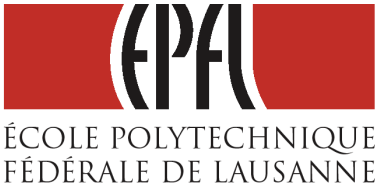

CAMERA EMBARQUEE ETHERNET / INTERNET

Projet de semestre de

Sylvain Pasini

Professeur : Pr. Paulo Ienne

Présenté le 10 février 2003

Responsable : René Beuchat

Merci

J'aimerais commencer par remercier le professeur, Paulo Ienne, ainsi que le responsable de ce projet, René Beuchat de m'avoir donné la chance de réaliser un projet dans ce laboratoire. L'important pour mener à bien un projet est d'apprécier l'environnement de travail ainsi que le sujet. Ce laboratoire m'a tout de suite mis à l'aise. De plus le projet m'intéresse énormément. Je voulais juste vous dire "Merci".

Je tiens à remercier Cédric Gaudin. Il a énormément contribué au bon déroulement de ce projet. Il m'a surtout aidé à démarrer. En effet, une partie de ce travail a consisté à apprendre les différents outils de développement, mais heureusement, par la suite il a fallu développer différents éléments. Là encore, il m'a aidé à résoudre de gros problèmes. La vue d'une personne extérieure aide toujours positivement. Cédric je voulais juste te remercier encore une fois.

Je remercie évidemment tous mes collègues du laboratoire d'architecture des processeurs pour leur bonne humeur. La motivation de chacun ainsi que l'ambiance générale était un environnement très favorable à la réalisation de projets.

Mes derniers remerciements vont bien sûr à ma famille. Merci de m'avoir soutenu et encouragé tout au long de mes études. Je remercie bien évidemment ma fiancée, Nadia, qui a réussi à m'accorder du temps libre afin que je mène à bien ce travail. Je la remercie également pour son Amour et ses encouragements qui m'ont fortement aidé tout au long de ce travail.

Un grand merci à tous …

Résumé

Table des matières :

INTRODUCTION ***1***
1.1 TRAVAIL EXISTANT 1
1.2 OBJECTIFS 1
1.3 APPLICATIONS 2
1.4 PRODUITS DU MARCHE 2
1.5 ORGANISATION DU TRAVAIL 3

MATERIEL A DISPOSITION ***4***
2.1 KIT DE DEVELOPPEMENT NIOS 4
2.2 KIT DE DEVELOPPEMENT ETHERNET POUR NIOS 5
2.3 CAMERA 2D 5
2.4 LOGICIELS 6
2.4.1 QUARTUS 6
2.4.2 SOPC Builder *6*
2.4.3 NIOS SDK *6*
2.4.3 Ultra Edit *6*
2.4.4 Visual C++ *6*
2.4.5 Logiciel CamAff *6*
2.4.6 Sniffer XP *6*

RESEAUX INFORMATIQUES ***7***
3.1 GENERALITES 7
3.2 DIVISION EN COUCHES 7
3.3 COUCHE DE LIEN : ETHERNET 8
3.4 COUCHE RESEAU : IP 9
3.5 COUCHE TRANSPORT : TCP ET UDP 9
3.6 COUCHE APPLICATION : HTTP 11

ARCHITECTURE GENERALE DU SYSTEME ***12***
4.1 SCHEMA BLOC 12
4.2 LES DIFFERENTS COMPOSANTS EN DETAILS 13
4.2.1 Module sous SoPc Builder *13*
4.2.2 Processeur Nios *14*
4.2.3 La mémoire de boot (ROM) *14*
4.2.4 UART *15*
4.2.5 Timer *15*
4.2.6 Boutons *16*
4.2.7 Les LED's *16*
4.2.8 Affichage 7 segments *17*
4.2.9 La bus de mémoire externe *17*
4.2.10 La mémoire statique (SRAM) *17*
4.2.11 La mémoire flash *18*
4.3 SCHEMA LOGIQUE 18
4.4 PLAN MEMOIRE ET INTERRUPTIONS 19

INTERFACE AVEC LA CAMERA ***20***
5.1 REMARQUES PREALABLES 20
5.2 SYNCHRONISATION SUR LES DONNEES DE LA CAMERA 21
5.3 NOUVELLE STRUCTURE 22
5.4 HORLOGE METASTABLE 23
5.5 INTEGRATION DES INTERFACES DANS LA LOGIQUE PROGRAMMABLE 23
5.6 REGISTRES DE L'INTERFACE 24
5.6.1 Registre de contrôle *24*
5.6.2 Registre d'état *25*
5.6.3 Registre des compteurs *25*
5.6.4 Registre de données *25*
5.7 UTILISATION AU TRAVERS DU PROCESSEUR 25
5.8 LIMITE DU SYSTEME 26

MODULE ETHERNET ***27***
6.1 REMARQUES PREALABLES 27
6.2 INTERFACE AVEC LE PROCESSEUR NIOS 27
6.3 LES "PLUGS" 28
6.4 INSERTION DE L'INTERFACE DANS L'ARCHITECTURE 29
6.5 UTILISATION 30

