République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Sousse

Institut Supérieur des Sciences
Appliquées et de Technologie
de Sousse

DEPARTEMENT GENIE ELECTRONIQUE

Rapport de projet de fin d'études

Présenté en vue de l'obtention du :

Diplôme de licence unifiée en

Electronique électrotechnique et automatique

Spécialité : système embarqué

Conception et Réalisation D'une Smart City

Réalisé par :

Mhadhebi Zied

Manaa Nidhal

Encadré par :

Mm.Imen Werda Souissi

Institut Supérieur Des Sciences Appliquées Et Technologie De Sousse

Année Universitaire 2022-2023

Zied & Nidhal

S

Introduction générale

Les villes intelligentes sont en constante évolution et sont de plus en plus axées sur l'intelligence artificielle (IA) pour améliorer le quotidien des citoyens. L'IA peut aider les villes intelligentes à analyser les données en temps réel, à anticiper les besoins des citoyens. Par exemple, les systèmes de gestion des transports dans les villes intelligentes peuvent utiliser l'IA pour prédire les niveaux de congestion du trafic routier afin d'ajuster convenablement les itinéraires des utilisateurs en temps réel.

Les villes peuvent également utiliser l'IA pour surveiller les conditions météorologiques, les niveaux de pollution et la production d'énergie en conséquence. L'utilisation de l'IA peut également aider les villes intelligentes à améliorer les services publics en permettant aux citoyens de communiquer avec les autorités locales de manière plus efficace. L'IA peut aider les villes intelligentes à améliorer la sécurité publique en permettant aux autorités de détecter les comportements suspects et les menaces potentielles avant qu'elles ne deviennent dangereuses. En somme, l'utilisation de l'IA dans les villes intelligentes offre un potentiel considérable pour améliorer l'efficacité, la sécurité et la qualité de vie des citoyens. Cependant, il est essentiel que l'IA soit utilisée de manière responsable, en garantissant la sécurité et la confidentialité des données et en assurant la transparence des systèmes d'IA utilisés.

Les villes intelligentes sont confrontées à des défis de plus en plus complexes en matière de circulation, tels que la congestion, les temps d'attente excessifs aux feux de circulation et les accidents de la route. Pour y faire face, l'IA sera utilisée pour contribuer à réduire les temps d'attente, à améliorer la fluidité du trafic et à réduire les émissions de gaz d'échappement. De plus, l'IA peut prédire les conditions de circulation, permettant ainsi aux conducteurs de

planifier leurs itinéraires en fonction des conditions en temps réel. C’est dans ce cadre que s’inscrit ce projet de fin d’étude qui vise à utiliser l’intelligence artificielle pour optimiser la gestion des feux de circulation, en fonction de l’état de l’encombrement du trafic collecté en temps réel.

Ce manuscrit est organisé comme suit : dans le premier chapitre, nous allons présenter les différents domaines d’applications d’IA dans les villes intelligentes en concentrant sur les systèmes de gestion de routier intelligents. Dans le deuxième nous allons détaillées les étapes de la conception d'un feu de circulation intelligent sur la carte **FPGA Pynq-Z2.**

Chapitre 1: Etat de l’art

I. Introduction:

L'urbanisation rapide et l'augmentation de la population dans les villes ont mis une pression accrue sur les infrastructures urbaines, les ressources et les services. Dans ce contexte, la composition de ville intelligente, est apparue comme une réponse innovante pour relever les défis urbains en utilisant les technologies de l'information et de la communication (TIC). Une ville intelligente utilise des capteurs, des appareils connectés, l'analyse de données et l'IA pour améliorer la qualité de vie des citoyens, optimiser les services urbains et réduire les coûts.

II. L'intelligence artificielle dans les villes intelligentes :

Les villes intelligentes ont multiplié les domaines d'utilisation de l'IA pour améliorer la qualité de vie des citoyens, optimiser la gestion des ressources urbaines et réduire l'impact environnemental des villes, comme illustré dans la figure 1.

Figure1 : Les éléments conceptionnelle d'une ville intelligente

L'IA est devenue en tout point indissociable de la ville intelligente dans de nombreux domaines telle que :

1. Les maisons intelligentes:

Une maison intelligente est généralement équipée de technologies connectées et automatisées, qui utilisent l'IA pour rendre la maison plus efficace, plus sûre et plus confortable. Deux modèles sont particulièrement intéressants :

1.1. Modèle d'une maison intelligente de Samsung :

Avec SmartThings Station, les utilisateurs peuvent automatiser différents aspects de leur environnement domestique. Les utilisateurs peuvent automatiser les tâches ménagères telles que l'ouverture et la fermeture des stores et des appareils électroménagères en utilisant l'application SmartThings de Samsung [2].

Figure 2 : Contrôle des appareils avec l'application SmartThings

1.2. Maison intelligente Alexa:

Alexa, illustré par la figure 3, peut transformer un foyer en une maison intelligente. Il vous suffit d'utiliser une commande vocale ou une routine personnalisée pour contrôler des appareils intelligents compatibles comme les ampoules, prises, caméras, serrures [3].

Figure3 : Communication vocale avec la maison intelligente

2. Gestion intelligente de l’énergie:

L'énergie intelligente devient de plus en plus vitale de nos jours. Un système énergétique intelligent est composé de nouvelles technologies et infrastructures qui créent de nouvelles formes de flexibilité [4]. Comme exemple, on peut admirer la fleur intelligente, présentée par la figure 4. Ses pétales solaires sont équipés de panneaux solaires rotatifs et se déplacent automatiquement pour suivre le soleil, optimisant ainsi la production d'énergie solaire. [5]

Figure 4 : La fleur intelligente

3. Parking intelligent :

Le smart parking (stationnement intelligent) a été développé pour diverses raisons. Depuis plusieurs années, les responsables de plusieurs villes ont remarqué que leurs conducteurs avaient de réels problèmes pour trouver une place de parking facilement.[6]

On intègre différentes technologies comme des capteurs, des caméras, des analyses de données et des systèmes de communication pour créer un parking intelligent. L'idée, c'est de donner des informations en temps réel, améliorer la gestion du stationnement, rendre l'expérience des utilisateurs meilleure et réduire les problèmes de congestion et l'impact sur l'environnement.

3.1. Gestion de parking avec un système RAPI :

Les systèmes de RAPI utilisent en général des séries de techniques et méthodes de traitement d'image afin de détecter, repérer et extraire les plaques d'immatriculation de l'image [7] comme illustré dans la figure 5.

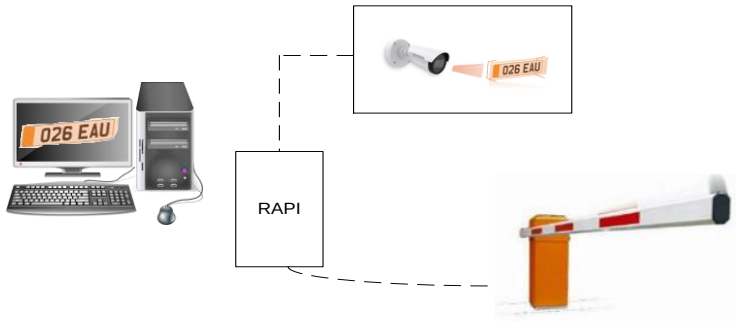


Figure 5 : Sytème RAPI

Parmi ces techniques, nous citons :

- Technique basée sur les propriétés de la plaque
- Technique basée sur les contours de la plaque
- Technique basée sur l'intelligence artificielle
- Technique basée sur la signature de la plaque

4. Gestion intelligente du traffic routier :

Plusieurs techniques ont été adoptées pour moderniser le traffic routier. Les approches plus innovantes sont présentées dans ce qui suit :

1.1. SCOOT (Split Cycle Offset Optimisation Technique):

Dans le système de contrôle développé par le TRL (Trafic Research Laboratory), l'ensemble des informations recueillies sur le terrain sera collecté par un centre de gestion, qui s'occupe de traiter les informations et renvoyer des indications directement aux intersections. Les véhicules sont détectés par des dispositifs pouvant être placés à divers endroits sur les voies : au niveau des feux ou à une certaine distance afin de pouvoir mesurer le débit du trafic [10], ce fonctionnement est illustré dans la figure 7 :

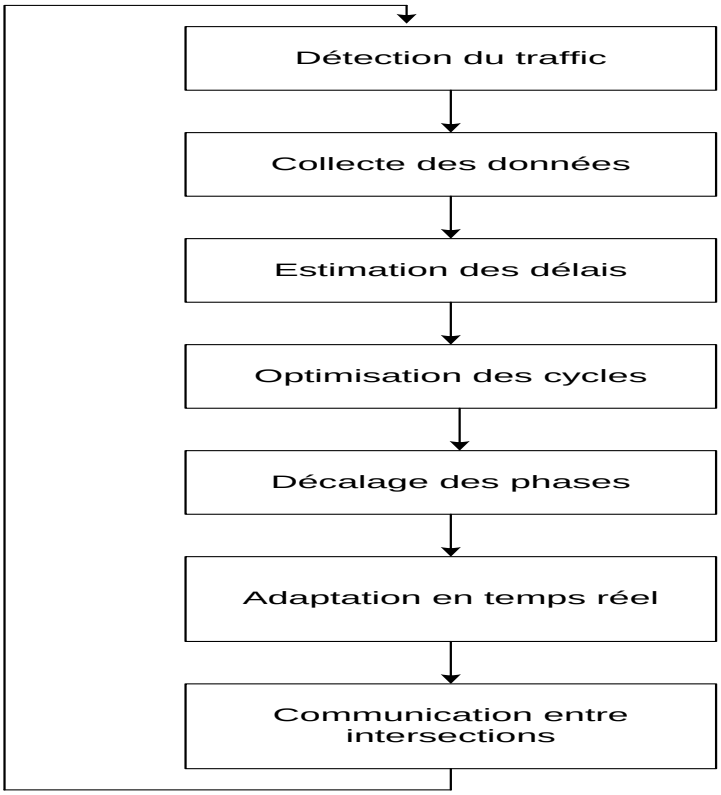


Figure 7: Fonctionnement de SCOOT

1.2. SCATS (The Sydney coordinated adaptive traffic system):

Ce système utilise une notion de hiérarchie pour ajuster le temps des cycles en fonction des données recueillies afin de diminuer le délai et les arrêts. Son fonctionnement est détaillé par la figure 8 :

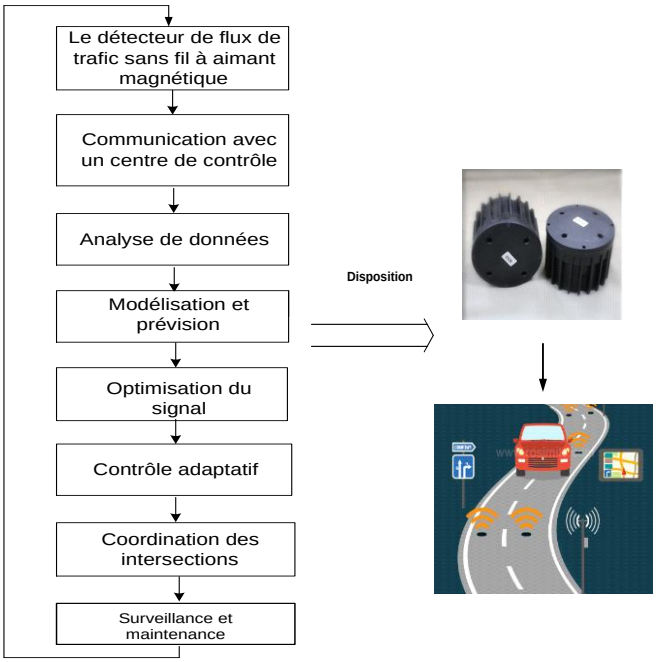


Figure 8 : Fonctionnement de SCATS

III. Problématique et objectifs:

Les systèmes de transport intelligents (STI) sont nés de la reconnaissance par les pouvoirs publics des limites des systèmes de transports, avec l'accroissement des phénomènes de congestion et d'accidents de la route. Ainsi, les premiers feux tricolores ont vu le jour aux États-Unis en 1914 et se sont généralisés par le processus de digitalisation observé dans les différents domaines [9]. L'évolution des STI est détaillée par la figure 7.

