21      uut : entity work.dett_blockage
22          port map (
23              nv_batterie => nv_batterie,
24              clk => clk,
25              blockage => blockage
26          );
```

Figure III.37 : Le banc de test du bloc de détection de la menace de blocage de mise en veille.

Après la simulation du banc de test, nous obtenons les signaux suivants :

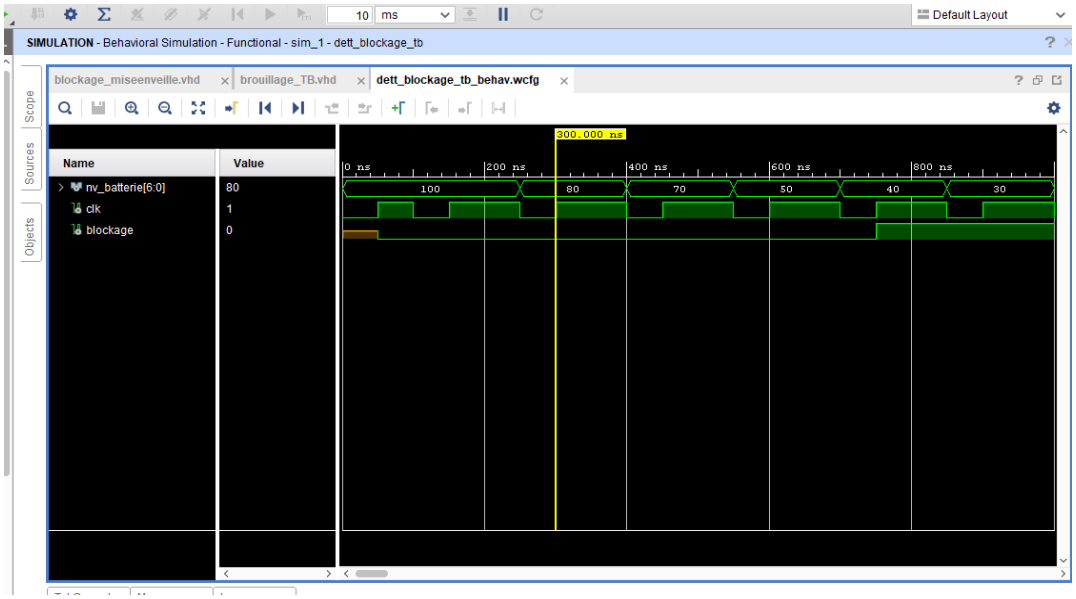


Figure 38 : Les signaux obtenus après la simulation de la menace de blocage de mise en veille.

Les signaux obtenus après la simulation, montrent le fonctionnement attendu du bloc. En effet, nous pouvons voir que lorsque le niveau de la batterie est supérieur à 50% le signal de sortie indique l'absence d'une menace (la sortie est à 0), lorsque le niveau de batterie est inférieur à 50% et le signal d'horloge a un front montant, le signal de sortie indique la présence d'une menace (la sortie est à 1).

III.4.3.2 Implémentation sur FPGA du bloc de détection du blocage de mise en veille

A. Fichier de contraintes

Le fichier des contraintes permet d'établir la liaison entre le circuit synthétisé par le code VHDL et les pins physiques du circuit FPGA.

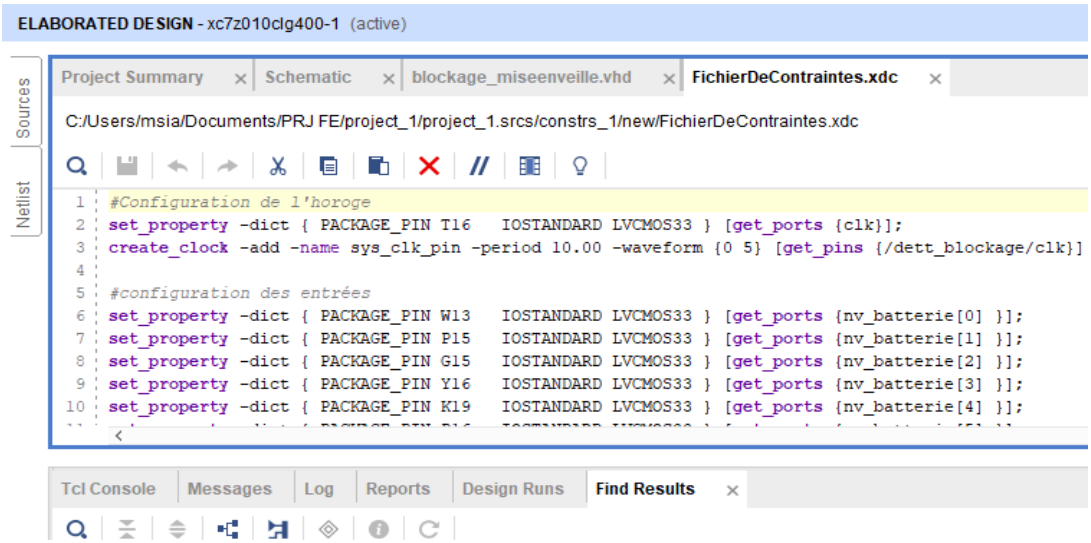


Figure III.39 : Fichier de contrainte.

B. Etape de synthèse

Elle consiste en une étape de synthèse logique qui consiste à traduire la représentation en langage HDL en une Netlist, puis, l'étape de correspondance, qui traduit la Netlist en des composants sur la carte, et la dernière étape de synthèse est le placement/routage qui consiste à interconnecter les composants du circuit entre eux sur FPGA.

C. Etape d'implémentation

Après l'étape d'implémentation, nous obtenons le rapport de l'énergie consommée par le système et les ressources matérielles exploitées sur le FPGA.

Sur la figure ci-dessous, nous pouvons voir que toute l'énergie est consommée par la partie logique programmable, majoritairement (98%) par les blocs d'entrées/sorties, le reste est partagé entre les blocs logiques et les signaux.

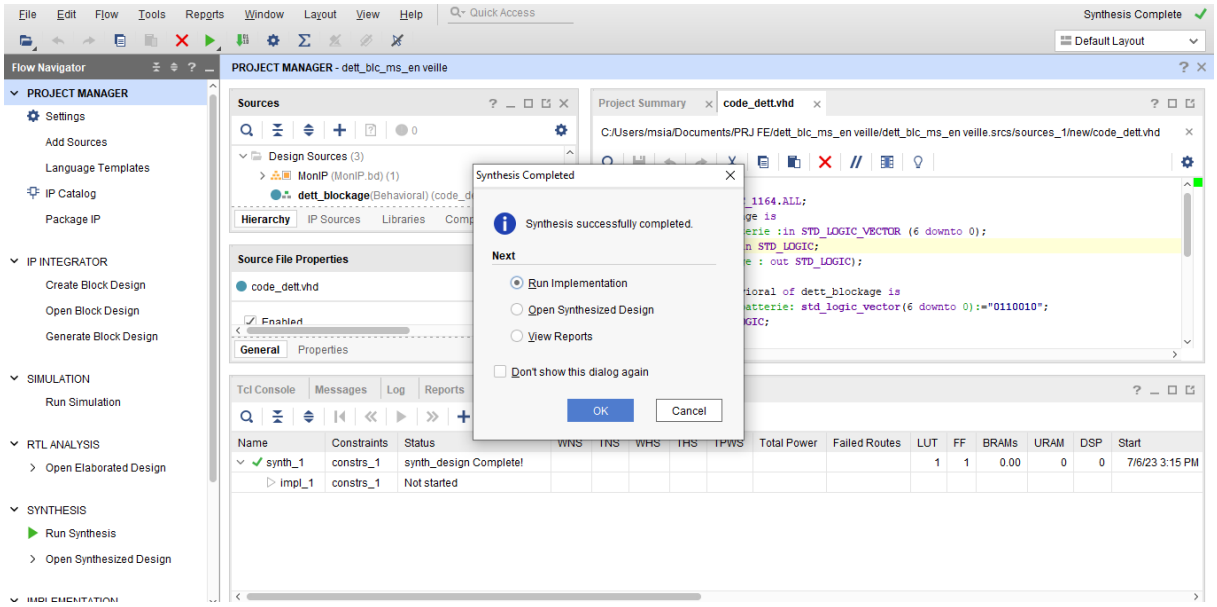


Figure III.40 : Fin de l'étape de synthèse.

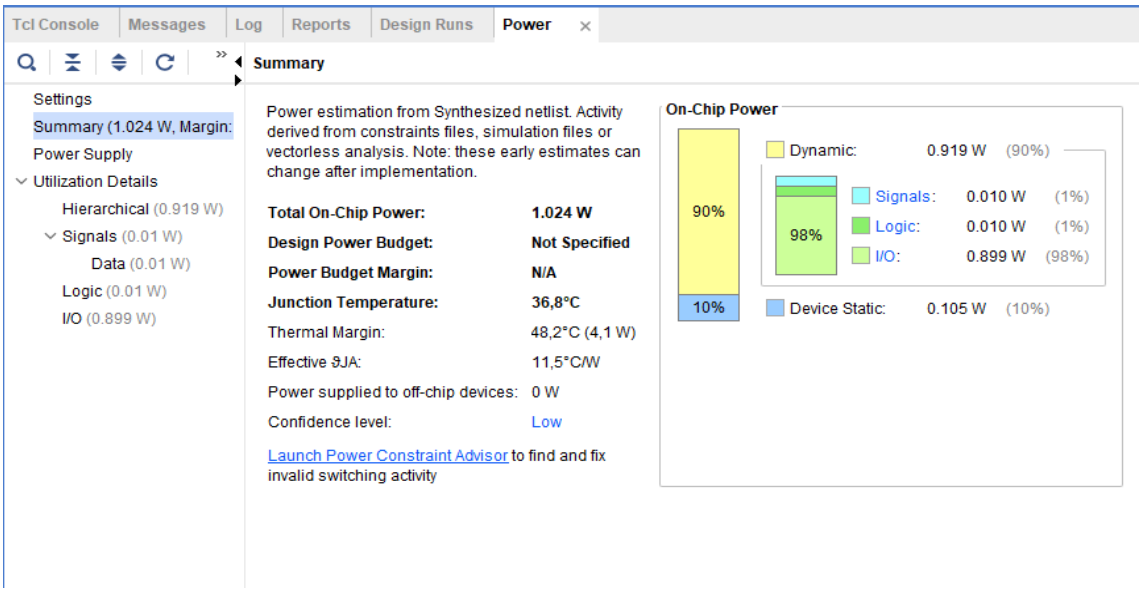


Figure III.41 : Rapport énergétique généré après l'étape de synthèse.

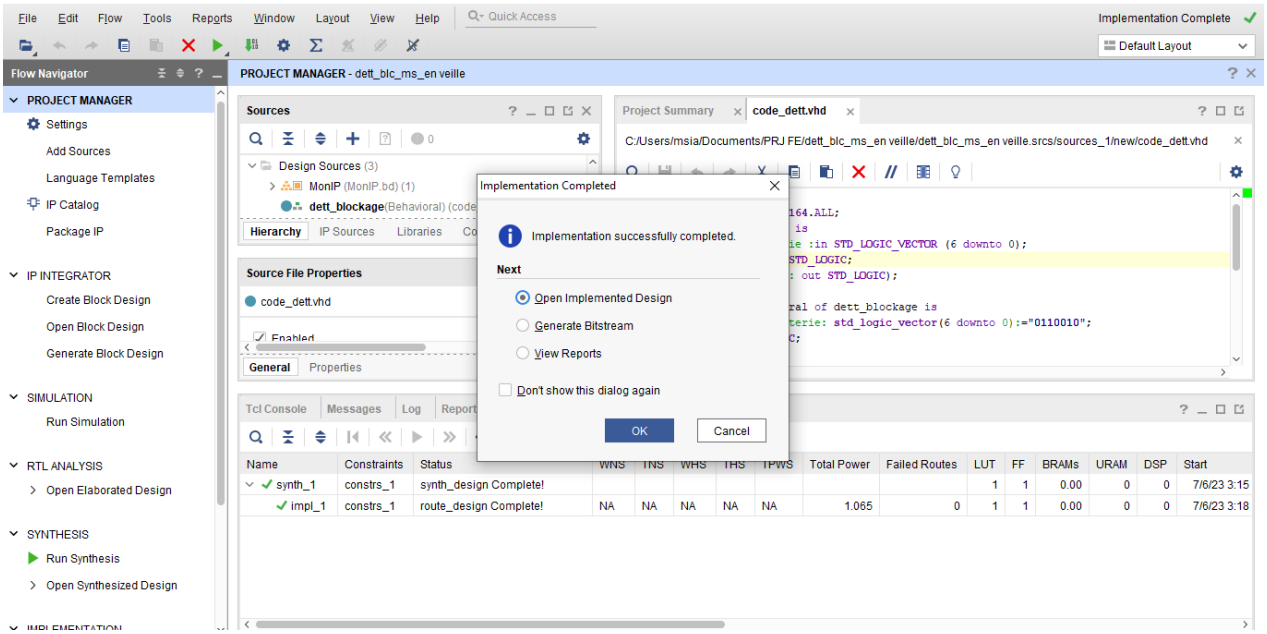


Figure III.42 : Fin de l'étape d'implémentation.

Dans la figure ci-dessous, nous pouvons voir les ressources matérielles utilisées par notre module IP. Les blocs colorés représentent les blocs de cellules logiques, les lignes en vert représentent les interconnexions avec les entrées/sorties.

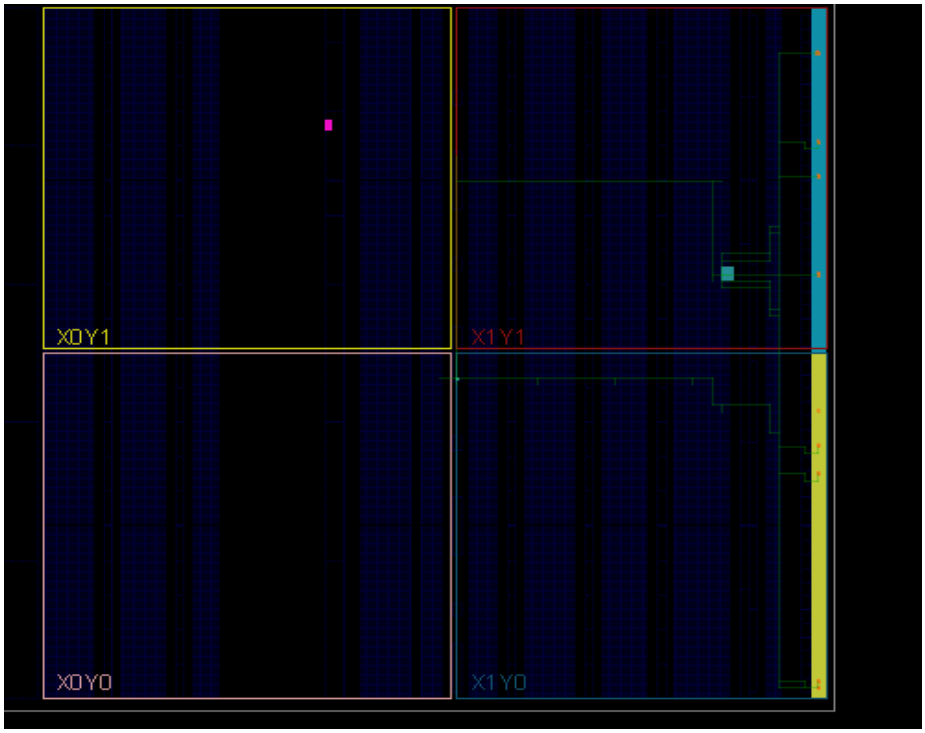


Figure III.43 : Les ressources matérielles utilisées.

D. Génération du fichier binaire (bitstream)

La dernière étape consiste à générer le fichier binaire (bitstream), que nous chargeons dans la carte FPGA pour la configuration de notre application.