SERVEUR WEB ***31***
7.1 POURQUOI UTILISER UN SERVEUR WEB ? 31
7.2 ROLE D'UN SERVEUR WEB 31
7.3 LE SERVEUR FOURNIT PAR ALTERA 32
7.3.1 Pages statiques *32*
7.3.2 Pages dynamiques *32*
7.4 CREATION DYNAMIQUE DES IMAGES 33
7.5 REPONSE A UNE REQUETE 34
7.6 CONCURRENCE : STRATEGIES DE CAPTURE ET DE TRANSMISSION 35
7.6.1 Capture pendant le traitement de la réponse *35*
7.6.2 Capture en permanence, transmission de la dernière *35*
7.6.3 Capture juste après une réponse *36*
7.7 PAGE HTML GERANT LES RAFRAICHISSEMENTS 37
7.7.1 Page HTML pure avec la commande "refresh" *37*
7.7.2 Page HTML utilisant Javascript *37*

ADAPTATION DU SYSTEME A LA CARTE ROKAPEX ***38***
8.1 MOTIVATIONS 38
8.2 ARCHITECTURE 38
8.3 LE SERVEUR WEB 39
8.4 EN RESUME 39

CONCLUSION ***40***
9.1 RAPPELS DES OBJECTIFS 40
9.2 APPORT DE CONNAISSANCES 40
9.3 TRAVAIL EFFECTUE 40
9.4 COMPARAISON PAR RAPPORT AU MARCHE 41
9.5 TRAVAIL FUTUR 41

BIBLIOGRAPHIE ***42***

LEXIQUE ***43***
Architecture des processeurs *43*
Réseaux informatiques *43*
Internet *43*

TABLE DES FIGURES ***44***

Chapitre 1

Introduction

1.1 Travail existant

Lors d'un précédent projet, une caméra a été développée. Ce travail a d'abord débuté par la conception d'une carte d'extension permettant de "monter" la caméra sur la carte principale. Ensuite une interface a été conçue. Cette interface est composée de deux parties distinctes : La première partie est une interface permettant de configurer la caméra. C'est-à-dire de la paramétrer de manière voulue. La seconde partie est l'interface de données. Elle permet d'acquérir les images. A ce stade, les images ne sont encore pas compréhensibles. L'interface de configuration fonctionne correctement. Le problème provient de l'interface de données.

Lors d'un autre projet, un "firewall" a été conçu. Ce dispositif utilisait deux modules d'extensions permettant de se connecter au réseau Ethernet. L'objectif étant de limiter l'accès en autorisant ou interdisant certaine connections. Un module Ethernet était donc utilisé en entrée et le second était utilisé en sortie. Une partie importante de ce travail était de pouvoir configurer les autorisations en fonctions de l'adresse de l'utilisateur. Cette partie de configuration était en réalité un serveur Web permettant d'accéder à une table contenant les informations.

1.2 Objectifs

Il s'agit maintenant de développer un serveur Web. Cette application permettra d'accéder aux images provenant de la caméra. De cette manière, seul un navigateur Web sera nécessaire pour la lecture des images. Ce système devra être autonome. Cela signifie qu'il ne devra pas nécessiter un appareil externe tel qu'un ordinateur.

La structure globale du système sera de la forme suivante :

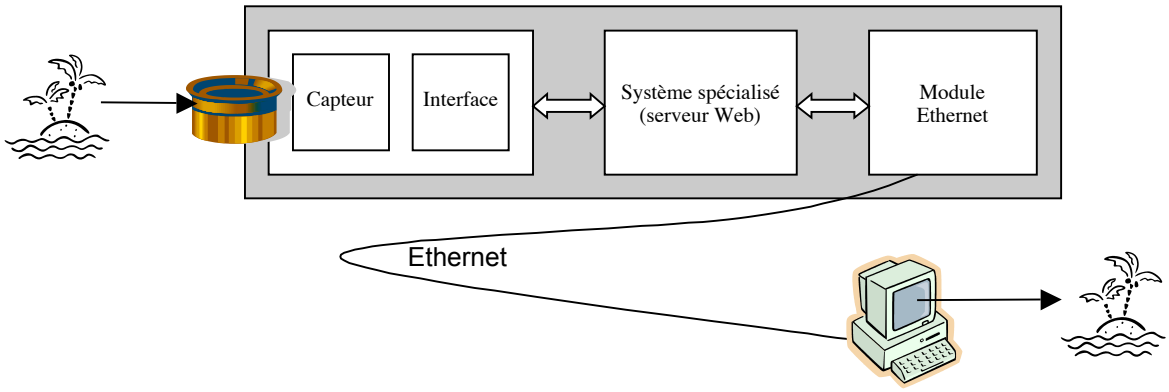

Figure 1.1 : Structure générale

Le travail à réaliser commence tout d'abord par améliorer le fonctionnement de la caméra. Dans un second temps, il s'agira de pouvoir transmettre des pages Web à travers le réseau, puis des images statiques. Ensuite, il faudra pouvoir transmettre les images provenant de la caméra. Enfin, il s'agira de développer une nouvelle carte afin d'y installer un capteur de meilleure résolution.