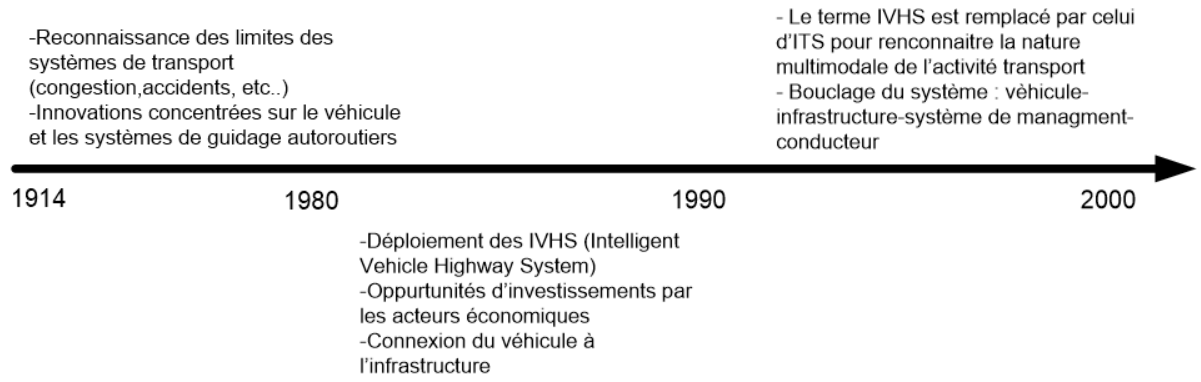


Figure 7 : Évolution des STI depuis 1914

Les systèmes de transports ont prouvé leurs efficacités pour améliorer la fluidité du trafic et à réduire les temps de déplacement. Cependant ils ont certaines limites :

- Coût élevé : Selon des études antérieures [12], l'installation de l'une de ces solutions « SCATS ou SCOOT » coûterait au total en moyenne 55.000$ par intersection, cela peut limiter sa mise en œuvre dans des villes plus petites ou des zones avec des budgets limités.

- Temps d'attente : Lorsque les temps de signalisation sont mal synchronisés ou trop longs, cela peut entraîner des temps d'attente excessifs pour les conducteurs, ce qui peut être frustrant et conduire à des embouteillages.[13]

- Impact environnemental : Le système SCOOT par exemple est contribué à une augmentation de la pollution atmosphérique, car il peut encourager une conduite plus agressive et augmenter le nombre de véhicules sur la route.[14]

Pour surpasser les limitations précédemment citées, on propose au niveau de notre projet de fin d'étude, de développer un système de feux de circulation intelligent. Notre objectif est de

concevoir une solution innovante qui utilise des technologies avancées telles que l'intelligence artificielle pour optimiser la gestion du trafic routier et réduire les embouteillages. Notre solution permettra d'améliorer la fluidité du trafic et d'optimiser les temps de parcours pour les usagers de la route. Notre projet permettra également d’accroître la sécurité routière en minimisant les risques d'accidents liés aux interférences entre les véhicules en favorisant une meilleure coordination des flux de circulation. Nous sommes déterminés à contribuer à l'amélioration de l'efficacité des systèmes de feux de circulation, en créant une solution intelligente et adaptative qui répond aux besoins des zones urbaines d'aujourd'hui.

IV. Conclusion:

En conclusion de ce premier chapitre sur les feux de circulation actuels, nous avons examiné les différents systèmes de gestion des transports utilisés et les problèmes auxquels ils sont confrontés. Ce premier chapitre a jeté les bases en identifiant les problèmes et les limitations des feux de circulation actuels. On peut ainsi concevoir notre solution de feux de circulation intelligents.

Chapitre 2 : Implémentation du système

I. Introduction:

Dans ce deuxième chapitre, nous détaillerons les étapes de la conception d'un feu de circulation intelligent sur la carte **FPGA Pynq-Z2**. La conception de l'architecture structurelle sera détaillée en présentant la machine d'état appliquée et en visualisant les phases de vérification des différents composants utilisés lors de la conception du système de feux de circulation.

II. Plateforme d'implémentation : FPGA Pynq-Z2

Les FPGA :

L'architecture de base d'un FPGA, illustrée par la figure 9 , est constituée principalement de cellules SRAM (Static Random Access Memory) [15]. Pour interfacer l'FPGA avec son environnement externe, des unités d'entrée-sortie IOB (Input Output Blocks) sont utilisées (périphériques, etc.…) [17].

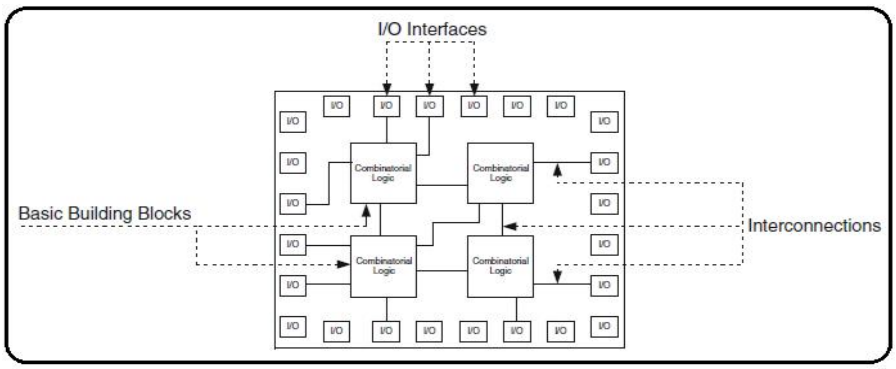


Figure 8 : Concept architectural de base d'un FPGA [16]

Les composants des CLB (Configurable Logic Block) permettent l'implémentation des fonctions de mémoire et synchronisent le code sur le FPGA [18].

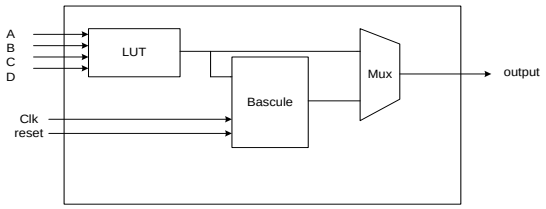


Figure 9 : Structure d'un CLB de base

La figure 11 détaille la carte Xilinx Pynq-Z2. Cette dernière a été sélectionnée pour la conception de notre système de feux de circulation intelligent.

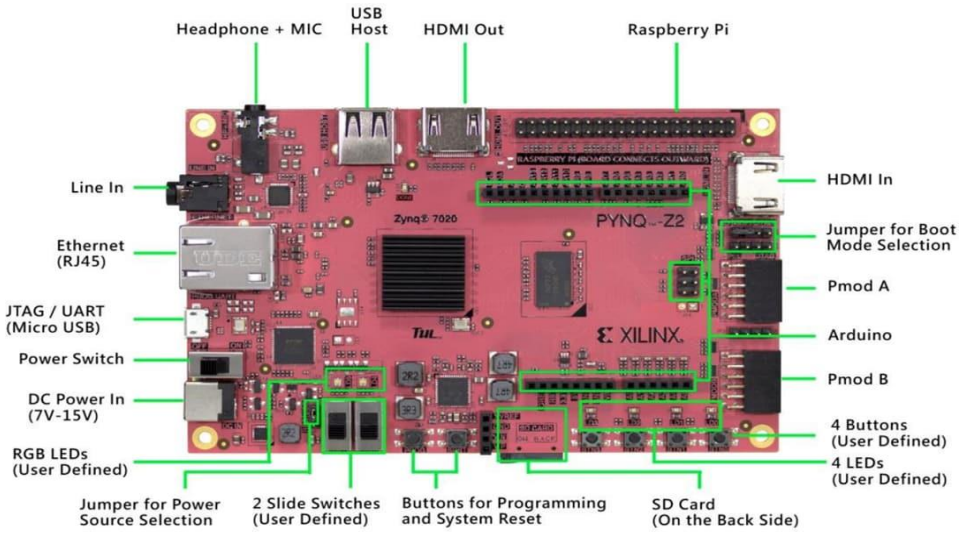


Figure 10 : La carte PYNQ-Z2[19]

Le SoC Zynq-7000 est caractérisé par sa combinaison unique d'un processeur ARM Cortex-A9 à deux cœurs et d'une matrice logique programmable FPGA. Cette intégration permet d'exécuter à la fois des traitements logiciels et matériels dans un même système, offrant ainsi une grande flexibilité et une puissance de traitement élevée. De plus, le Zynq-7000 dispose d'une connectivité étendue avec des interfaces HDMI, USB, Ethernet et des broches d'entrée/sortie comme illustré dans « le tableau de caractéristiques de Soc-Zynq-7000 » dans l'annexes [15].

Cette plateforme de développement présente plusieurs avantages :

- Flexibilité et puissance de traitement : La Pynq-Z2 offre la possibilité de réaliser à la fois des traitements logiciels et matériels dans un même système.
- Prototypage rapide : La Pynq-Z2 facilite le prototypage de systèmes embarqués grâce à son environnement de développement convivial offrant la capacité à combiner les traitements logiciels et matériels.
- Communauté active : La carte Pynq-Z2 bénéficie d'une communauté d'utilisateurs active, offrant des ressources, des exemples de projets et un support pour faciliter le développement et l'apprentissage.

Conception de l’architecture générale

La carte Pynq-Z2 est doté d’un oscillateur de 50 MHz comme horloge principale. La carte dispose également d'une source d'horloge supplémentaire fonctionnant à une fréquence de 125 MHz. Cette horloge est connectée à la broche H16 de la FPGA et provient de l'Ethernet Phy comme indiqué par la figure 12.

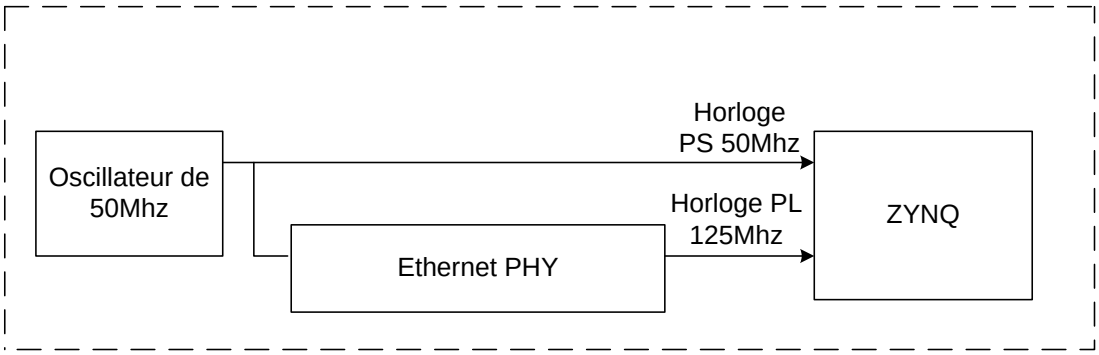


Figure 11 : Horloge source de Pynq-Z2

Les cycles de passage entre les différentes couleurs d'un feu de circulation, nécessitent des durées spécifiques pour assurer un flux de circulation régulier et sécurisé.

En utilisant un diviseur de fréquence, comme premier composant (C1), nous pouvons adapter la fréquence d'horloge de la carte Pynq-Z2 à des valeurs plus appropriées pour le fonctionnement du système de feux de circulation. Cela permet de simplifier la gestion du temps, de synchroniser les différentes phases et de contrôler précisément la durée de chaque phase du feu. La sortie du diviseur de fréquence sera le signal d'entrée du compteur (C2). Ce composant délivre à sa sortie le nombre d'impulsions d'horloge N qui sera l'entrée de la machine d'état (C3) et assurera la commande de la durée de chaque état du feu.