Les switches et les boutons poussoirs de la carte sont utilisés comme des entrées et les LED comme des sorties. Les valeurs des entrées indiquent le niveau de la batterie, le module d'IP effectue le traitement et envoie le signal de sortie sur la LED, si elle est allumée (état logique 1) une menace de blocage est détectée, si elle est éteinte (état logique 0), aucune menace n'est détectée.

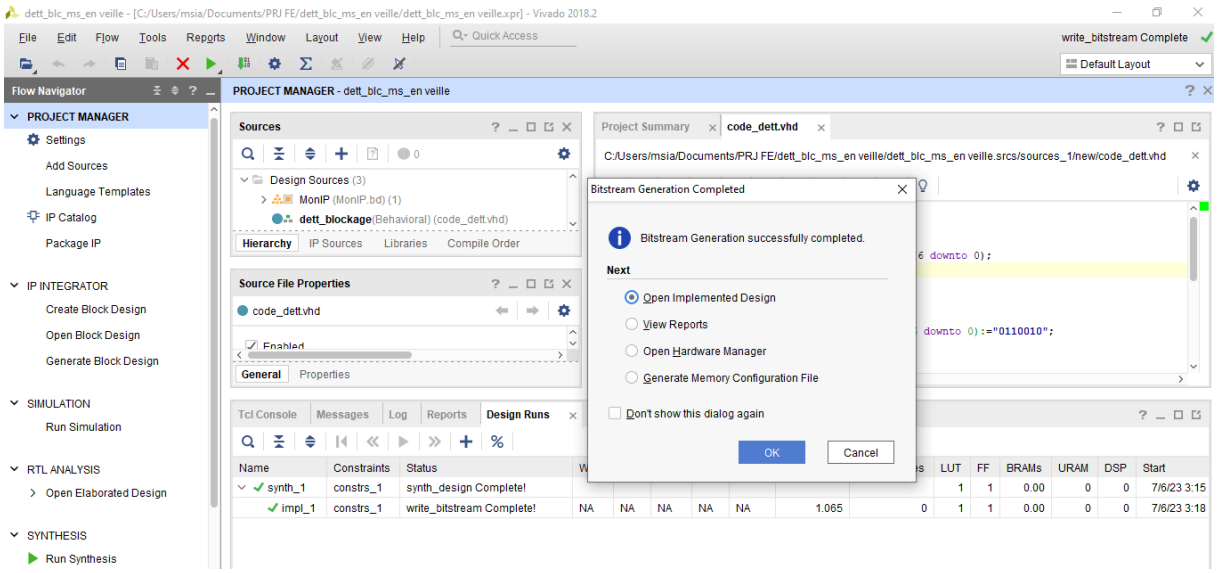


Figure III.44 : Génération du fichier binaire (bitstream).

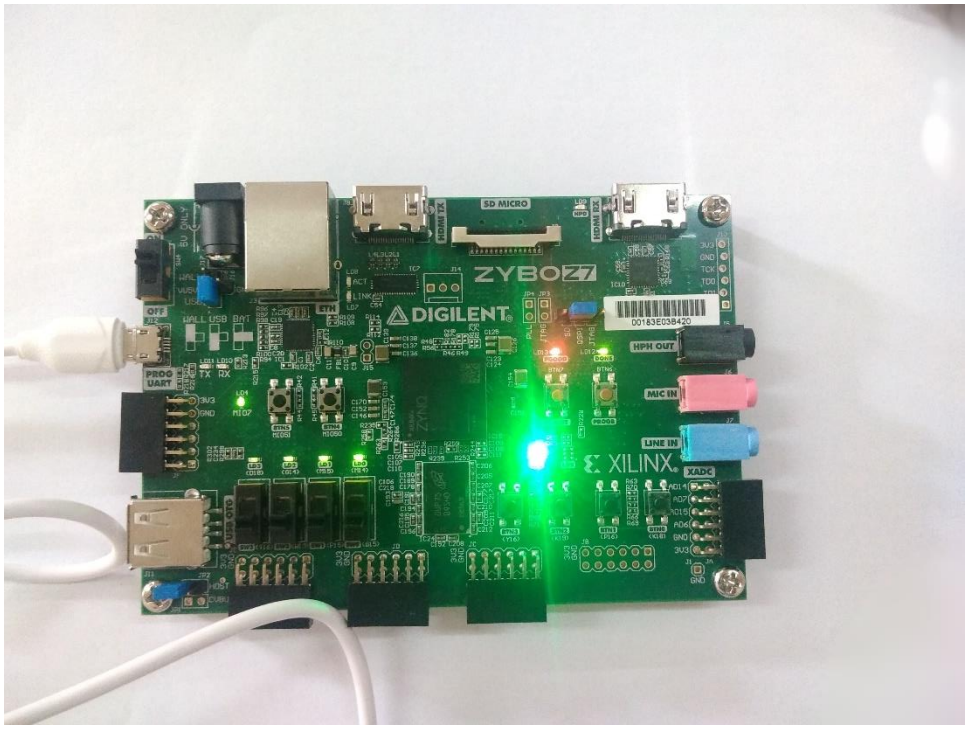


Figure III.45 : Implémentation sur la carte FPGA.

Une fois, la carte FPGA est branchée au PC, nous chargeons le fichier binaire généré (bitstream) dans le circuit FPGA, puis nous configurons les switches et les boutons comme entrées indiquant le niveau de batterie en testant plusieurs valeurs possibles entre 0 et 100. La sortie du bloc est représentée par la LED qui indique la présence ou l'absence d'une menace. Si la LED est éteinte, il n'y a pas de blocage et si elle est allumée il y a un blocage.

III.5 Structure globale du système de détection du blocage de mise en veille

Cette partie s'implémente sur le processeur ARM A9 de la carte Zybo Z-7, elle va se charger de gérer le bloc de la partie matérielle, il s'agit du bloc de détection de la menace de blocage de mise en veille.

III.5.1 Le schéma du module

Dans ce schéma, nous retrouvons plusieurs blocs IP que nous avons ajoutés afin de faire communiquer notre IP de détection avec le processeur ARM A9 et les entrées/sorties de la carte. Les blocs d'IP qu'on retrouve sont :

- IP de détection de la menace : c'est le composant virtuel que nous avons développé et testé dans la partie matérielle du projet. Cet IP est utilisé pour détecter la menace de blocage de mise en veille en testant le niveau de la batterie de l'objet, s'il est en dessous d'une valeur seuil, la menace est détectée.
- L'IP AXI interconnecte : c'est un bloc IP fourni par XILINX et utilisé pour organiser les échanges des données entre les maitres et les esclaves AXI. Voici quelques utilisations du module AXI interconnecte :
    - Gérer les communications entre les maitres et les esclaves.
    - Multiplexages des signaux en provenance des maitres vers les esclaves appropriés et démultiplexages dans le sens inverse.
    - Gérer les retards des signaux causés par les interconnexions.
    - Offrir une configuration flexible.
- L'IP AXI GPIO : c'est un bloc IP fourni par XILINX qui crée une interface GPIO et permet à notre IP de détection du blocage de mise en veille d'accéder aux entrées/sorties de la carte. Voici quelques utilisations du module AXI GPIO :
    - Contrôle des périphériques externes.
    - Interconnexion avec d'autres blocs logiques.
    - Interaction avec le processeur ce qui facilite la communication entre la partie logique programmable et la partie processeur.