1.3 Applications

Un grand nombre d'applications sont envisageables. De plus en plus, les réseaux de type Ethernet se construisent. La plupart des entreprises en possède un. Certains particuliers commencent même à en avoir chez eux. Il est alors facile d'installer une ou plusieurs caméras sur un réseau. Voici une liste des applications possibles, mais il en existe bien sûre d'autres :

- Alarme :

Comme expliqué ci-dessus, beaucoup d'entreprises et de particuliers possèdent un réseau Ethernet. Il est alors possible et facile d'y connecter des caméras réseau. Ces dernières peuvent alors détecter des mouvements simplement, en comparent l'image du présent avec les images du passé.

- Surveillance de lieux publics :

Il serait envisageable d'installer des caméras réseau dans des endroits publics. Ceci permettrait de contrôler l'état des lieux facilement sans même se déplacer.

- Diagnostic à distance :

On pourrait imaginer dans une entreprise, d'installer des caméras permettant de diagnostiquer l'état de machines. Il serait également possible d'en placer afin d'identifier la provenance d'une panne.

- Alarme à incendie :

L'installation de caméra sensible à la lumière infrarouge, permettrait de détecter facilement les sources de chaleur. Dans le cas d'enregistrements, il serait alors facile de connaître la source de l'incendie. Grâce au niveau de gris de l'image, il est possible d'identifier la température. Il serait possible de déclencher une alarme lorsqu'une source de chaleur est supérieure à un certain seuil.

- Robot cyclope :

Une application interne au laboratoire serait de pouvoir visualiser les images provenant du robot tout en le guidant. Le lien entre le robot et l'utilisateur serait le serveur Web à travers la connexion Ethernet.

1.4 Produits du marché

Il ne s'agit pas ici d'énumérer chaque produit existant ainsi que ces caractéristiques. Seuls les principes de fonctionnement sont expliqués.

Une très grande partie du marché est ce que l'on appelle couramment une "webcam". Ce dispositif permet en effet de transmettre des images à travers le Web. Cependant, ce n'est pas une caméra réseau.
Le principe est simple. Une caméra est reliée par le port USB à un ordinateur. L'ordinateur est relié au réseau. Les images sont alors transmises à travers le réseau grâce à un logiciel qui permet de les capturer de la caméra.

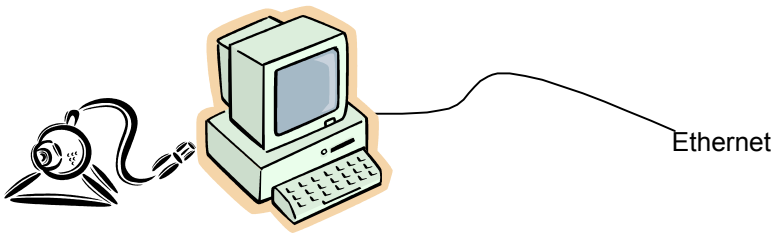

Figure 1.2 : Système utilisant un ordinateur

La popularité de ce produit provient principalement de deux critères. Le premier est qu'il est accessible à un grand nombre de clients. En effet, la majorité des foyers possède un ordinateur relier au Web à leur domicile. Le second est du à son prix qui est relativement faible.
L'inconvénient est que cette configuration n'est pas autonome. Par conséquent, elle nécessite un ordinateur.

Une autre grande partie du marché sont les caméras professionnelles. Des entreprises spécialisées fabriquent des systèmes permettant de visionner des images provenant de caméras réseau. La particularité est que les images ne sont accessibles qu'au travers d'un logiciel adapté.

Figure 1.3 : Système à logiciel spécialisé (Source : Visual Inspection Systems)

L'inconvénient est que les images ne sont accessibles qu'au travers d'un logiciel spécialisé. Il serait pratique dans certaines applications de pouvoir visualiser les images simplement à l'aide d'un navigateur Web.

1.5 Organisation du travail

La première semaine a été consacrée à l'installation de la place de travail et à la recherche de documentations. Le travail durant les deux semaines suivantes a été l'apprentissage des outils de conception ainsi que l'étude du processeur Nios. Deux semaines ont été accordées à la compréhension du module Ethernet. Les deux semaines d'après ont consisté à comprendre et faire fonctionner l'interface avec la caméra. La semaine durant la présentation intermédiaire a entièrement été consacrée à la synthèse du travail réalisé ainsi que du travail restant.
Enfin, le cœur de se projet, le développement du serveur Web. Pour terminer, il serait bien d'implémenter le tout sur la carte RokApex développée au LAP lors d'un projet de semestre. L'avantage par rapport à la carte de développement est qu'il est possible de la monter sur le robot cyclope, également conçue au LAP.