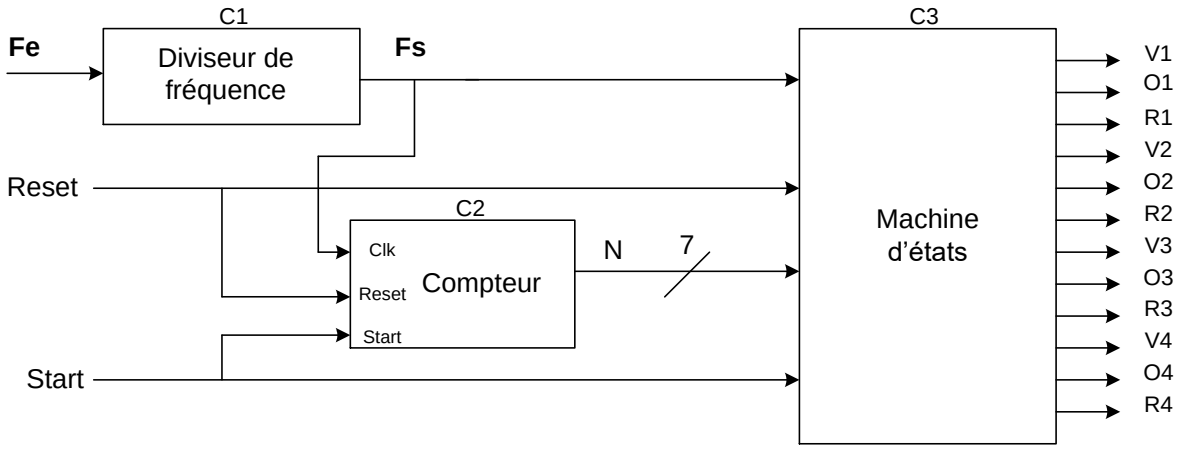


Figure 12 : Connexion des blocs

Diviseur de fréquence :

Présentation d'un diviseur de fréquence adapté aux besoins du système de feux de circulation :

La fréquence d'entrée de ce composant est de 125 MHz. On va générer une fréquence de sortie plus basse de Fe(p) en appliquant la formule suivante :

FS(p)= FE(p)/ (Nd + 1)

FS(p) et FE(p) étant respectivement les fréquences de sortie et d'entrée. Nd est la valeur de division à appliquer.

Pour obtenir une fréquence cible FS(p)=0,01Hz on va attribuer une valeur de division de Nd=12499999. Ainsi, pour la fréquence de sortie Fs0.01 Hz.

Le fonctionnement de ce diviseur est décrit par la figure 13 :

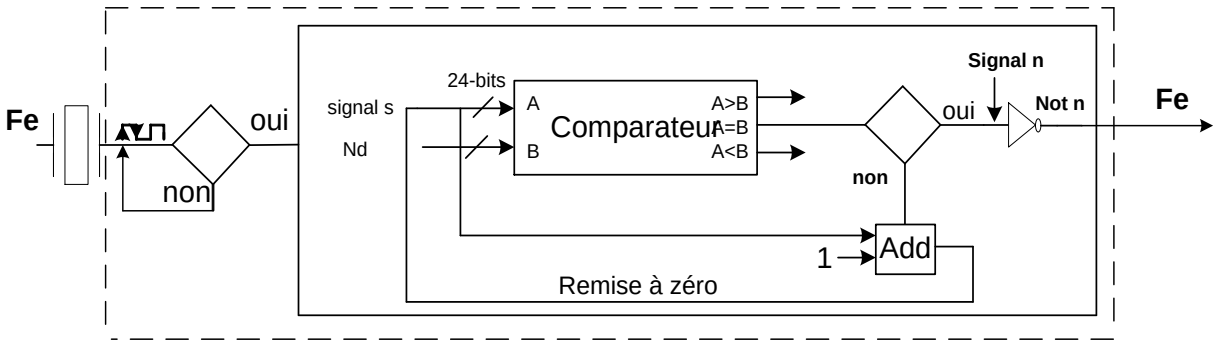


Figure 13 : Description interne du diviseur de fréquence

Un premier test T1 sera appliqué sur la fréquence d'entrée pour détecter un évènement sur l'entrée Fe. Une c

Après le lancement de l'analyse syntaxique et la synthèse pour générer un fichier Netlist représentant le circuit logique comme indiqué dans l'organigramme de la figure 14.

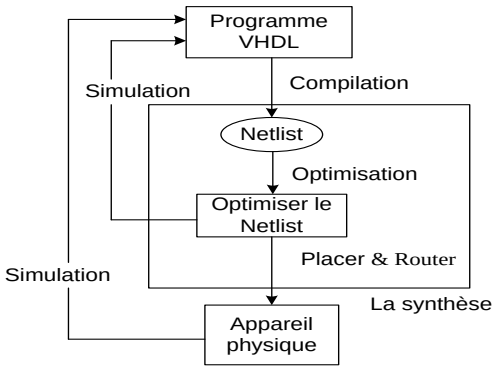


Figure 14 : Flux de conception VHDL

Nous obtenons le bloc Design de diviseur comme illustré dans la figure 15. Ensuite nous effectuons l'implémentation en réalisant le placement et le routage des composants logique sur la carte FPGA. Les pins utilisés sont spécifiés dans le tableau 2 dans l'annexes.

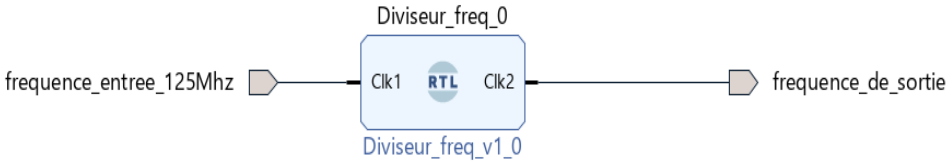


Figure 15 : Bloc design de diviseur

III. Compteur:

Nous initialisons le signal n2 avec la valeur "0000000", ensuite une opération de comparaison se déroule entre la valeur binaire maximale de compteur équivalente à 104 (1101000 en binaire), cette comparaison est effectuée pour vérifier si le compteur a atteint sa valeur maximale.

L'incrémentation du compteur "n2 <= n2 + 1" est une opération d'addition entre le signal n2 et la valeur 1. Cette opération est réalisée lorsque le signal de démarrage (start) est actif.

L'assignation de la valeur du compteur au signal de sortie N "N <= n2" est une opération d'assignation. La description du comportement de compteur est illustrée dans la figure 17.

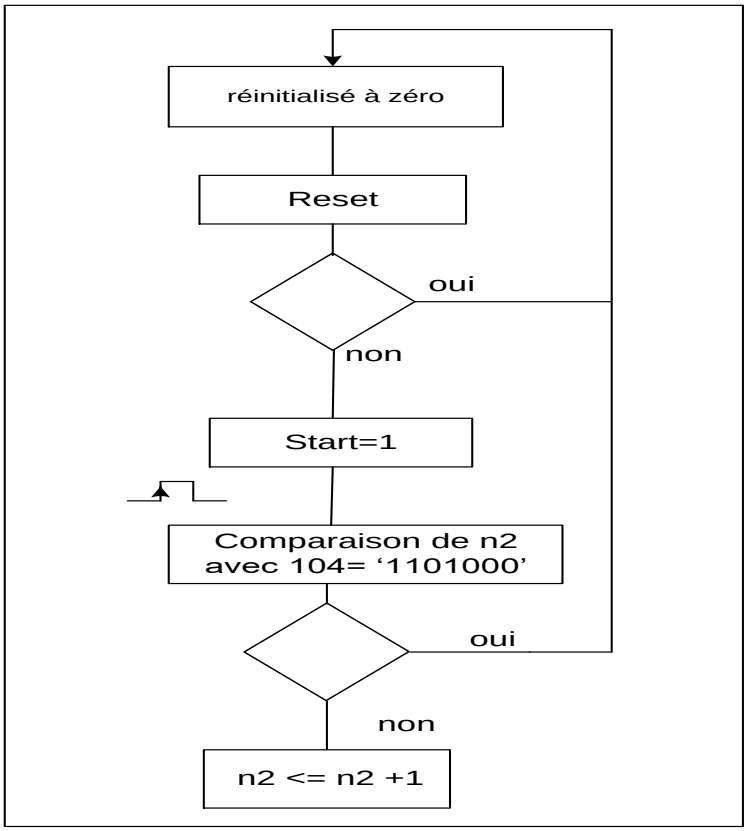


Figure 18 : Fonctionnement de compteur

Nous effectuons une analyse et une synthèse du design pour générer la représentation logique de compteur comme indiqué dans la figure ci-dessous.

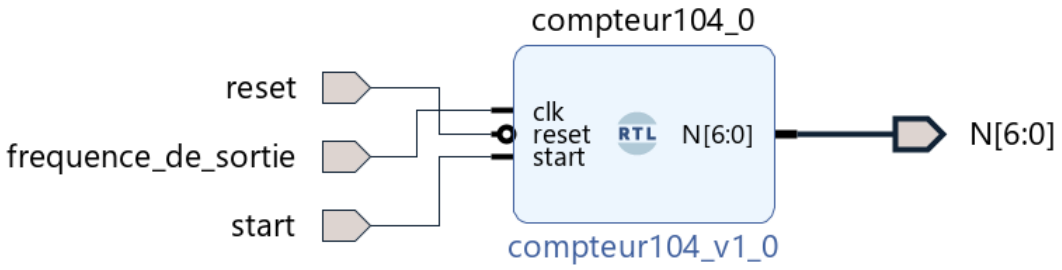


Figure 19 : Bloc du design de compteur

V. Machine d’états:

1. Description du fonctionnement du système de feux de circulation à l'aide d'une machine d'états :

Le système de feux de circulation est contrôlé à l'aide d'une machine d'états implémentée en VHDL. La machine d'états est basée sur une variable d'état (state) qui représente l'état actuel du système. Le fonctionnement du système est divisé en plusieurs états (s0, s1, s2, ..., s24) qui définissent les différentes phases du cycle des feux de circulation.

Lorsque le système est réinitialisé (RESET='0'), permet de mettre tous les feux en orangé clignotant. Ensuite, à chaque front montant de l'horloge (fréquence de sortie comme illustré dans le diviseur de fréquence), la machine d'états examine l'état actuel et effectue une transition vers l'état suivant en fonction de certaines conditions.

Le fonctionnement du système est déterminé par les transitions entre les états. Par exemple, lorsqu'un signal de démarrage (START='1') est détecté, le système passe de l'état s0 à l'état s1, et ainsi de suite, c’est le fonctionnement ordinaire. Chaque transition d'état est basée sur la valeur de l'entrée N, qui représente la valeur du compteur comme indiqué dans la figure 21 qui décrit le fonctionnement avec une machine d’états.

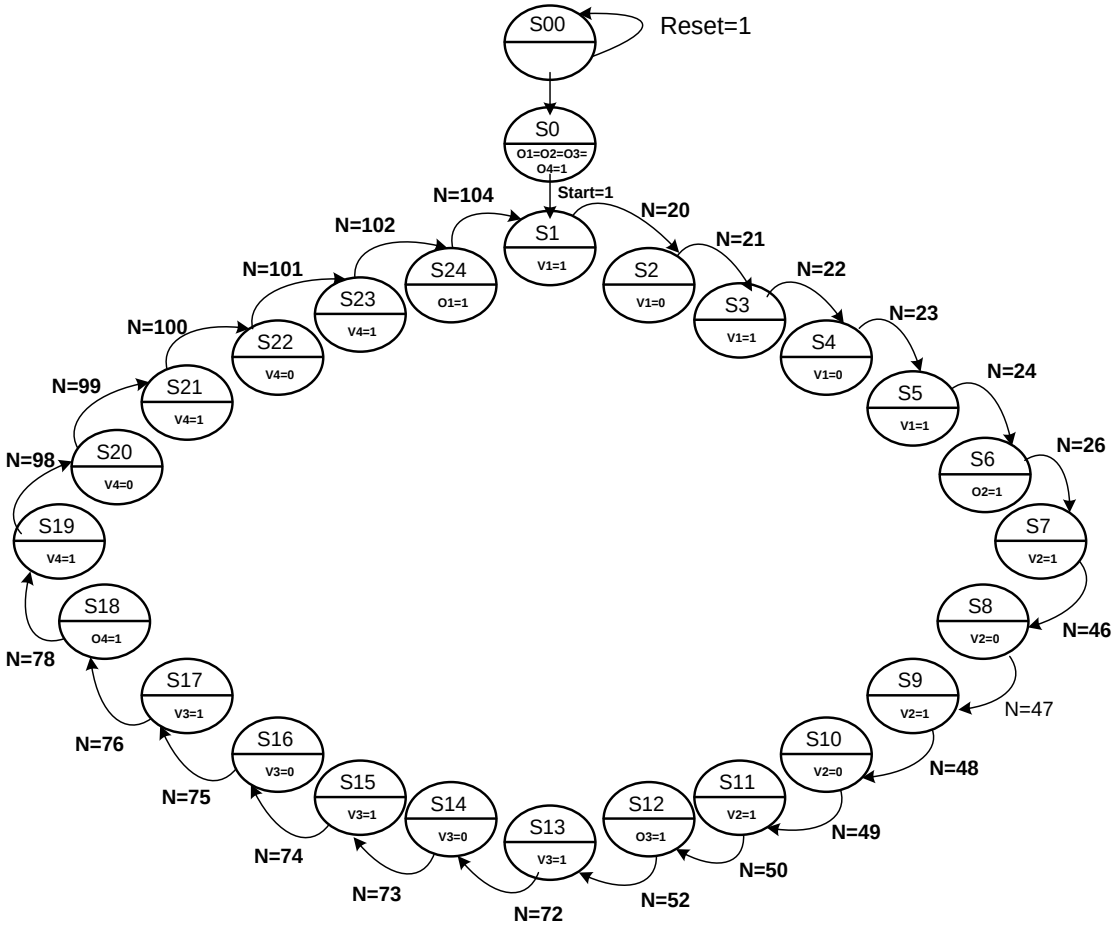


Figure20 : Machine d'états de feux de circulation

Les durées des feux sont : 10 secondes pour le feu vert suivies de 2 secondes de clignotement. La durée du feu orangé est de 1 seconde.