- L'IP système ZYNQ : il s'agit de l'IP système configuré par XILINX, qui permet à notre IP de détection d'accéder au processeur ARM A9 et aux ressources telles que les mémoires, il offre plusieurs utilisations :
    - Accès aux deux cœurs du processeur ARM A9.
    - Accès à la mémoire.
    - Une multitude d'interfaces périphériques telles que : UART et I2C.
    - Gérer le démarrage du système.
    - S'occuper de la gestion de l'énergie en gérant l'alimentation, le mode de mise en veille et le mode économie d'énergie.

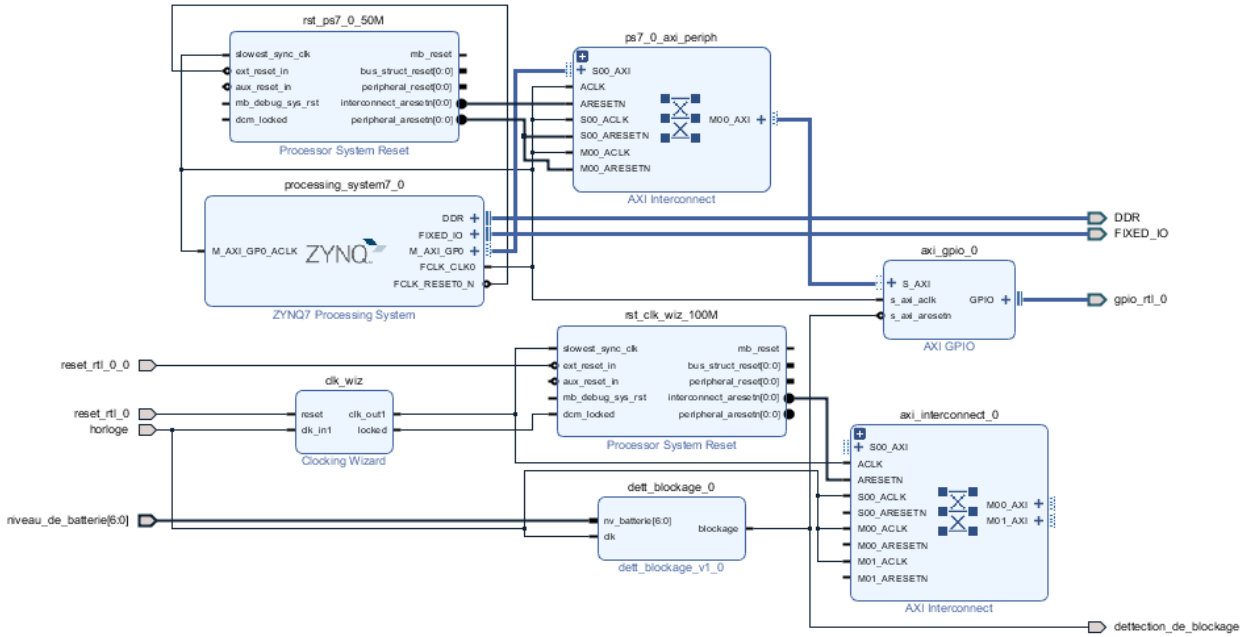


Figure III.46 : Architecture matérielle complète du système de détection de la menace de mise en veille.

III.5.2 Génération du wrapper

Un Wrapper est décrit dans langage de description matérielle (VHDL dans notre cas), il s'agit d'une entité qui encapsule un autre composant, il fournit une interface pour que les entités communiquent entre elles en incluant des signaux d'horloge, des signaux synchronisation et des signaux de validation.

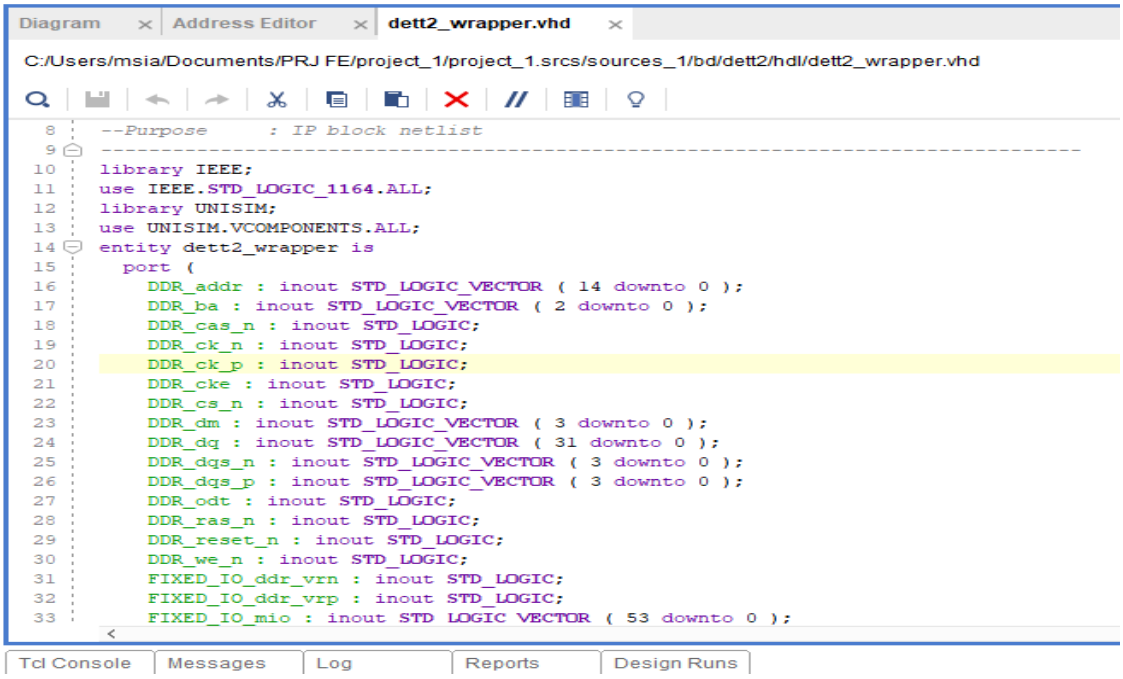


Figure III.47 : Le wrapper.

III.5.2 Synthèse et implémentation

Après la génération du wrapper, nous effectuons une étape de synthèse ainsi qu'une étape d'implémentation qui donne un rapport de la consommation des ressources de la carte FPGA en termes d'énergie et de matériel.

Nous pouvons voir sur la figure ci-dessous les ressources matérielles utilisées par notre système. Dans la partie en bleu, il s'agit des ressources du FPGA. Si nous effectuons un zoom nous pouvons voir les LUT, les bascules et les blocs d'entrées/sorties. Dans la partie en vert, il s'agit des ressources du processeur ARM A9.

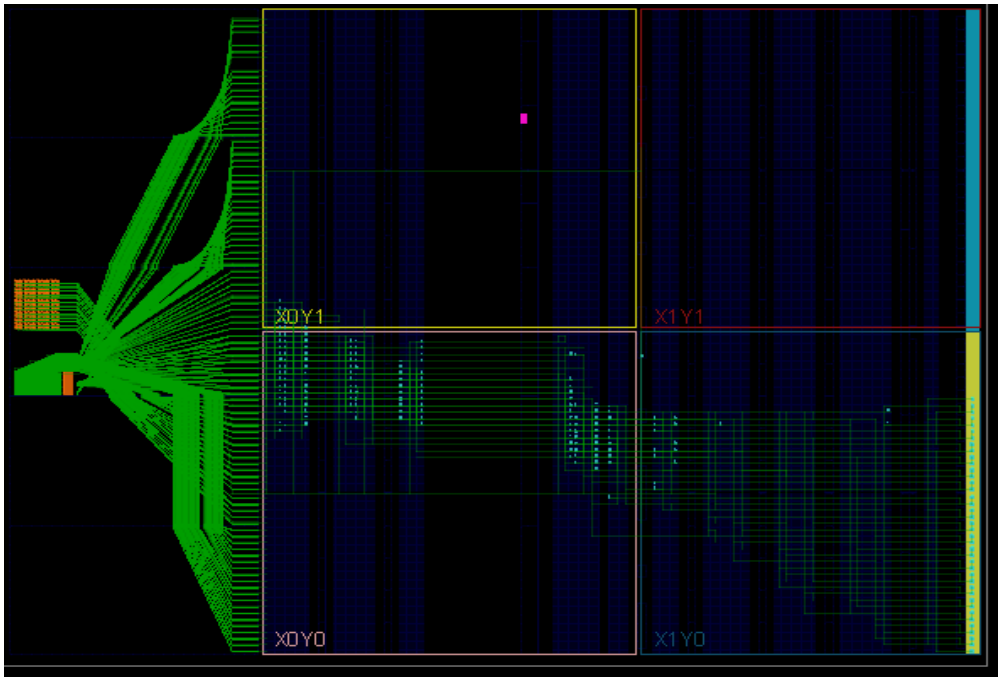


Figure III.48 : Les ressources matérielles utilisées.

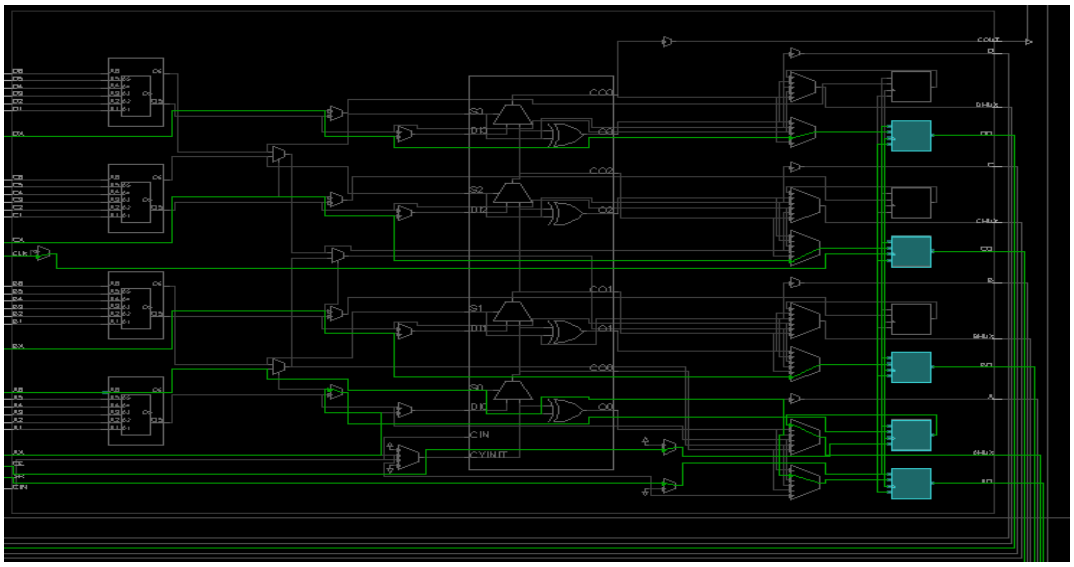


Figure III.49 : Les CLB et les interconnections de la partie FPGA.

Dans la figure ci-dessous, nous trouvons le rapport de la consommation de ressources énergétiques, nous pouvons voir que la majorité de l'énergie (96%) est consommée par le processeur (la partie PS) et le reste est devisé entre les cellules logiques, les entrées/sorties, les signaux et les horloges.

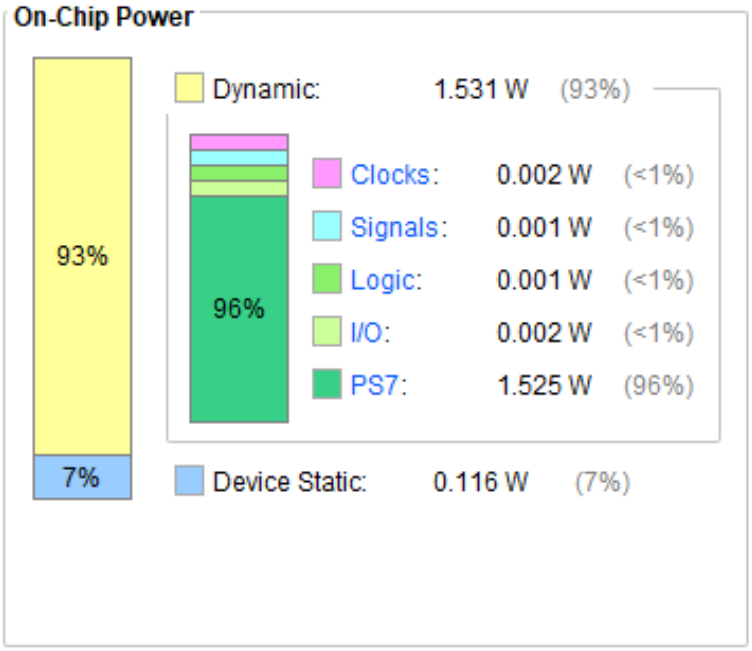


Figure III.50 : Rapport énergétique.

La prochaine étape consiste à générer un fichier binaire (bitstream) qui est importé avec les fichiers déjà créés dans l'outil SDK.

III.6 Le logiciel Xilinx SDK

SDK est un outil qui permet la création des applications en langage C/C++, qui s'exécutent sur le processeur physique ARM A9 du circuit FPGA. L'application permet de gérer la partie matérielle du projet à travers le processeur, qui permet de faire des applications réelles.

III.7 Conclusion