n° semaine	Date	Titre	Détail
1	21.10	Objectifs et documentation	Définir les objectifs et regrouper la documentation
2	28.10	Environnement & NIOS	Apprendre les outils de développement et le processeur NIOS
3	04.11		
4	11.11	Apprentissage Ethernet	Apprendre à utiliser le module Ethernet
5	18.11		
6	25.11	Apprentissage Caméra2D	- Apprendre à utiliser la caméra - Corriger l'interface de donnée
7	02.12		
8	09.12	Présentation intermédiaire	Réaliser la présentation intermédiaire
9	16.12	Serveur Web	Concevoir le serveur Web répondant aux objectifs fixés
10	06.01		
11	13.01		
12	20.01	Implémenter sur RokApex	Transférer toute l'architecture sur la carte RokApex
13	27.01		
14	03.02		
15	10.02	Mémoire	Peaufiner le mémoire final
16	17.02	Rendre mémoire	Date limite pour rendre le mémoire au responsable
17	24.02	Présentation - Démonstration	Préparer la présentation et la démonstration
18	03.03	Défense	

Figure 1.4 : Organisation du travail

Chapitre 2

Matériel à disposition

2.1 Kit de développement NIOS

Ce kit de développement permet l'apprentissage des outils proposés par Altera. Il contient une partie matérielle qui est en fait la carte de développement "Excalibur".
Elle contient les éléments suivants :

- Logique programmable de type APEX 20K200E
- Mémoire SRAM de 256 Ko
- Mémoire FLASH de 1 Mo
- Affichage 7 segments sur deux digits
- Affichage de 8 LED
- 2 LED supplémentaires
- 4 boutons, en plus du reset matériel et du reset logiciel
- 8 interrupteurs
- Connecteur JTAG
- Connecteur RS-232
- 6 connecteurs d'extension génériques
- 2 connecteurs PMC
- Connecteur SODIMM

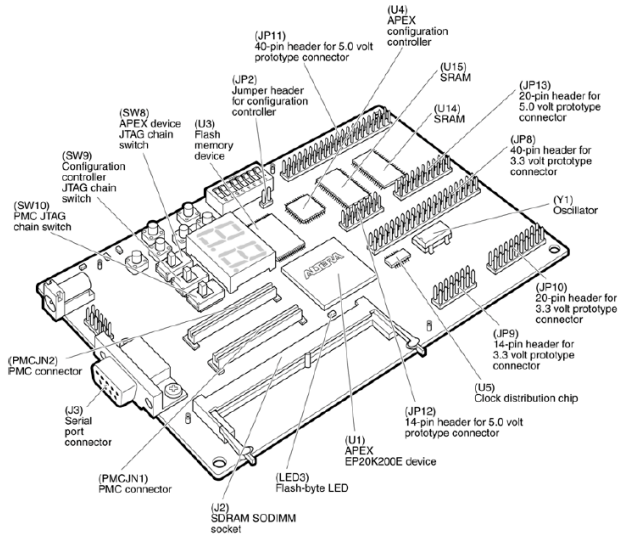

Figure 2.1 : Carte Excalibur, source [3]

Le kit contient également l'alimentation 9V qui suffit entièrement à la carte.

Une seconde partie du kit est l'ensemble des logiciels. En effet, une certaine quantité de logiciels sont fournis afin de pouvoir utiliser le kit de développement.

Une partie très importante est évidemment la documentation. Altera fournit un support relativement complet. Le tout est accessible rapidement grâce à une structure décomposée en plusieurs fichiers, chacun portant sur un sujet précis. Chaque partie utile est décrite de manière relativement détaillée.

2.2 Kit de développement Ethernet pour NIOS

Le contenu de ce kit est comparable à celui présenté précédemment. Il contient une partie matérielle, qui est en fait une carte fille venant s'adapter au connecteur génériques de la carte Excalibur. Elle contient principalement :

- Connecteurs génériques permettant de l'interfacer avec la carte Excalibur et éventuellement des cartes supérieures
- Connecteur RJ-45 pour la liaison au réseau Ethernet
- Circuit logique pour les fonctions réseau distribué par la société "Cirrus Logic" (CS8900A)

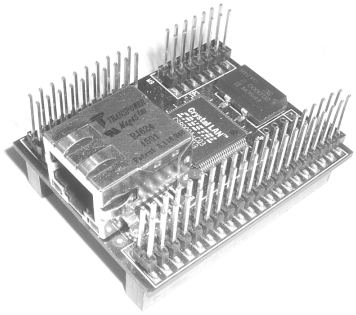

Figure 2.2 : Carte Ethernet

Le kit contient également un câble réseau et un adaptateur. Ce dernier permet de relier directement la carte Ethernet à une autre carte réseau, ceci sans passer par l'intermédiaire du type Hub.
Le kit contient évidemment une documentation très complète.
Les fichiers permettant de configurer l'interface avec le processeur Nios au travers de SOPC Builder sont également fournis.
Une librairie permettant de gérer les différents protocoles réseaux est fournies. Elle est livrée sous forme de codes sources écrits dans le langage C. Des exemples ont étés ajoutés afin d'aider la compréhension de la libraire.
Pour terminer, quelques exemples d'architectures sont livrés avec ce kit de développement ainsi qu'un exemple de serveur Web.