Pour les transitions on passe du vert vers le vert clignotant suivi par le rouge. Et du rouge vers l'orangé suivi par le vert.

La durée d'un clignotement est de 1 seconde (le feu s'allume pendant 0,5 seconde et s'éteint pendant 0,5 seconde).

2. Intégration de machine d'états en VHDL :

Nous effectuons une analyse et une synthèse du design pour générer la représentation logique de compteur comme indiqué dans la figure ci-dessous.

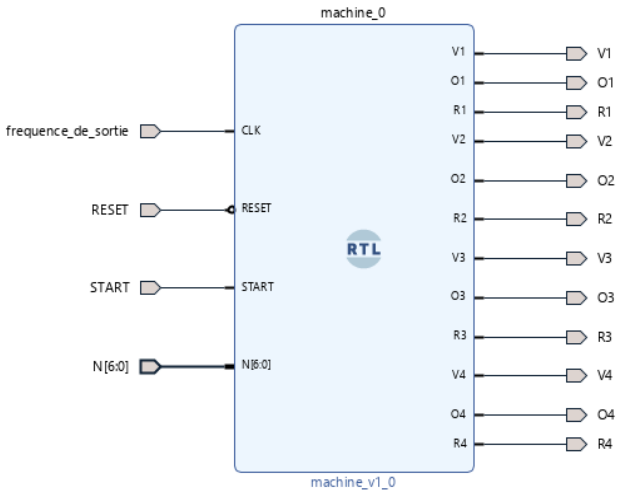


Figure 21 : Bloc design de la machine d'états

3. Connexion et intégration des blocs :

Les blocs de la machine à feux de circulation sont interconnectés pour assurer un fonctionnement synchronisé. Les signaux d'horloge et de contrôle sont utilisés pour relier les différents blocs entre eux.

Ces connexions permettent de transmettre les signaux nécessaires pour compter le temps écoulé dans chaque phase du feu, contrôler les feux de circulation et gérer les transitions entre les différentes phases. Grâce à ces connexions, les blocs fonctionnent de manière coordonnée pour assurer un système de feux de circulation efficace et sûr comme illustré dans la figure 22.

VI. Validation et tests du système de feux de circulation :

Avant de commencer les tests, il est important de vérifier que la conception du système de feux de circulation est correcte. Cela inclut la revue du code, l'inspection des blocs fonctionnels tels que le diviseur de fréquence, le compteur et la machine d'états, ainsi que la vérification des connexions entre ces blocs.

1. Test de diviseur de fréquence :

Le diviseur de fréquence que nous avons donné fonctionne en divisant la fréquence d'entrée par un facteur de 12 500 000 (12499999 en binaire). Cela signifie que pour chaque cycle d'horloge de la fréquence d'entrée, il génère 12 499 999 cycles d'horloge de la fréquence de sortie.

Si la fréquence d'entrée est de 10 ns (100 MHz) comme indiqué dans la figure ci-dessous

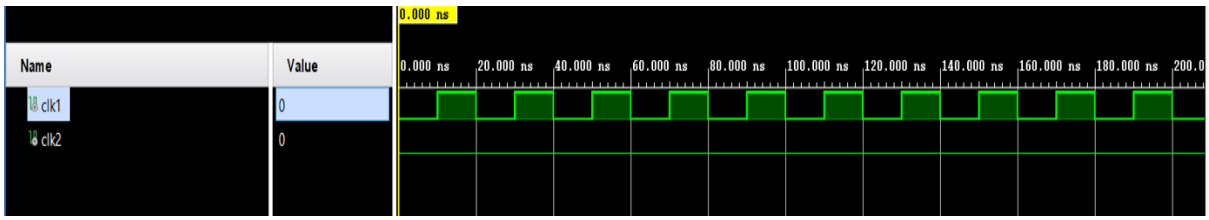


Figure 23 : Chronogramme d'horloge d'entrée

Nous pouvons calculer la fréquence de sortie comme suit :

Fréquence de sortie = Fréquence d'entrée / 12 500 000

= 100 MHz / 12 500 000

= 0,008 Hz (ou 8 mhz)

Donc, avec une fréquence d'entrée de 10 ns, la fréquence de sortie du diviseur serait d'environ 8 mhz. Cela signifie qu'il génère un cycle de sortie toutes les 125 ms environ, comme illustré dans la figure 24.

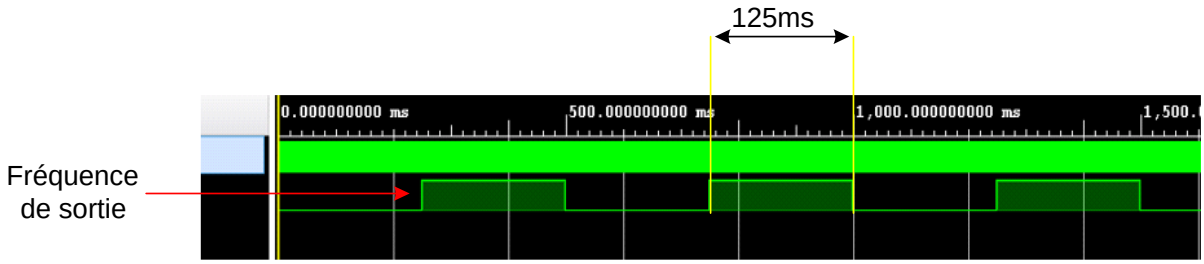


Figure 24 : Chronogramme d'horloge de sortie

Après avoir validé le diviseur de fréquence et le générateur d'horloge global, il a été confirmé que le diviseur fonctionne correctement lorsqu'il est intégré avec le compteur et la machine d'états.

Le système génère le signal d'horloge de manière fiable, ce qui assure le bon fonctionnement de l'ensemble du système comme indiqué dans la figure 25.

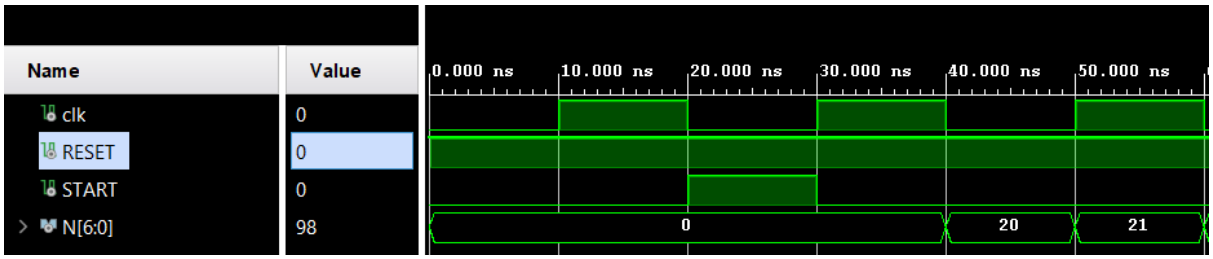


Figure 25 : Validation de diviseur dans le système global

Cette validation renforce la fiabilité et la stabilité de notre système global de feux de circulation.

2. Validation des boutons de mise en marche :

Lors du test de démarrage du système de feux de circulation, il a été observé que lorsque le bouton "Start" est activé, le compteur du système commence à compter. Cela indique que le système réagit correctement à l'action du bouton et commence à effectuer le processus de comptage des cycles de feux de circulation comme illustré dans la figure ci-dessous :

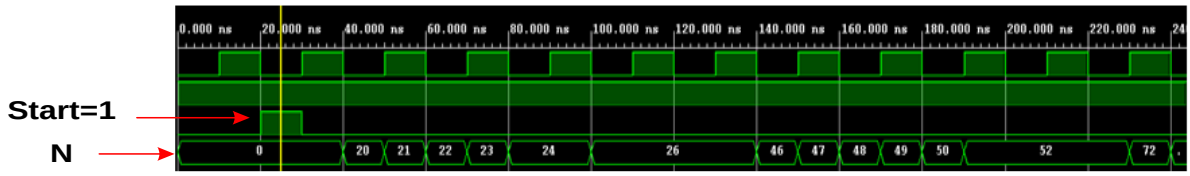


Figure 26 : Validation du bouton Start

Lors du test de la fonction de réinitialisation du système de feux de circulation, il a été constaté que lorsque le signal de réinitialisation est activé, les feux de couleur orange s'allument. Cela indique que le système réagit correctement à la réinitialisation et passe à l'état initial où les feux de couleur orange sont allumés comme indiqué dans la figure 26. Ce comportement confirme que la fonction de réinitialisation fonctionne comme prévu dans le code de la machine d'états.

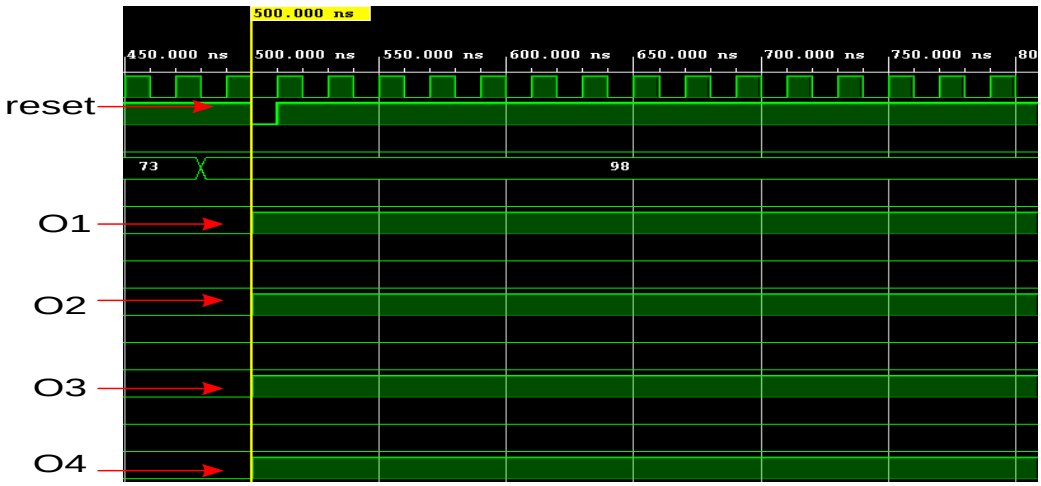


Figure 27 : Validation du bouton Reset

Lors du test de la priorité de la fonction de réinitialisation par rapport au bouton de démarrage, il a été constaté que lorsque le signal de réinitialisation est activé, il a la priorité sur le bouton de démarrage, indépendamment de son état (0 ou 1).

Cela signifie que lorsque le signal de réinitialisation est actif, le système se réinitialise immédiatement, peu importe l'état du bouton de démarrage.

Ce comportement confirme que la fonction de réinitialisation a la priorité sur le bouton de démarrage, conformément aux spécifications du système de feux de circulation comme : indiqué dans la figure ci-dessous :

Figure 28: Validation de priorité

Après avoir effectué la validation de notre système de feux de circulation sur une carte PCB, nous confirmons que celui-ci fonctionne correctement comme indiqué dans la figure 29 :

Les différentes fonctionnalités du système, telles que les feux rouges, verts et oranges, s'allument et s'éteignent aux moments appropriés, respectant ainsi les règles de circulation attendues.