Dans ce chapitre, nous avons commencé par définir la structure matérielle et logicielle constituant notre système. Ensuite, nous avons présenté les étapes de création de notre projet en utilisant l'outil Vivado et le langage VHDL ainsi que les étapes de création des IP. Nous avons également expliqué le principe de fonctionnement des trois IP qui composent notre système avec les résultats de test de chaque bloc, les menaces considérées sont : la menace de déni de service, la menace de blocage de mise en veille, et la menace de brouillage. Les étapes de l'implémentation sur FPGA du bloc de détection de la menace de blocage de mise en veille ont été présentées. Enfin, nous avons défini l'architecture matérielle globale qui permet de connecter ce bloc au processeur.

Conclusion générale et perspectives

L'internet des objets est une technologie qui a connu beaucoup de succès ces dernières années, et qui est de plus en plus présente dans tous les domaines tels que : la santé, l'énergie nucléaire et la sécurité.

Les progrès des systèmes embarqués ont permis d'intégrer de plus en plus de puissance, de capacité d'acquisition et de calcul. Les FPGA qui offrent à la fois une puissance de calculs élevée, une grande flexibilité et une capacité de reprogrammation en fonction de l'application souhaitée, ont contribué aux avancées dans le domaine des objets connectés. Cependant, les objets connectés présentent un sérieux problème qui est la sécurité, car ils embarquent peu de puissance avec des ressources énergétiques réduites. Par conséquent, protéger les objets IOT contre les attaques, est devenu une nécessité pour assurer leur bon fonctionnement.

Dans le cadre de notre projet de fin d'études, nous avons développé un système qui permet de détecter trois types de menaces sur les objets connectés que nous avons implémenté sur une carte FPGA Zybo Z-7, le système conçu est composé d'une partie matérielle représentant les différents blocs d'IP, chaque IP se charge de la détection de l'une des menaces, et d'une partie logicielle gérée par le processeur ARM A9 du circuit FPGA.

Le premier bloc d'IP de note système détecte la menace déni de service (DOS), en analysant la trame IP (Internet Protocol) afin de détecter des anomalies dans les paquets transmis.

Le deuxième bloc d'IP détecte la menace de brouillage. Ce module effectue une FFT sur les signaux afin de détecter les anomalies.

Le troisième bloc d'IP permet de détecter la menace de blocage de mise en veille. Ce bloc vérifie le niveau de la batterie afin de détecter une consommation anormale de la batterie.

Afin d'assurer le bon fonctionnement des IP de détection des menaces conçus, nous avons effectué des bancs de test (test bench) à l'aide l'outil de simulation de VIVADO. Les résultats de simulations obtenus ont montré le bon fonctionnement de chaque bloc. Le bloc de détection de blocage de la mise en veille a ensuite été synthétisé et implémenté sur la carte FPGA Zybo Z-7. Une architecture matérielle complète pour ce bloc a également été définie comprenant l'IP de détection de la menace de blocage de mise en veille, le processeur ARM A9 et d'autres blocs IP développés par Xilinx, ensuite, le wrapper de cette architecture a été généré, puis, nous avons effectué la synthèse et l'implémentation ce qui a permet de générer la consommation de ressource matérielle et énergétique. L'architecture matérielle créée doit être pilotée par une partie software en créant une application en C sur le logiciel SDK qui s'exécute par le processeur ARM A9. Cette application va gérer l'IP de détection de blocage de mise en veille.