2.3 Caméra 2D

Cette partie est le résultat de deux projets de semestres successifs. Une carte a donc été développée. Le centre de celle-ci est évidemment le capteur d'images.

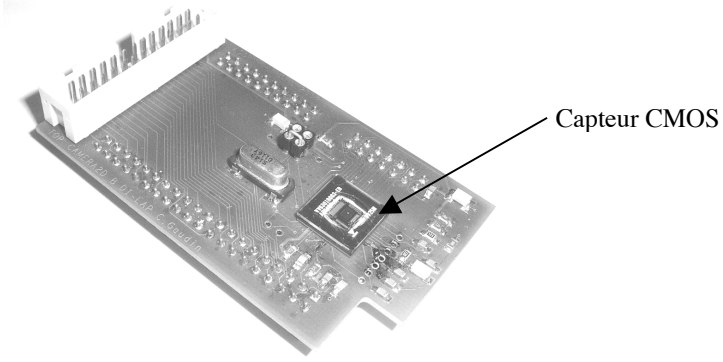

Figure 2.2 : Carte Caméra2D (sans optique)

Une grande partie du développement de ces projets fut évidemment l'interface avec le processeur Nios. En réalité, le problème a été découpé en deux. La première fraction s'occupe de la configuration de la caméra. Cette interface fonctionne correctement. La seconde fraction est responsable de l'acquisition des données provenant de la caméra. Cette interface ne fonctionne pas encore correctement, les images sont incompréhensibles.
Un logiciel permettant d'afficher les images a été développé. Il utilise le port RS-232, ceci donne un taux de rafraîchissements très lent. Ce programme méritera également quelques corrections.

2.4 Logiciels

Ce paragraphe ne décrit pas l'utilisation de chaque logiciel en détail. Il s'agit uniquement de donner une idée de leur utilité ainsi que de justifier leur utilisation.

2.4.1 Quartus

Ce logiciel s'occupe principalement de l'architecture de la logique programmable. A l'aide de ce logiciel, il est possible de réaliser tout d'abord le design de l'architecture. Il est ensuite possible de réaliser la synthèse, puis le placement et le routage pour enfin charger la configuration dans le circuit logique.

2.4.2 SOPC Builder

Ce logiciel est en principe fournit avec Quartus. La signification de SOPC est "System On Programmable Chip". Il est utilisé au travers de Quartus. Il sert à configurer des composants spéciaux. Dans ce travail, il sera utilisé pour configurer le processeur Nios ainsi que toutes les interfaces qui lui sont intégrées. Cet outil gère le plan mémoire, les interruptions. Il génère les fichiers de la librairie. Il génère le code de l'architecture complète en "vhdl". Il réalise également la synthèse afin d'optimiser le tout. Enfin, il permet de créer les fichiers permettant la simulation.

2.4.3 NIOS Shell

Ce logiciel permet de communiquer avec le processeur Nios au travers du port RS-232. Il est également utilisé pour la compilation et le linkage des fichiers sources. Il permet ensuite de charger ces fichiers binaires sur la carte. Enfin, il est utilisé pour l'exécution de ces derniers.

2.4.3 Ultra Edit

Il a été utilisé dans ce travail pour la réalisation des codes sources. Il permet d'écrire du code dans les langages C, vhdl et bien d'autres. Il gère les indentations, les couleurs, et d'autres. Ceci facilite énormément le travail et permet de se concentrer sur les problèmes de conception.

2.4.4 Visual C++

Cet outil est un éditeur. Il permet cependant de créer un projet complet et gère les différents fichiers. Il permet surtout de réaliser une interface visuelle très facilement. Il permet aussi d'affecter une action à un événement tel qu'un click de souris sur la case "Ok". Pour terminer, il crée un fichier exécutable. En résumé, il est facile de réaliser un logiciel dans un but précis en utilisant le langage C++ et les outils graphiques.

2.4.5 Logiciel CamAff

Ce logiciel a été réalisé à l'aide de Visual C++. Il permet d'afficher des images provenant de la caméra. Il est possible de les afficher en motif de Bayer ou en RGB. Les images originales sont en motif de Bayer. Une conversion a donc été réalisée afin de les convertir en RGB.

2.4.6 Sniffer XP

Cet un outil très important. Il permet d'écouter sur une connexion réseau et de capturer tous les paquets voulus. Il est possible de réalise un pré tri en fonction du protocole. De cette manière, il est par exemple possible de capturer uniquement les paquets utilisant le protocole TCP. Il est ensuite possible de lire les informations contenues à l'intérieures. Le logiciel s'occupe de décortiquer les informations de l'en-tête, comme par exemple, les adresses MAC de sources et de destinations, les adresses IP de sources et de destinations, la taille du paquet, etc.