VII. Conclusion:

Ce chapitre a permis la réalisation d'un feu de circulation basique en utilisant le diviseur de fréquence, le compteur et la machine d'états sur la carte Pynq-Z2. Nous avons acquis une compréhension approfondie de ces composants et de leur intégration dans un système plus large. Ce projet nous a permis de développer nos compétences en VHDL, en conception matérielle et en validation de système.

Dans le prochain chapitre, nous explorerons l'utilisation de l'intelligence artificielle pour rendre nos feux de circulation intelligents. Nous mettrons en œuvre l'algorithme OPENCV en utilisant la puissance de calcul de la carte Pynq-Z2 pour une détection précise des véhicules.

Chapitre 3 : Intégration d’IA

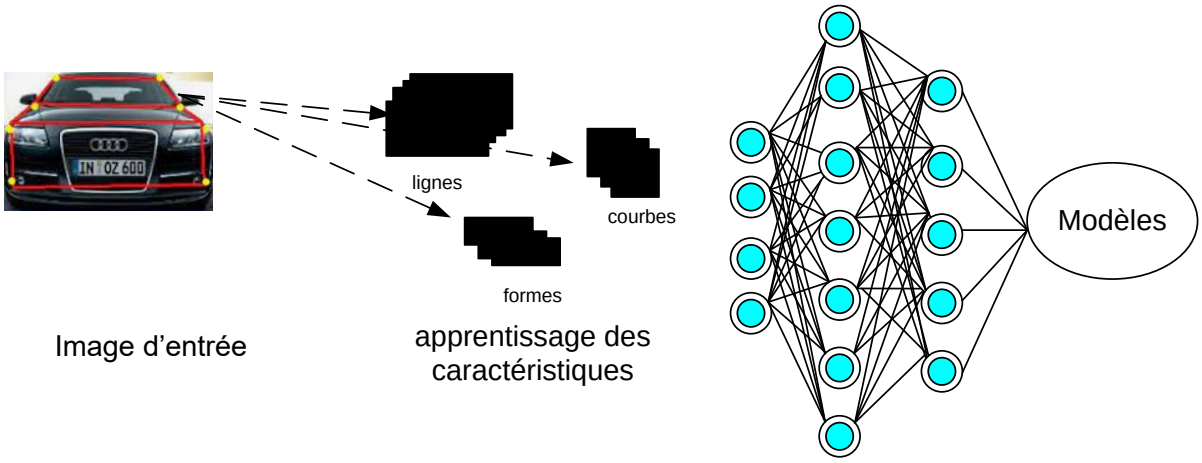
I. Introduction :

Dans ce chapitre notre système est utilisé pour détecter et compter les véhicules. Il s'agit d'une application de la vision par ordinateur, une branche de l'intelligence artificielle qui traite de la capture et de l'analyse d'informations, des algorithmes d'apprentissage en profondeur. Le processus de détection de véhicule commence par la capture d'images à partir d'une vidéo, qui est ensuite analysée par l'IA pour détecter les véhicules. PYNQ offre la possibilité de concevoir et de tester rapidement de telles applications en combinant la facilité d'utilisation de Python avec les performances du traitement matériel programmable. Cela permet au système de traiter les images en temps réel avec une latence minimale, une caractéristique essentielle pour de telles applications.

II. Les différentes méthodes de comptage de véhicule

1. Comptage des véhicules basée sur CNN :

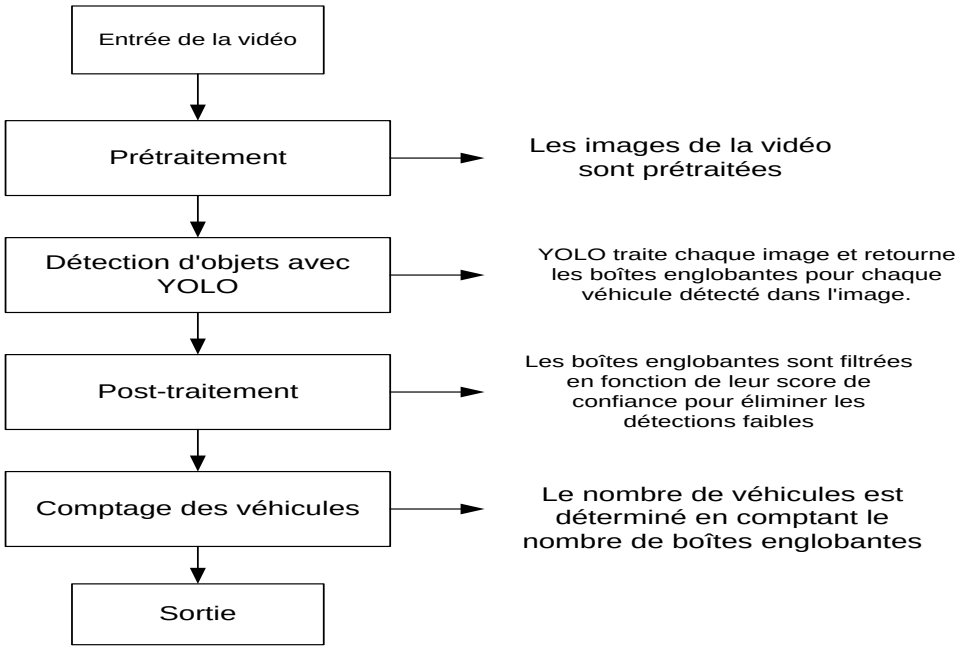
Le système de comptage de véhicules basé sur CNN (Réseau neuronal convolutif) commence par une vidéo en entrée, dont chaque image est prétraitée selon les exigences de l'algorithme CNN. Les CNN sont ensuite utilisés pour extraire les caractéristiques des images. Ces caractéristiques sont ensuite passées à une couche de classification, qui détermine si un véhicule est présent ou non dans une certaine région de l'image. Les détections sont ensuite post-traitées pour éliminer les fausses détections et les détections multiples du même véhicule. Le nombre de véhicules est ensuite déterminé en comptant le nombre de détections positives restantes après le post-traitement [Dans Latex]. L'architecture de CNN est indiquée dans la figure :



2. Détection de véhicule avec YOLO :

YOLO (You Only Look Once) est un modèle de détection d'objets préformé qui a récemment été développé. Il est considéré comme le modèle le plus performant en termes de vitesse et de précision. Étant donné que la vitesse et la précision sont des facteurs critiques dans le comptage des véhicules à partir de vidéos préenregistrées, YOLO est particulièrement adapté à cette tâche. [Dans Latex]

Le fonctionnement d'algorithme de YOLO est décrit dans l'organigramme :



2.1. Modèle de détection des véhicules avec YOLOV3

Pour entrainer un modèle YOLO afin de détecter les véhicules, nous devons suivre plusieurs étapes clés, Tout d'abord, préparer un ensemble de données comprenant des images de véhicules avec leurs emplacements. Entraîner le modèle sur plus de 100 images peut fournir des résultats initiaux satisfaisants :

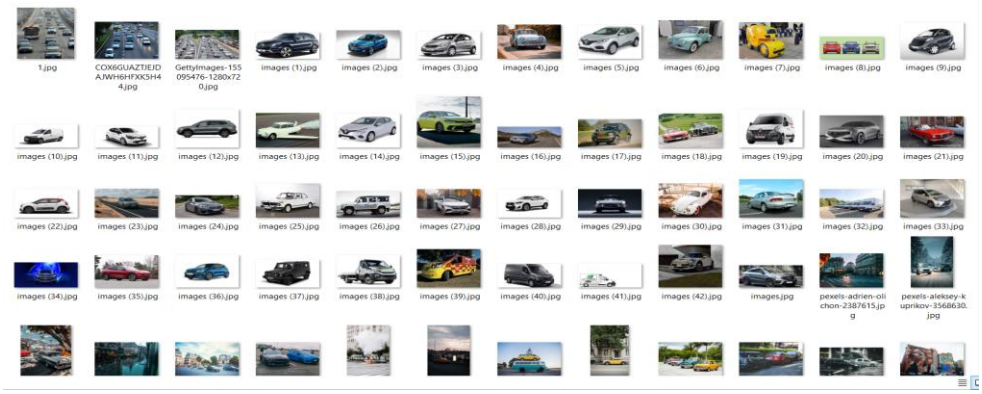


Figure : Dans Latex

Après la collection des images nous devons également spécifier où se trouve l'objet personnalisé sur l'image spécifique. Nous aurons besoin un logiciel **labellmg ;** Ce logiciel permet de marquer les véhicules dans les images en créant des boîtes englobantes précises autour de chaque véhicule comme indique la figure suivante :

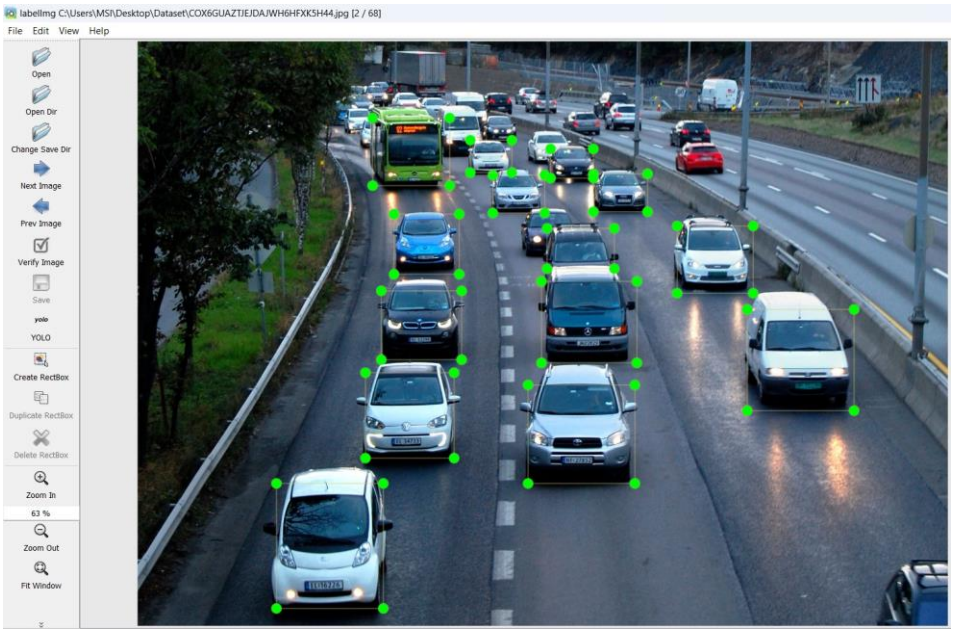
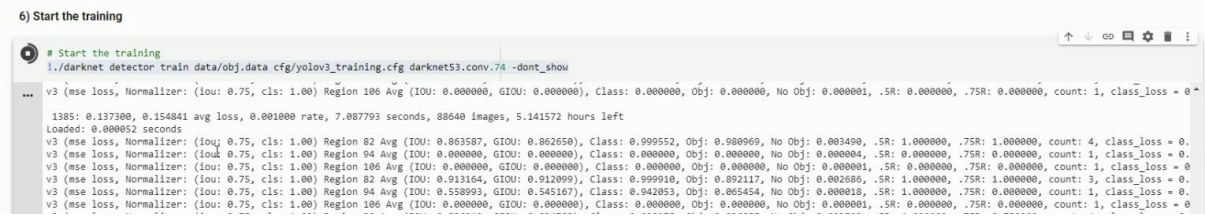


Figure :

À la fin de cette opération, chaque image du dossier devrait être associée à un fichier .txt portant le même nom. Ces fichiers .txt contiendront les annotations correspondantes pour chaque image spécifique.

Nous entrainons maintenant les données d'images en utilisant le serveur **google colab** car ce dernier offre un service gratuit pour l'apprentissage automatique. Nous devons installer la Framework 'Darknet' qui est utilisé pour exécuter et former YOLO comme illustre la figure :



Après la fin d'exécution d'entrainement de notre modèle nous avons l'accès de télécharger les fichier (yolov3.weights et yolov3.cfg) pour tester notre modèle avec OpenCV en utilisant la fonction (cv2.dnn.readNet ("yolov3.weights" , "yolov3.cfg")).