Plusieurs perspectives sont envisageables afin d'améliorer les performances de notre système. Ces améliorations peuvent être faites sur les modules de notre projet comme suit :

- Pour le module de détection de la menace de brouillage : nous avons utilisé la transformée de fourrier rapide pour la détection du brouillage, cette méthode permet de savoir la plage de fréquences utilisée pour le brouillage, donc, nous pouvons ensuite utiliser un filtre pour éliminer le brouillage.
- Pour le module de détection de la menace de blocage de mise en veille : la méthode utilisée n'est pas très fiable. En effet, l'état de la batterie ne peut pas être déterminé avec certitude s'il y a une menace ou pas, elle peut être défectueuse, donc, nous proposons de rajouter un pare-feu (firewall) pour la détection de menace de blocage de mise en veille.

Dans notre projet, nous avons sécurisé les objets eux-mêmes, nous pouvons sécuriser les données transmises entre les objets et avec internet à l'aide des algorithmes de cryptographie.

Annexes

Annexe N^0 01

Le code VHDL du bloc de détection de la menace de brouillage

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_STD.ALL;
use IEEE.std_logic_unsigned.all;

entity dett_DOS is
    Port (
        trame_ethernet : in STD_LOGIC_VECTOR (1517 downto 0);
        valide : in STD_LOGIC;
        attaque_det : out STD_LOGIC
    );
end dett_DOS;

architecture Behavioral of dett_DOS is
    signal entet_trame_IP : std_logic_vector (159 downto 0);
    signal type_ethernet : std_logic_vector (15 downto 0);
    signal ttl : std_logic_vector (7 downto 0);
    signal ttl_avant : std_logic_vector (7 downto 0);
    signal compteur : integer range 0 to 10000 := 0;
begin
    type_ethernet <= trame_ethernet(15 downto 0);

    process(trame_ethernet, valide, ttl, ttl_avant, compteur)
    begin
        if valide = '1' and type_ethernet = x"0800" then
```

```
            entet_trame_IP <= trame_ethernet(159 downto 0);
            ttl <= entet_trame_IP(87 downto 80);

            if compteur >= 1000 and ttl_avant = ttl then
                attaque_det <= '1';
            else
                attaque_det <= '0';
            end if;

            if compteur = 10000 then
                compteur <= 0;
            else
                compteur <= compteur + 1;
            end if;

            ttl_avant <= ttl;
        else
            compteur <= 0;
            ttl_avant <= x"00";
            attaque_det <= '0';
        end if;
    end process;
end Behavioral;
```