Chapitre 3

Réseaux informatiques

3.1 Généralités

Afin de comprendre le fonctionnement du système développé dans ce projet, il est important de connaître quelques notions de base des réseaux informatiques.
Le développement des réseaux informatiques à commutation de paquets a commencé dans les années 60. Il est devenu la forme de réseau entre ordinateur la plus largement utilisée dans les années 90 grâce à "Internet". Il s'agit dans ce chapitre d'énoncé la série de protocoles concernés.

3.2 Division en couches

Les protocoles réseaux sont organisés en couches. Chacune d'elle prend en charge une partie des communications. Une série de protocole, comme par exemple, TCP/IP, est une combinaison de différents protocoles situés à des couches différentes. Les différentes couches sont les suivantes :

4	Application
3	Transport
2	Réseau
1	Lien

Figure 3.1 : Organisation en couches

Chacune des couches possède un rôle différent. Le détail de celles-ci sont expliquée dans les paragraphes suivants. Afin de bien comprendre la division en couche et l'utilité de celles-ci, prenons un exemple d'usage courant : L'organisation d'un voyage aérien.

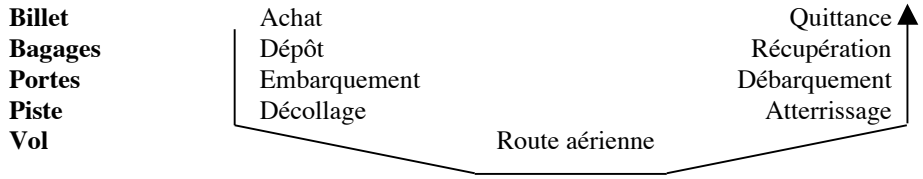

Figure 3.2 : Un voyage aérien

Avant de pouvoir prendre l'avion, il faut en principe acheter un billet. Il alors possible de déposer les bagages puis d'embarquer. Enfin l'avion décolle, il suit alors un certain chemin pour arriver à sa destination et enfin atterrit. A présent, il faut débarquer puis récupérer ses bagages. Enfin pour sortir, il faut quittancer son billet et passer à la douane.
Dans ce schéma, il est possible de voir les différentes couches que comporte un voyage aérien. Il est clair que les informations de départ sont importantes à l'arrivée, mais uniquement dans la couche concernée. Par exemple, le dépôt des bagages n'est utile à l'arrivée que pour la couche s'occupant des bagages, les autres couches ne sont pas concernées, le débarquement se fera peu importe l'organisation au niveau des bagages. Par contre, si l'embarquement n'est pas correct, le débarquement non plus.

Les protocoles réseaux fonctionnent de la même manière. Chaque couche est responsable d'une opération. Afin de la réaliser, la couche de niveau supérieur est encapsulée dans un paquet, une en-tête contenant des informations est alors ajoutée. Prenons le cas de notre projet. Les protocoles utilisés sont TCP/IP, évidemment ceux-ci sont utilisés aux travers d'un réseau Ethernet.

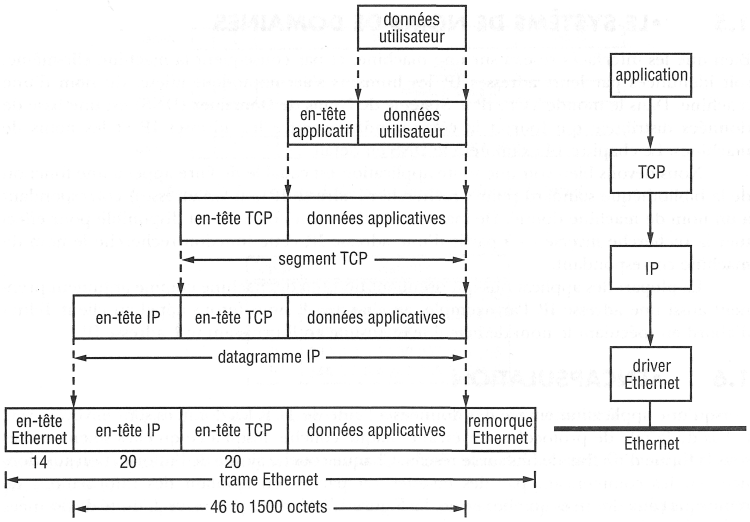

Figure 3.3 : Encapsulation des données des couches hautes vers les couches basses, source [6]

Maintenant chacun des paragraphes suivants explique la fonction et l'utilisation de la couche en question. Le protocole le plus important dans ce projet est alors mis en évidence.

3.3 Couche de lien : Ethernet

Cette couche est responsable de la transmission d'un paquet d'un poste à un autre. Ceci est cependant réalisé uniquement dans le réseau interne ou sous réseau. Afin d'identifier les différents clients présents sur le sou réseau, il existe les adresses physiques. Elles sont en général nommées "adresses MAC" et contiennent 48 bits. Ces adresses sont définies par le fabriquant de la carte. Elles sont uniques et sont fixes à cette partie matérielle.