D’ailleurs l’image de la carte PYNQ Z2 ne prend en compte la version d’OpenCV convenable pour lire ces fichiers. Pour cela nous avons créé un modèle de détection et comptage des véhicules répond à nos besoins et compatible avec notre version

3. Système d’IA de comptage des véhicules basée sur OpenCV

OpenCV (Open **C**omputer **V**ision **L**ibrary) est une bibliothèque open source de vision par ordinateur et d'apprentissage automatique pour Python. Il est livré avec de nombreux algorithmes optimisés qui peuvent être utilisés pour détecter, identifier des objets, suivre des objets en mouvement, créer des cadres de délimitation (bounding box) [Dans Latex]

Environnement :

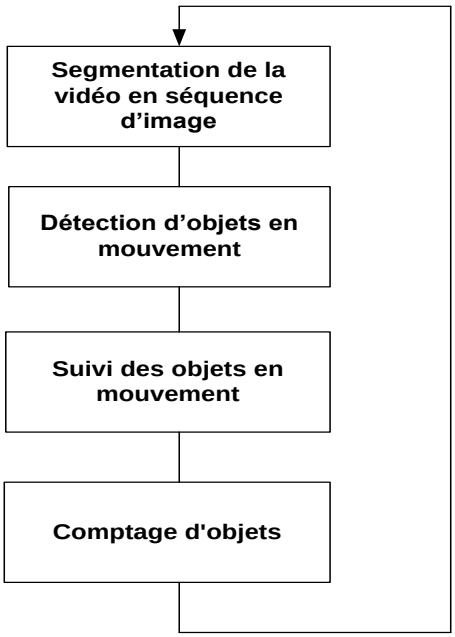
Dans le cadre de notre projet, nous avons utilisé l'environnement Python Jupyter sur la carte de développement PYNQ Z2.

Cependant, nous devons noter que notre carte de développement a un système avec une image spécifique qui n'accepte pas l'entraînement d'algorithmes avancés. C'est pourquoi nous avons choisi d'utiliser OpenCV, car il offre une grande flexibilité et une gamme d'outils qui nous permettent de développer une solution efficace pour le comptage de véhicules, même sans la possibilité d'utiliser des techniques d'apprentissage profond plus avancées.

Même que Xilinx dès maintenant ne développe pas l’image de PYNQ Z2.

La détection d'objets joue un rôle clé dans le développement de systèmes de feux de circulation intelligents. Ces systèmes utilisent l'IA et la vision par ordinateur pour améliorer l'efficacité et la sécurité du trafic routier.

A partir d'une entrée vidéo, un système de feux de signalisation intelligents peut non seulement détecter et suivre les objets, mais aussi les compter et analyser leur comportement, le fonctionnement du système est détaillé dans la figure :



I. Detection d'objets en mouvement

Avant de commencer la détection d'objet il faut d'abord transformer la vidéo en images séparées appelés « frame » le travail va se faire sur ses dernières. On a utilisé deux méthodes de détection d'objets la première c'est par la différence d'images et la deuxième par la méthode du modèle de mélanges gaussiens. La détection des objets en mouvement utilise un algorithme de soustraction de l'arrière-plan. [Dans latex]

1.1. Découpage d'objets avec OPENCV :

Pour commencer le processus de découpage des objets, nous allons exécuter le programme qui va capturer une image de la vidéo sélectionnée pour le traitement. Cela permettra d'effectuer les opérations de découpage sur une image spécifique extraite de la vidéo ZIED.mp4 [reférence] :

Figure : Image prise de la vidéo

Une fois que nous obtenons l'image capturée dans la boucle de lecture de la vidéo, nous avons appliqué les étapes suivantes pour isoler les objets :

-Convertissez l'image en niveaux de gris comme indique la figure :

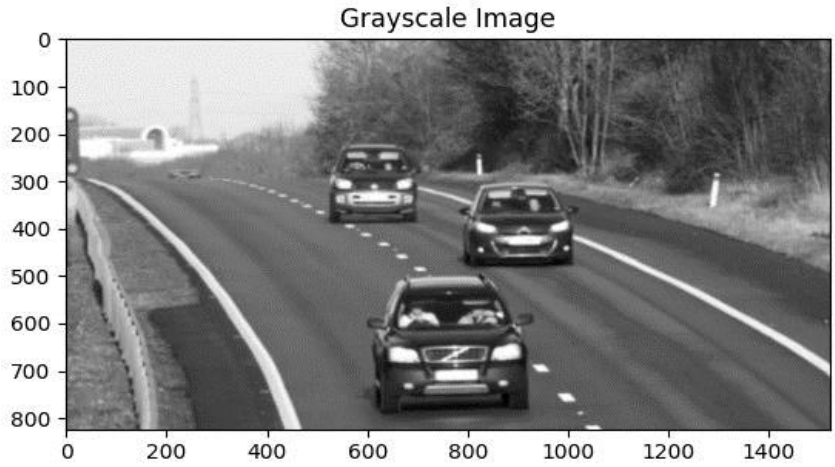


Figure : Image convertissée en niveaux de gris

-binariser l'image et séparer les objets du fond comme indique la figure :

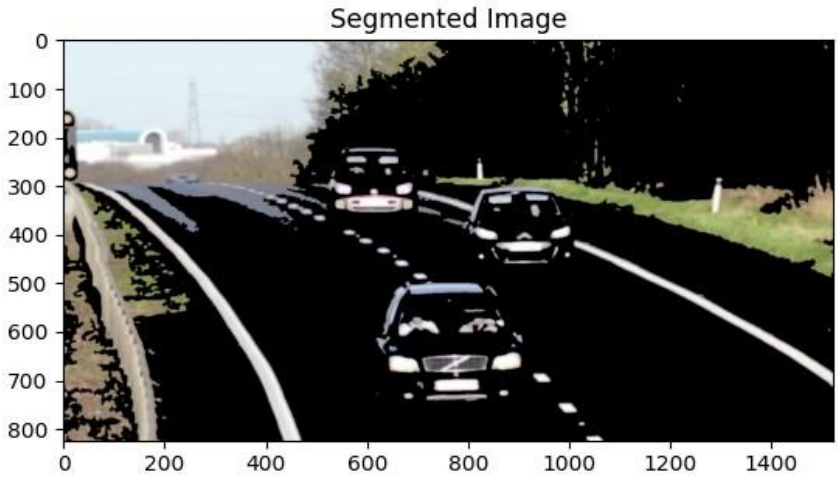


Figure : Image segmentée [Dans Latex]

-trouver les contours des objets dans l'image binarisée.

- filtrer les contours en fonction de leur taille, leur forme ou d'autres critères en utilisant des opérations de filtrage comme indique la figure :

Figure : Minimisations des bruits de l'image [Dans Latex]

-obtenir les coordonnées des rectangles englobants autour des contours

-découper les objets en utilisant ces coordonnées en appliquant les fonctions findContours, drawContours et boundingRect comme indique la figure :

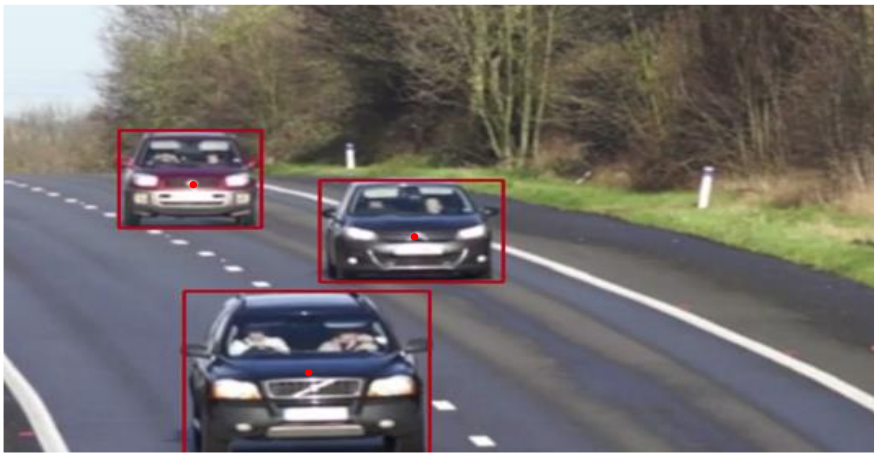


Figure : Détection des formes

Pour chaque contour valide, un rectangle est dessiné autour de l'objet détecté (cv2.rectangle), et le centre du rectangle est calculé avec la fonction (pega_centro). Les centres sont stockés dans la liste (detec) pour effectuer le suivi des objets détectés.

II. Comptage des véhicules :

Pour chaque centre dans detec, on vérifie si le centre y est proche de la ligne de détection (pos_linha) avec une marge d'erreur spécifiée par offset. Si un objet traverse la ligne, le nombre de voitures est incrémenté (carros += 1), et le centre est retiré de la liste detec. Le nombre total de voitures détectées est affiché à l'écran comme illustre la figure :

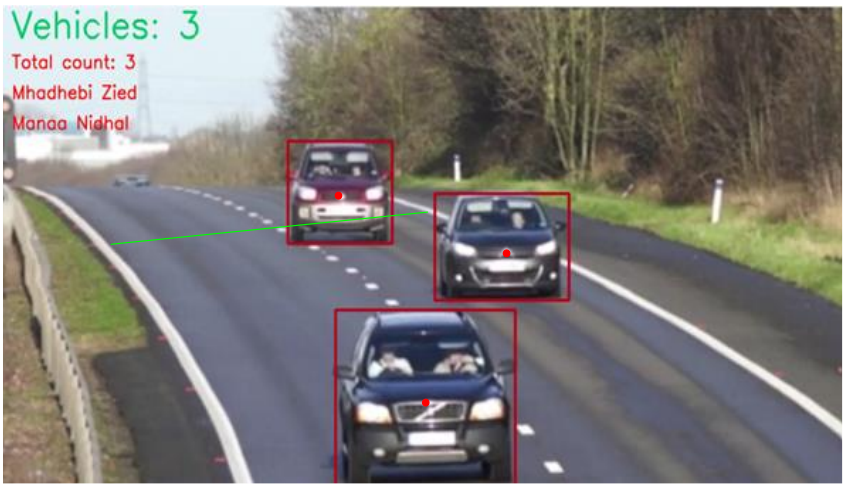
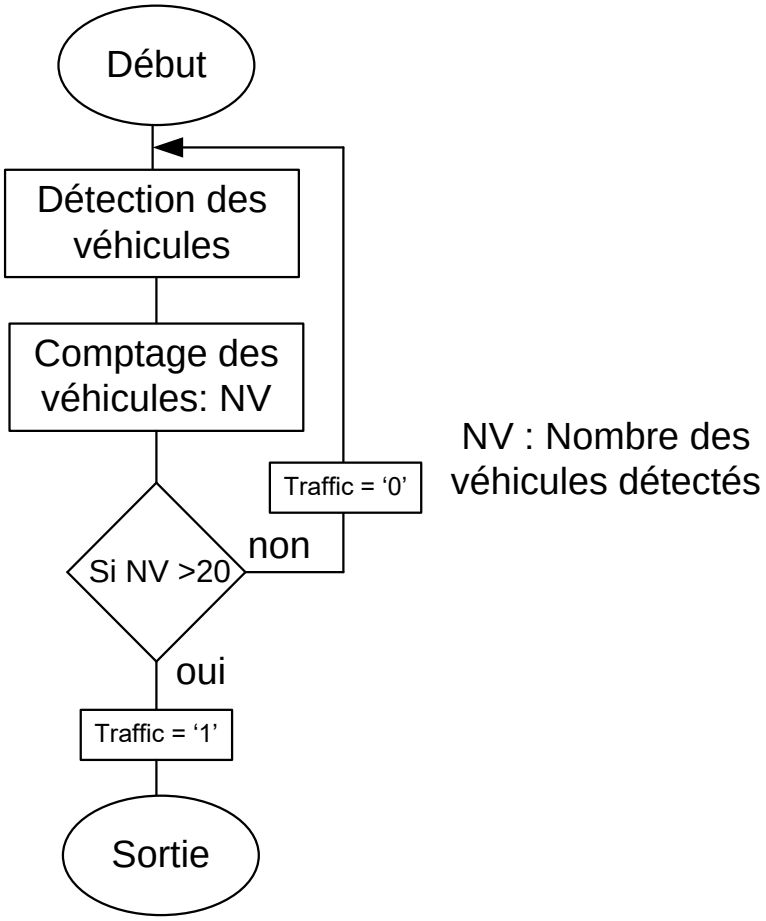