Annexe N^0 02

Le code VHDL du banc de test de l'IP de détection de brouillage

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;
use work.CosinusPackage.all;
```

```
use work.SinusPackage.all;
use work.ArrondissementPackage.all;

entity FFT_Brouillage_Detection_TB is
end entity FFT_Brouillage_Detection_TB;

architecture Behavioral of FFT_Brouillage_Detection_TB is
    -- Constantes pour le testbench
    constant N : positive := 8;
    constant Longeur_donnees : positive := 8;
    constant Val_seuil : integer := 100;

    -- Signaux du testbench
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal donnees_entree : std_logic_vector(Longeur_donnees-1 downto 0);
    signal brouillage_dett : std_logic;

begin
    -- Instanciation du composant sous test
    UUT : entity work.FFT_Brouillage_Detection
        generic map (
            N => N,
            Longeur_donnees => Longeur_donnees,
            Val_seuil => Val_seuil
        )
        port map (
            clk => clk,
            reset => reset,
            donnees_entree => donnees_entree,
            brouillage_dett => brouillage_dett
        );
```

```
-- Processus de génération de l'horloge
process
begin
    while true loop
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
        wait for 50 ns;
    end loop;
    wait;
end process;

-- Processus de test
process
begin
    -- Initialisation
    while true loop
    reset <= '1';
    donnees_entree <= (others => '0');
    wait for 60 ns;
    reset <= '0';
    wait for 60 ns;

    -- Envoi des données d'entrée
    donnees_entree <= "11111111";
    wait for 500 ns;
    donnees_entree <= "11110000";
    wait for 500 ns;
    donnees_entree <= "00000000";
    wait for 500 ns;
    donnees_entree <= "10101010";
```

```
        wait for 500 ns;
        donnees_entree <= "01010101";
        wait for 500 ns;
        donnees_entree <= "00001111";
        wait for 500 ns;
        end loop;
    end process;
end architecture Behavioral;
```

Annexe N^0 03

Le code en VHDL du bloc de détection de la menace de blocage de mise en veille

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dett_blockage is
    Port ( nv_batterie :in STD_LOGIC_VECTOR (6 downto 0);
           clk : in STD_LOGIC;
           blockage : out STD_LOGIC);
end dett_blockage;
architecture Behavioral of dett_blockage is
constant seuille_batterie: std_logic_vector(6 downto 0):="0110010";
signal blc: STD_LOGIC;
begin
process (clk, nv_batterie)
begin
if rising_edge(clk) then
    if nv_batterie<seuille_batterie  then
        blc<='1';
    else
        blc<='0';
    end if;
end if;
end process;
```

```
blockage<= blc;
end Behavioral;
```