Sur un réseau Ethernet tous les ordinateurs ou autres systèmes réseau sont connectés ensemble. Il faut donc pouvoir transmettre des paquets sans perturber les autres. Il existe plusieurs formes de multiplexage. Il existe par exemple le multiplexage temporel, chacun émet l'un après l'autre dans une tranche de temps qui lui est allouée. Il existe également le multiplexage fréquentiel, comme par exemples les transmissions radios, chacun émet dans sa bande de fréquences réservée. Ethernet utilise un procédé d'écoute. Si la ligne est libre, l'utilisateur peut envoyer des données. Il sait alors qu'il n'y a pas eu de collision, s'il ne reçoit pas de signal pendant un certain temps après le début de la communication. C'est un système aléatoire qui est donc basé sur l'écoute Il est en général nommé CSMA/CD qui signifie "Carrier Sense Multiple Access / Collision Detection".
Les paquets Ethernet envoyés sur la ligne ont la structure suivante :

Preamble	Adresse destination	Adresse source	Type	Données	CRC

Figure 3.4 : Paquet Ethernet

Lorsqu'un client envoi un paquet, il est émis sur la ligne et capturé par tous les systèmes présents sur le réseau. Seuls ceux concernés vont le garder, les autres systèmes vont l'éliminer. La sélection est réalisée grâce à l'adresse de destination, qui est l'adresse MAC. Si un paquet doit être envoyer à l'extérieur du sous réseau, la couche physique s'occupe uniquement de l'envoyer au routeur. Ensuite il sera envoyé de routeur en routeur, chaque fois la couche réseau sera utilisée, mais cette couche n'est pas responsable du chemin. La couche responsable de ceci est la couche réseau.

3.4 Couche réseau : IP

Cette couche est responsable de la transmission d'un paquet au travers d'un réseau. Elle s'occupe de choisir le routeur le plus approprié. Elle peut par exemple, renvoyer des paquets pour les rediriger plus optimalement.

Le protocole présenté signifie "Internet Protcol". Il utilise les adresses IP contenant 32 bits. Elles doivent être uniques dans le réseau et sont en général configurables au travers du système d'exploitation.
Les paquets sont envoyer au travers du réseau suivant un chemin. Ce chemin est déterminé au fur et à mesure grâce a des requêtes spécifiques permettant de connaître le meilleur routeur suivant. L'en-tête ajouté est un peu plus complexe.

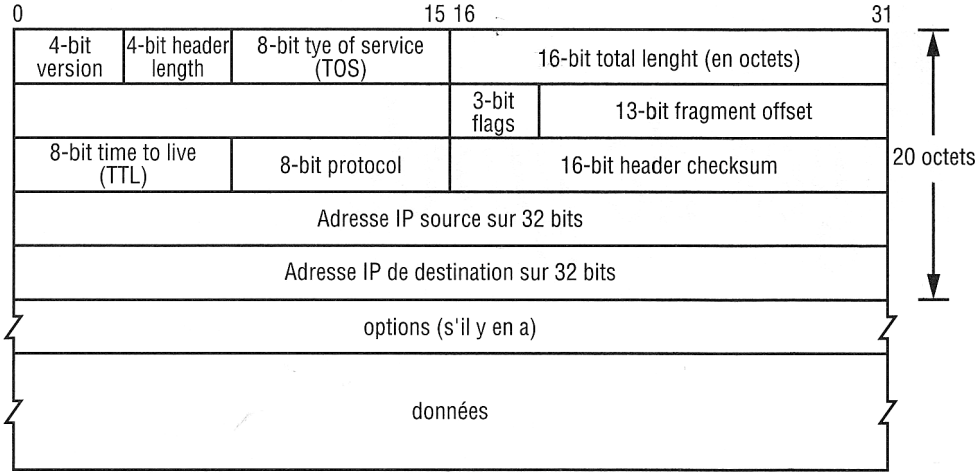

Figure 3.5 : Structure d'un paquet IP, source [6]

Il utilise donc une autre forme d'adresses qui sont les adresses IP. Il est important de savoir que les adresses IP ont une signification logique dans le sens où il est possible de localiser un système à l'aide de celle-ci. Ce qui n'est pas le cas des adresses physiques.

3.5 Couche de transport : TCP et UDP

Cette couche gère le flux entre deux utilisateurs. Elle peut par exemple mettre en œuvre des retransmissions dans le cas de pertes de données ou gérer des quittancements de paquets ou encore établir des temporisations pour attendre des quittances, etc. Elle gère également le démultiplexage interne à la machine. En effet, si plusieurs applications fonctionnent sur la même machine, jusqu'à maintenant il n'y avait aucun moyen d'identifier à quelle application appartient le paquet. Ce moyen nécessite l'introduction de la notion de "port". Un port identifie une application sur la machine en question. Le port identifie également le type de protocole de la couche application. La question est le port identifie l'application ou le protocole ?
Prenons exemple : un ordinateur dont deux navigateurs tournent. Chacun d'eux fait une requête au serveur http. Ils utilisent donc leur propre port pour la source, mais tout les deux auront un port de destination de 80(http). Le serveur leur répond avec comme port source 80(http), mais la destination identifiera le navigateur en question.
Les ports de 1 à 1023 sont réservés pour les protocoles courants. Le reste est libre. Quelques exemples :

Port	Application
80	http
21	ftp
25	smtp

Il existe principalement deux protocoles, TCP et UDP. Ils ont chacun leurs avantages et leurs inconvénients.