Figure : traçage de ligne et pointage des objets

Le comptage des véhicules est exploité pour définir l'état routier du carrefour. Nous avons affecté une sortie dans ce programme qui s'appelle 'Traffic', la description du système est indiquée par la figure :



III. Interface de PS et PL :

La détection et le comptage des véhicules se fait dans l'environnement Jupyter de la carte PYNQ Z2 dans la partie PS (Processor System). Ainsi le traitement de base de notre carrefour est fait dans la partie PL (Programmable Logic). Notre sortie de comptage va gérer le carrefour, donc nous sommes besoin d'interfacer cette sortie avec la partie PL comme indiqué par la figure :

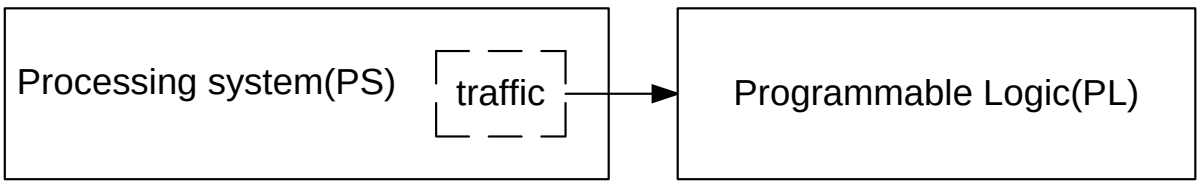


Figure :

Solution :

Le dispositif Zynq a jusqu'à 64 GPIO (General Purpose Input/Output, soit Entrée/Sortie à usage général) du PS (Processing System, soit Système de traitement) vers le PL (Programmable Logic, soit Logique programmable). Ces GPIO peuvent être utilisés pour des opérations de contrôle simples. Les GPIO du PS sont une interface très simple et il n'y a pas besoin d'IP (Intellectual Property, soit Propriété intellectuelle) dans le PL pour les utiliser. [Dans Latex]

Après avoir démarré la logique programmable (PL) de votre FPGA avec le design du feu de circulation, vous avez créé un nouveau design permettant de contrôler l'entrée du trafic depuis la PL via le système de processeur (PS). Ce design comprend un feu de circulation normal et un système de processeur supplémentaire qui a été téléchargé sur le FPGA. Les broches d'entrée/sortie générales (GPIO) du FPGA sont connectées à l'entrée du trafic
La figure illustre l'interface PS/PL dans Vivado :

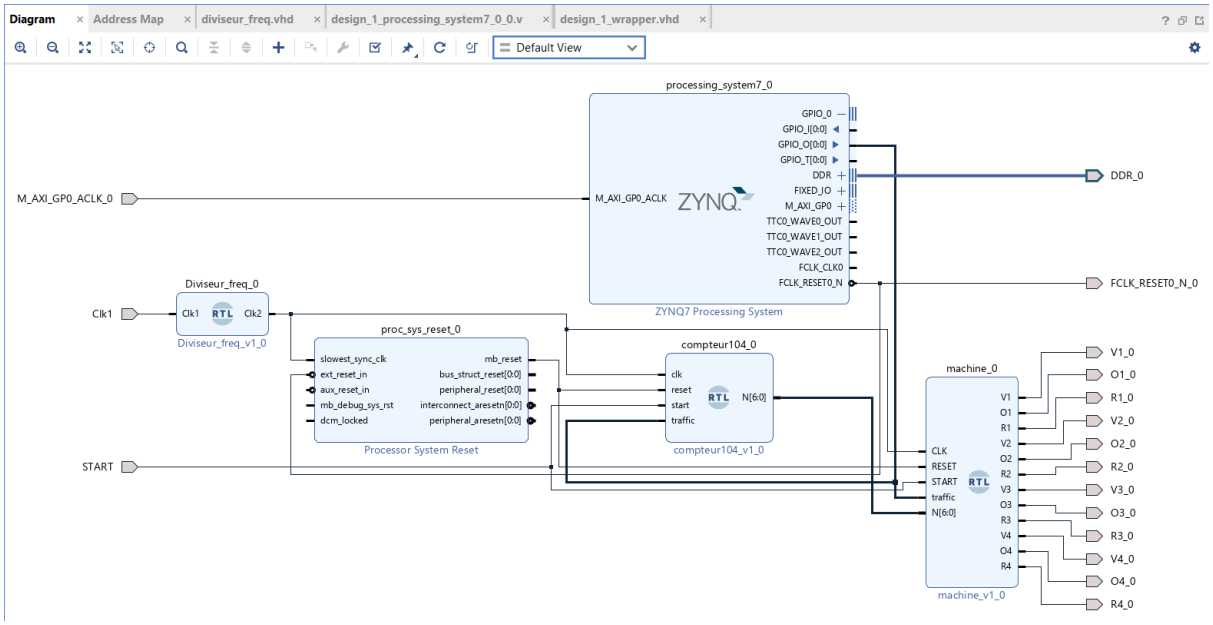


Figure : Interface PS/PL

Overlays :

Les superpositions, ou bibliothèques matérielles, sont des conceptions FPGA programmables/configurables qui étendent l'application utilisateur du système de traitement du Zynq à la logique programmable. Les superpositions peuvent être utilisées pour accélérer une application logicielle ou pour personnaliser la plate-forme matérielle pour une application particulière. [Dans Latex]

Après l'exportation de fichier (.bit), nous avons passé par ces superpositions pour que le PS exécute le fichier bitstream.

PYNQ fournit une interface Python pour permettre aux superpositions dans le PL d'être contrôlées à partir de Python exécuté dans le PS comme indiqué dans la figure :

```
In [ ]: from pynq import Overlay

        overlay = Overlay('path/to/bitstream.bit')
```

Le GPIO du PS (Système de traitement) utilise un module du noyau Linux pour contrôler le GPIO. Cela signifie que les systèmes d'exploitation assignent un numéro au GPIO lors de son exécution. Avant d'utiliser le GPIO du PS, il faut associer le numéro de la broche Linux à l'instance GPIO Python.

La fonction get_gpio_pin (), qui fait partie de la classe GPIO, est utilisée pour associer le numéro de la broche du PS au numéro de la broche Linux. Voici un exemple de la façon dont cette fonction peut être utilisée dans la figure :

```
In [ ]: from pynq import GPIO
        traffic = GPIO(GPIO.get_gpio_pin(0) , 'out')
```

Figure :

Une fois les étapes mentionnées précédemment franchies, nous sommes maintenant en mesure de contrôler le feu de circulation à l'aide d'un signal provenant d'un code d'IA. Le fonctionnement général de système est illustré par la figure :

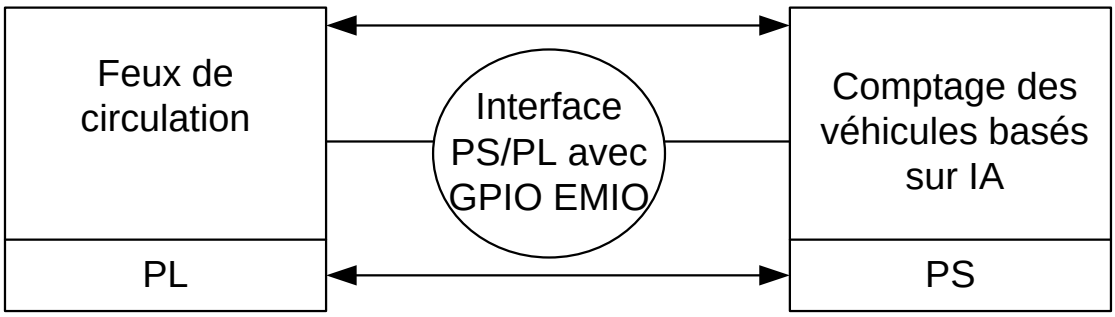


Figure : Interface PS/PL

Conclusion :

À la fin de ce chapitre, nous avons effectué avec succès la détection et le comptage des véhicules. Nous avons ensuite créé une interface entre le système de traitement du FPGA (PS) et la logique programmable (PL). L'interface est créée à l'aide de broches GPIO (EMIO). Cette intégration, permette de traiter les données de détection de véhicules en ajustant les séquences de feux de signalisation pour optimiser la fluidité du trafic et améliorer la sécurité routière.

Perspectives

Dans le cadre de mes travaux sur le projet avec PYNQ, j'ai acquis une expérience précieuse dans l'utilisation de cette plate-forme FPGA pour le développement de systèmes embarqués. Cependant, je souhaite maintenant explorer de nouvelles perspectives en utilisant la carte Zedboard.

La Zedboard est une autre carte de développement FPGA populaire, offrant des fonctionnalités similaires à celles de PYNQ. En utilisant la Zedboard, je pourrai continuer à exploiter les capacités de la FPGA pour la conception et l'implémentation de systèmes complexes. [Dans Latex]

Horloge source de Zedboard :

La fréquence d'horloge par défaut pour la logique programmable (PL) dans le SoC Zynq-7000, utilisé dans le Zedboard, est de 100 MHz. Cette fréquence d'horloge est dérivée de la source d'horloge principale générée par les processeurs ARM Cortex-A9 dans la section du système de traitement (PS).

Diviseur de fréquence :

Pour générer une fréquence de sortie plus basse, FS(z), à partir de cette fréquence d'entrée, FE(z), vous pouvez utiliser la formule suivante :

FS(z) = FE(z) / (Nd + 1)

Dans notre cas, vous souhaitez obtenir une fréquence de sortie cible de FS(z) = 0,01 Hz.

Pour cela, nous pouvons attribuer une valeur de division, Nd, telle que :

Nd = FE(z) / FS(z) – 1

Nd = 100 MHz / 0,01 Hz – 1

Nd ≈ 99 999 999

Ainsi, en utilisant une valeur de division de Nd = 99 999 999, nous devrions obtenir une fréquence de sortie de FS(z) ≈ 0,01 Hz à partir de l'horloge d'entrée de 100 MHz du Zedboard.

Conclusion générale

Ce projet nous a permis d'explorer les défis et les limitations des systèmes de feux de circulation actuels, ainsi que de développer une solution de feux de circulation intelligents basée sur l'intelligence artificielle.

La programmation du FPGA a été réalisée en utilisant des langages de description matérielle tels que VHDL. Nous avons développé une compréhension approfondie de ce langage et des concepts de conception matérielle nécessaires pour configurer le FPGA selon nos besoins. De plus, nous avons réalisé les configurations nécessaires pour l'intégration du FPGA avec le reste du système de traitement ou PS et les interfaces GPIO (EMIO) pour la communication avec les feux de circulation. Cette maîtrise des configurations du FPGA et du système dans son ensemble a été essentielle pour atteindre notre objectif de mise en place d'un système de feux de circulation intelligents performant et fonctionnel.

L'intégration de l'intelligence artificielle avec la logique programmable du FPGA nous a offert une plus grande flexibilité et adaptabilité. Nous avons pu ajuster dynamiquement les séquences des feux de circulation en fonction des conditions de trafic en temps réel, ce qui a contribué à une meilleure fluidité du trafic et à une réduction des congestions.

Références

[1] https://www.medicis-patrimoine.com/actualites-immobilier-neuf/marche-de-l-immobilier/2020/05/19/3097-l-intelligence-artificielle-au-service-de-la-smart-city.html

[2] https://www.realites.com.tn/fr/article/87827/samsung-devoile-la-station-smartthings-au-ces-2023

[3] https://www.amazon.ca/-/fr/b?ie=UTF8&node=21497225011

[4] https://fr.heliox-energy.com/blog/smart-energy-how-ev-charging-enables-smart-energy-management

[5] MEMOIRE ´ DE MASTER PROFESSIONNEL En Informatique Option Administration et Sécurité des Réseaux Thème d Réalisation d'un parking intelligent Présenté par : Mr. Alia Lounes Mr. Ghersa Elyes Soutenu le 01 Juillet 2017-B´ejaia, Juillet 2017

[7] MINISTERE DE L'ENSEIGNEMENT SUPEREUR ET DE LA RECHERCHE SCIENTIFIQUEF-Faculté des Sciences et de la Technologie DEPARTEMENT DE GENIE ELECTRIQUEMEMOIRE Présenté pour obtenir le diplôme de MASTER EN ELECTRONIQUE DES SYSTEME EMBARQUES-Par Ould Mammar Mohammed Elamine Midoun Mohamed Essedik-Réalisation d'un système embarqué à base d'un Raspberry pour le contrôle d'accès à un parc automobile-Soutenu le 02 juillet 2020

[8] https://fr.digi.com/blog/post/introduction-to-smart-transportation-benefits

[9]https://www.linkedin.com/pulse/syst%C3%A8me-de-transport-intelligent-et-

mobilit%C3%A9-30-d%C3%A9finition-hache/?originalSubdomain=fr

[10] http://www.scoot-utc.com/documents/1_SCOOT-UTC.pdf

[11] M. Abbas, H. Charara, N. Chaudhary, and Y. s. Jung. Distributed architecture and algorithm for robust real-time progress evaluation and improvement. Texas Transportation Institute, Texas A & M University System, 2006

[12] M. Selinger and L. Schmidt. A review of the cost, maintenance and reliability of popular adaptive traffic control technologies. Adaptive Traffic Control Systems in the United States, sep 2009

[13] https://www.rtbf.be/article/la-minute-insolite-combien-de-temps-passons-nous-dans-une-vie-a-attendre-au-feu-rouge-11022553

[14] D. Robertson and R. Bretherton. Optimizing networks of traffic signals in real time-the scoot method. IEEE Transactions on Vehicular Technology, 40(1) :11 –15, Feb. 1991.