Annexe N^0 04

Le code VHDL du banc de test de l'IP de détection du blocage de mise en veille

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dett_blockage_tb is
end dett_blockage_tb;

architecture Behavioral of dett_blockage_tb is
    component dett_blockage is
    Port (
        nv_batterie : in STD_LOGIC_VECTOR (6 downto 0);
        clk : in STD_LOGIC;
        blockage : out STD_LOGIC
    );
    end component;

    signal nv_batterie : STD_LOGIC_VECTOR (6 downto 0):="1100100";
    signal clk : STD_LOGIC := '0';
    signal blockage : STD_LOGIC := '0';
begin
    -- Instanciation de l'unité sous test
    uut : entity work.dett_blockage
        port map (
            nv_batterie => nv_batterie,
            clk => clk,
            blockage => blockage
        );
```

```
process
begin


   while true loop
   clk<='0';
   wait for 50 ns;
   clk <='1';
   wait for 50 ns;
   nv_batterie <= "1100100"; --100%
   clk<='0';
   wait for 50 ns;
   clk <='1';
   wait for 100 ns;
   nv_batterie <= "1010000"; --80%
   clk<='0';
   wait for 50 ns;
   clk <='1';
   wait for 100 ns;
   nv_batterie <= "1000110"; --70%
   clk<='0';
   wait for 50 ns;
   clk <='1';
   wait for 100 ns;
   nv_batterie <= "0110010"; --50%
   clk<='0';
   wait for 50 ns;
   clk <='1';
   wait for 100 ns;
   nv_batterie <= "0101000"; --40%
   clk<='0';
   wait for 50 ns;
```

```
        clk <='1';
        wait for 100 ns;
        nv_batterie <= "0011110"; --30%
        clk<='0';
        wait for 50 ns;
        clk <='1';
        wait for 100 ns;
        nv_batterie <= "0000000"; --0%
        clk<='0';
        wait for 50 ns;
        clk <='1';
        wait for 100 ns;

        end loop;
    end process;
end Behavioral;
```

Annexe N^0 05

Le code VHDL du banc de test de l'IP de détection du déni de service

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_STD.ALL;
use IEEE.std_logic_unsigned.all;

entity dett_DOS is
    Port (
        trame_ethernet : in STD_LOGIC_VECTOR (1517 downto 0);
        valide : in STD_LOGIC;
        attaque_det : out STD_LOGIC
    );
end dett_DOS;
```

```
architecture Behavioral of dett_DOS is
    signal entet_trame_IP : std_logic_vector (159 downto 0);
    signal type_ethernet : std_logic_vector (15 downto 0);
    signal ttl : std_logic_vector (7 downto 0);
    signal ttl_avant : std_logic_vector (7 downto 0);
    signal compteur : integer range 0 to 10000 := 0;
begin
    type_ethernet <= trame_ethernet(15 downto 0);

    process(trame_ethernet, valide, ttl, ttl_avant, compteur)
    begin
        if valide = '1' and type_ethernet = x"0800" then
            entet_trame_IP <= trame_ethernet(159 downto 0);
            ttl <= entet_trame_IP(87 downto 80);

            if compteur >= 1000 and ttl_avant = ttl then
                attaque_det <= '1';
            else
                attaque_det <= '0';
            end if;

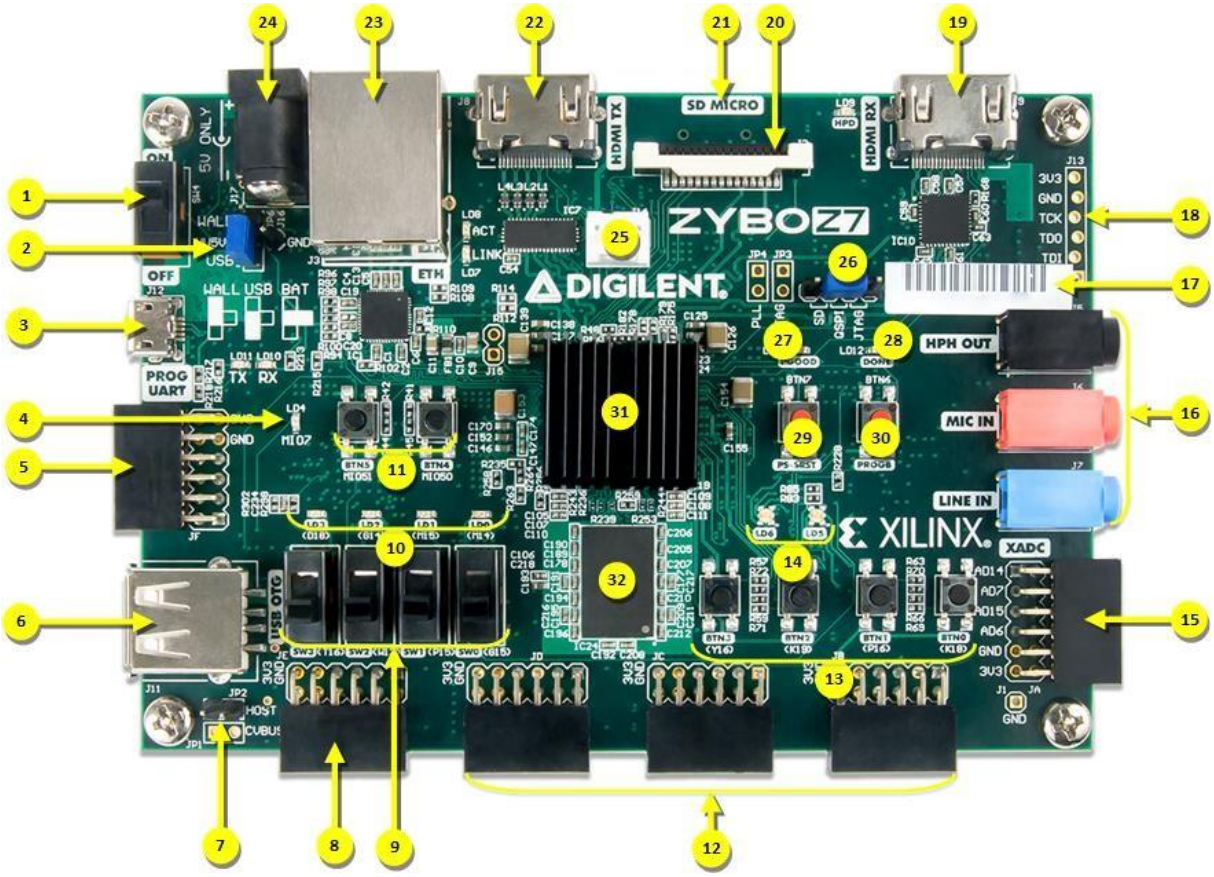
            if compteur = 10000 then
                compteur <= 0;
            else
                compteur <= compteur + 1;
            end if;

            ttl_avant <= ttl;
        else
            compteur <= 0;
            ttl_avant <= x"00";
            attaque_det <= '0';
```

```
        end if;
    end process;
end Behavioral;
```

Annexe N^0 06

Présentation de la carte Zybo Z-7



Ports	Description
1	Le switch d'alimentation
2	Power select jumper
3	Port USB JTAG/UART
4	Utilisateur LED MOI
5	Port Pmod MOI
6	Port USB 2.0 Host/OTG
7	USB host power enable jumper
8	Port Pmod standard
9	Switches
10	LED
11	Boutons MI
12	Port Pmod haute-vitesse
13	Boutons
14	LED RGB
15	Port Pmod XADC

16	**Port audio codec**
17	**Adresse MAC**
18	**Port JTAG externe**
19	**Port d'entrée HDMI**
20	**Port Pcam CSI-2**
21	**Connecteur microSD**
22	**Port de sortie HDMI**
23	**Port Ethernet**
24	**Connecteur d'alimentation externe**
25	**Connecteur**
26	**Jumper mode programmation**
27	**Connecteur d'alimentation LED**
28	**FPGA programming done LED**
29	**Bouton reset du processeur**
30	**Bouton de remise à 0 du FPGA**
31	**Zynq-7000**
32	**Mémoire DDR3L**

Bibliographie

[1] Mokhtari.H., (2021), Conception d'un circuit FPGA/ASIC pour l'optimisation du point de puissance maximale (MPPT), Ecole Nationale Supérieure de Technologie Alger, Algérie.

[2] Belhachat.F., (2021), Electronique numérique avancée, Ecole Nationale Supérieure de Technologie Alger, Algérie

[3] http://meteosat.pessac.free.fr/Cd_elect/CFAO/cpld/pld/html_pld/pld.htm, date de consultation : 26 mai 2023.

[4] Belhachat.F., (2022), Architecture des systèmes embarqués II, Ecole Nationale Supérieure de Technologie Alger, Algérie.
.
[5] https://www.researchgate.net/figure/Description-de-larchitecture-generique-dun-FPGA_fig5_288327838, date de consultation : 21 avril 2023.

[6] Abbadi.B., (2021), Architecture des Systèmes Embarqués, Ecole Nationale Supérieure de Technologie Alger, Algérie.

[7] Zellat.K., (2022), Exploitation des Plateformes des Systèmes Embarqués, Ecole Nationale Supérieure de Technologie Alger, Algérie.

[8] Benabdallah, A.Y, Boudour.R., (2010) Méthodologie de Conception à base d'IP, Université Badji-Mokhtar Annaba, Algérie.

[9] https://www.researchgate.net/figure/Flot-classique-de-conception-FPGA-simplifie-Effectivement-dans-un-contexte_fig4_278816392, date de consultation : 26 mai 2023.

[10] Prost-Boucle.A., Muller.O., Rousseau.F., (2014), Outils et langages gratuits et/ou open-sources pour les FPGA, laboratoires de recherche TIMA-CNRS/Grenoble-INP/UJF, France.

[11] https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual

[12] https://image5.slideserve.com/9669755/zynq-architecture-l.jpg. Date de consultation : 21 avril 2023.

[13] Kareem Jumaa, N. (2017). Survey: Internet of Things using FPGA. *Iraqi Journal for Electrical And Electronic Engineering*, *13*(1), 38–45. https://doi.org/10.33762/eeej.2017.128785.

[14] Boyer.Y., (2020), Étude et conception de méthodes de protection face aux attaques par corruption de mémoire pour systèmes embarqués dans le contexte de l'Internet des Objets. Informatique et langage [cs.CL], Université Montpellier, France.

[15] https://imgur.com/ZZJJwsw. Date de consultation : 6 mai 2023.

[16] https://commons.wikimedia.org/wiki/File:Diversit%C3%A9_IOT.jpg?uselang=fr. Date de consultation : 27 mars 2023.

[17] Ouali-Alami.M., (2016), La conception d'une prise connectée basée sur la technologie d'IoT (Internet of Things), Université Sidi Mohamed Ben Abdellah, Maroc.

[18] Abbad.L., (2021), Réseaux informatique, Ecole Nationale Supérieure de Technologie Alger,Algérie.

[19] https://user.oc-static.com/upload/2021/06/01/16225670098336_P2C5-1.png.Date de consultation : 21 mai 2023.

[20] https://www.frameip.com/tcpip/, date de consultation : 22 mai 2023.

[21] Elnawawy, M., Farhan, A., Nabulsi, A. A., Al-Ali, A. R., & Sagahyroon, A. (2019, December). Role of FPGA in internet of things applications. *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. http://dx.doi.org/10.1109/isspit47144.2019.9001747.

[22] Macherez. A., (2019), Détection des menaces IOT sur FPGA. Université de Lille, France.

[23] Fernades.D et Sarr.P.A., (2010), La protection des réseaux contre les attaques DOS, Université Paris Descartes, France.

[24] Vignau.B., (2020), La sécurité dans l'internet des objets : des configurations par défaut aux dénis de services, Université du Québec à Chicoutimi, canada.

[25] Gomes, T., Pinto, S., Gomes, T., Tavares, A., & Cabral, J. (2015, September). Towards an FPGA-based edge device for the Internet of Things. *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. http://dx.doi.org/10.1109/etfa.2015.7301601.

[26] Zhou, B., Egele, M., & Joshi, A. (2017, September). High-performance low-energy implementation of cryptographic algorithms on a programmable SoC for IoT devices. *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. http://dx.doi.org/10.1109/hpec.2017.8091062

[27] www.xilinx.com, date de consultation : 29 mai 2023.

ملخص

الهدف الرئيسي لمشروع التخرج هذا هو تصميم نظام يسمح بالكشف عن بعض التهديدات على الأجهزة المتصلة بالأنترنت، التهديدات المعتبرة هي: تهديد خدمة الرفض، وتهديد التشويش، وتهديد حظر وضع السكون التي تم إنشاؤها على شكل مكونات افتراضية.

للتأكد من صحة تصميم المكونات الافتراضية، تم إجراء اختبارات أداء. نتائجها بينت العمل الجيد لكل مكون افتراضي. المكون المستخدم للكشف عن حظر وضع السكون تم توليفها وتنفيذها على البطاقة FPGA Zybo Z-7.

تم أيضا تحديد بنية هندسية تضم هذا المكون الافتراضي، المعالج أرم 9أ ومكونات افتراضية مطورة من طرف XILINX

الكلمات المفتاحية: أنترنت الأشياء، رفض الخدمة، التشويش، منع وضع السكون، المكونات الافتراضية

Résumé

L'objectif principal de ce projet de fin d'études est de concevoir un système qui permet de détecter certaines menaces sur les objets connectés, les menaces considérées sont : la menace de déni de service, la menace de brouillage et la menace de blocage de la mise en veille créés sous forme de composants virtuels (IP).

Afin d'assurer le bon fonctionnement des IP conçus, des tests bench ont été effectués en utilisant l'outil Vivado de Xilinx. Les résultats de simulations obtenus ont montré le bon fonctionnement de chaque bloc. Le bloc d'IP de détection de blocage de la mise en veille a été synthétisé et implémenté sur la carte FPGA Zybo Z-7. Une architecture matérielle complète pour ce bloc a été aussi définie incluant cet IP, un système à processeur ARM A9 ainsi que d'autres IP développés par Xilinx.

Mots clés : Internet des Objets, déni de service, blocage de mise en veille, brouillage, FPGA, IP, processeur ARM A9.

Abstract

The main objective of this final-year project is to design a system that can detect certain threats on connected objects, the threats considered are: the denial-of-service threat, the jamming threat and the standby blocking threat created in the form of virtual components (IPs).

In order to ensure the proper functioning of the designed IPs, tests bench was carried out using the Vivado tool from Xilinx. The simulation results obtained showed that each block functioned correctly. The standby detection IP was synthesized and implemented on the Zybo Z-7 FPGA board. A complete hardware architecture for this block was also defined, including this IP, an ARM A9 processor system and other IP developed by Xilinx.

Keywords: Internet of Things, denial of service, sleep mode blocking, jamming, FPGA, IP, ARM A9 processor.