[15] I. KUON, R. TESSIER, J. ROSE, « FPGA Architecture: Survey and Challenges », Foundations and Trends in Electronic Design Automation, Vol. 2, No, 2 (2007) 135-253.

[16]https://www.ummto.dz/dspace/bitstream/handle/ummto/578/NACHEF%20Toufik.pdf?sequence=1&isAllowed=y

[17] D. SAPTONO, « Conception d'un outil de prototypage rapide sur le FPGA pour des applications de traitement d'images », Thèse de Doctorat, Université de BOURGOGNE, Faculté de Science et Technologie E2S, 4 Novembre 2011.

[18]https://www.ni.com/docs/en-US/bundle/labview-nxg-fpga-targets/page/configurable-logic-blocks.html

[19] XILINX : https://www.xilinx.com.

[20]https://www.xilinx.com/products/designtools/vivado.html#:~:text=Vivado%20IP%20Integrator%20provides%20a,well%20as%20your%20own%20IP.

[21] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” The Journal of physiology, vol. 160, no. 1, pp. 106–154, 1962

[22] https://theses.hal.science/tel-03370137/document

[23]https://medium.com/egen/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9

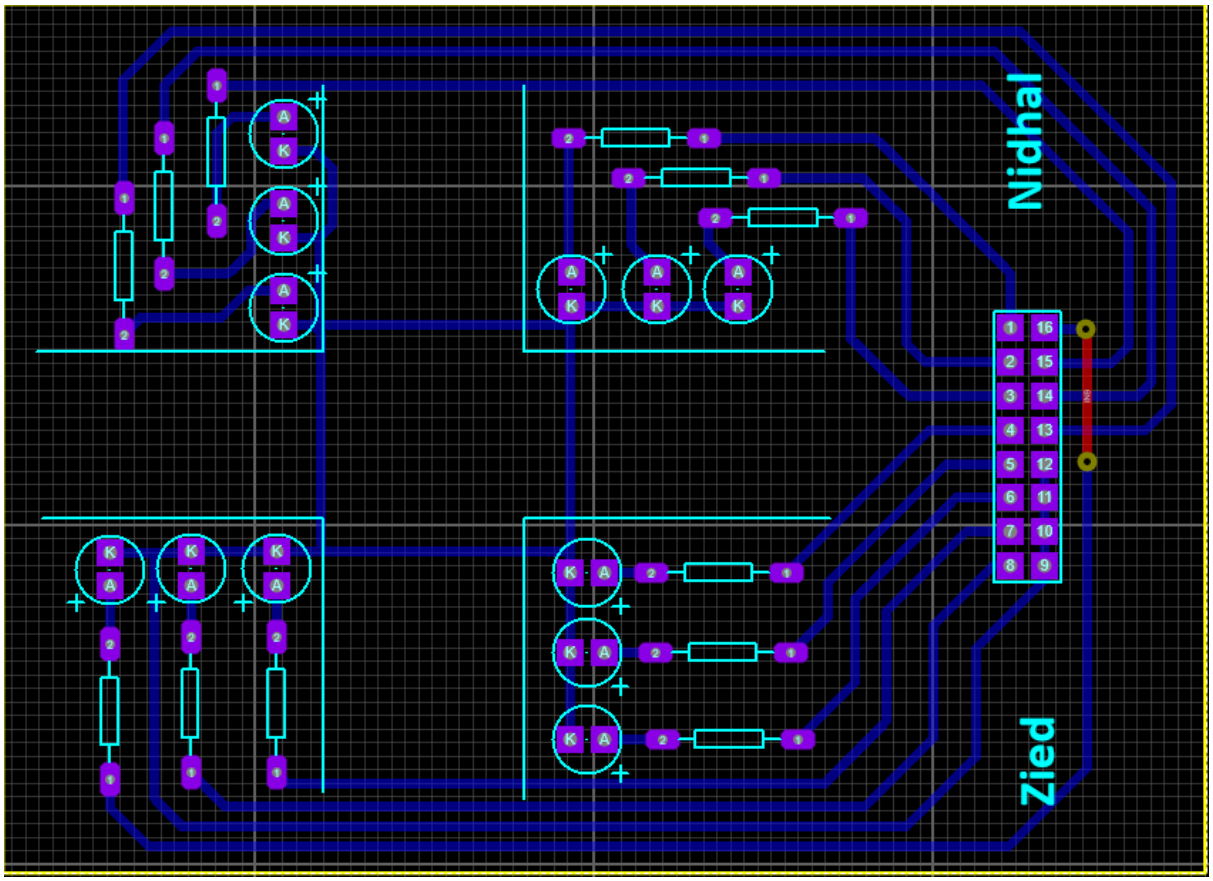
[24] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Advances in neural information processing systems, 2015, pp. 91– 99

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[26] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” arXiv preprint, 2017.

[27] ——, “YOLOv3: An incremental improvement,” arXiv, 2018.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to sift or surf,” in 2011 International conference on computer vision. Ieee, 2011, pp. 2564–2571.


Nidhal
Zied

Annexes

SoC	ZYNQ XC7Z020–1CLG400C
ARM Processor	650MHz dual-core Cortex-A9 processor
Memory Controller	DDR3 memory controller with 8 DMA channels and 4 high-performance AXI3 slave ports

High-Bandwidth Peripheral Controllers	1G Ethernet, USB 2.0, SDIO
Low-Bandwidth Peripheral Controllers	SPI, UART, CAN, I2C
Programming Interfaces	JTAG, Quad-SPI flash, MicroSD Card
Logic Slices	13,300 — Each slice is with four 6-input LUTs and 8 flip-flops
Block Ram	630 KB (Fast)
Switches, push-buttons, and LEDs	- 4x push-buttons - 2x slide switches - 4x LEDs - 2x RGB LEDs
Power	Powered from USB or any 7V-15V source
Expansion Connectors	- 2x Pmod ports - 16x FPGA I/O

Audio and Video	- Microphone: Electret microphone with pulse density modulated (PDM) output - Audio Jack: Line-in with 3.5mm jack - HDMI : HDMI sink port (input), HDMI source port (output) - I2S interface with 24-bit DAC with 3.5mm TRRS jack (Différent de Z1)
Raspberry Pi connector	- 28x Total FPGA I/O (8 shared pins with Pmod A port) (Différent de Z1)

Tableau 1 : caractéristiques de Soc-Zynq-7000

Nom du signal	**Description**	**Code pin**	**ZYNQ pin**
JA1_P	I/O	Pmod A : 1	Y18
JA1_N	I/O	Pmod A : 2	Y19
JA2_P	I/O	Pmod A : 3	Y16
JA2_N	I/O	Pmod A : 4	Y17
JA3_P	I/O	Pmod A : 7	U18
JA3_N	I/O	Pmod A : 8	U19
JA4_P	I/O	Pmod A : 9	W18
JA4_P	I/O	Pmod A : 10	W19
JB1_P	I/O	Pmod B : 1	W14
JB1_N	I/O	Pmod B : 2	Y14
JB2_P	I/O	Pmod B : 3	T11
JB2_N	I/O	Pmod B : 4	T10
SW0	Ø	Ø	M20

SW1	Ø	Ø	M19

Tableau 2 : Les pins spécifiés de la carte PYNQ Z2

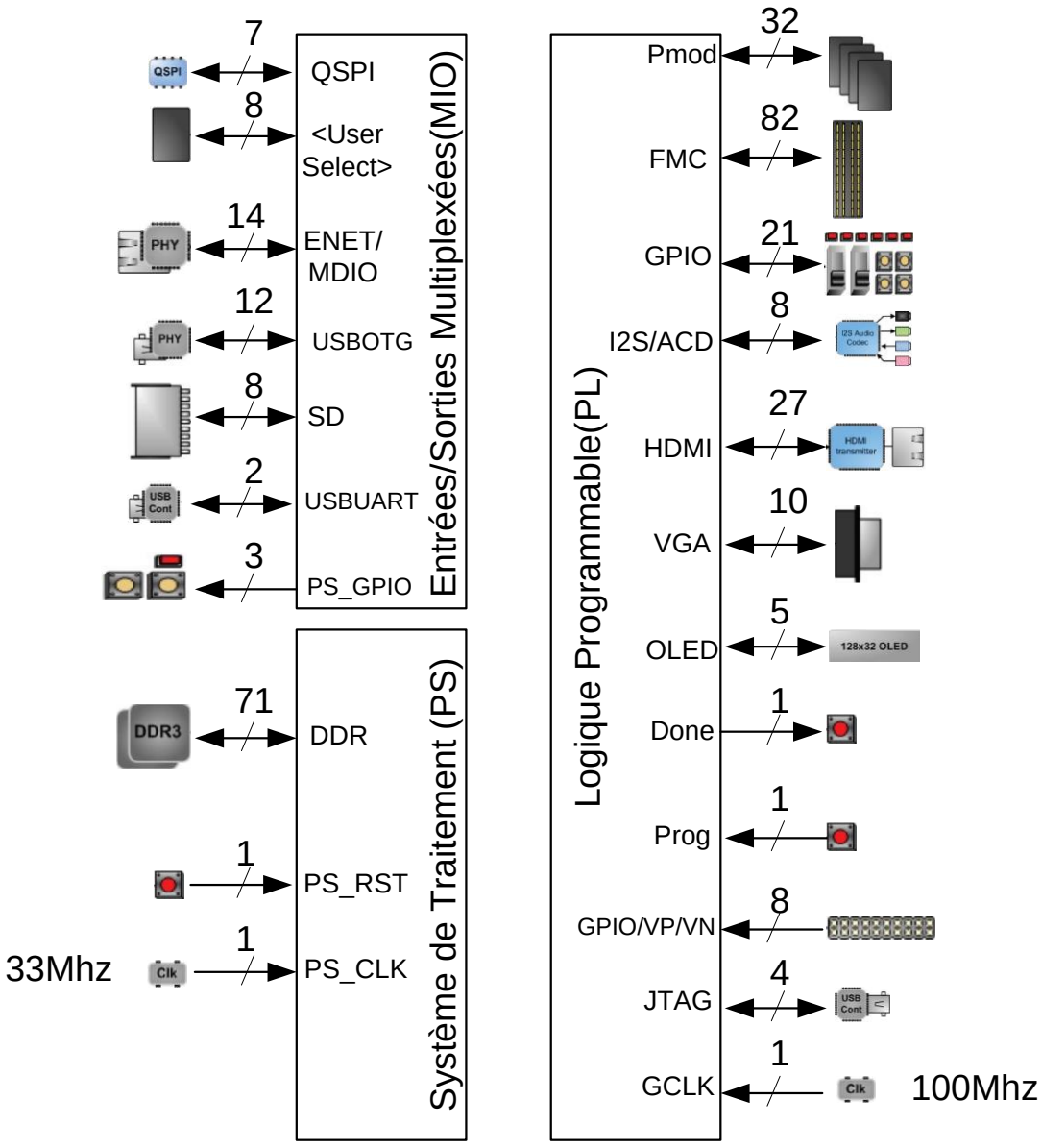


Figure : schéma bloc matériel du Zedboard

Nom du signal	**ZYNQ pin**
GCLK	Y9

Tableau 3 : Pin spécifiés de diviseur pour Zedboard