Commençons par UDP. Ce protocole est simple. Il ne nécessite pas de connexion et ne garantit pas la transmission des paquets. Il n'y a pas de contrôle de flux ni de contrôle de congestion. En résumé, ce protocole envoi des paquets sans se soucier de ce qu'ils deviennent. L'en-tête ajouté à la structure est le suivant.

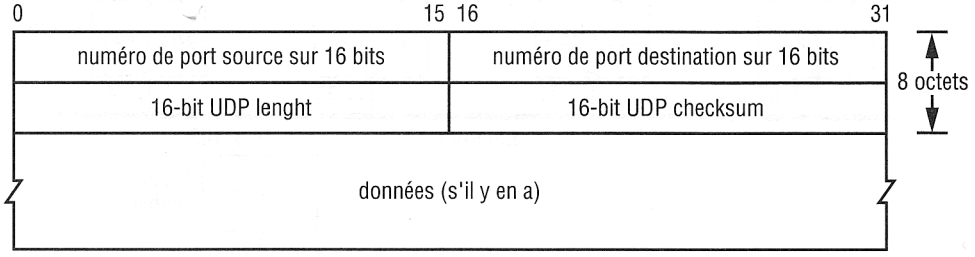

Figure 3.6 : Structure d'un paquet UDP, source [6]

La constatation à faire est que les seules informations sont les numéros des ports de source et de destination, la taille du paquet et le checksum permettant de savoir si le paquet est sans erreurs.

Au contraire TCP est un protocole relativement complexe. Il nécessite une connexion, c'est-à-dire qu'avant d'envoyer des informations, une connexion est ouverte et à la fin la connexion est refermée. De cette manière, chacun des host garde des informations sur l'autre. Ce protocole est fiable, il garantit l'arrivée à destination des paquets. Il garantit également qu'il y aura ni pertes ni erreurs. De plus, il garantit que tous les paquets seront dans l'ordre. Grâce à sa connexion, ce protocole permet de gérer le flux et finalement il est capable de réaliser un contrôle de congestion.

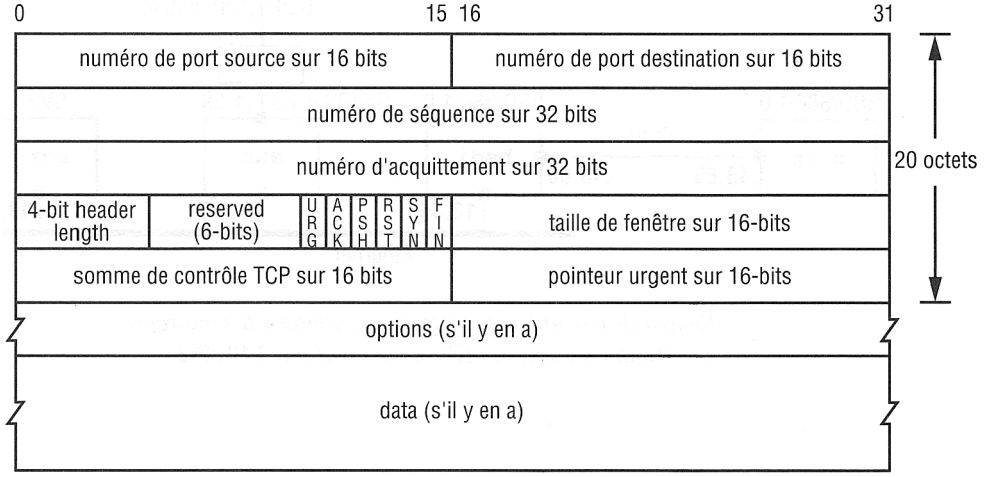

Figure 3.7 : Structure d'un paquet TCP, source [6]

Le choix entre TCP et UDP se fait selon l'application. En principe, si un service fiable est nécessaire, le choix sera TCP. Si par contre, un service rapide est nécessaire, le choix se fera sur UDP.
Voici quelques exemples :

Application	Protocole couche application	Protocole couche transport
e-mail	smtp	TCP
Accès distant à un terminal	telnet	TCP
Web	http	TCP
Transfert de fichiers	FTP	TCP
Multimedia		TCP ou UDP
Telephonie sur internet		UDP

Figure 3.8 : Applications sur TCP ou UDP

Par exemple, dans le cas de la téléphonie, quelques pertes ne sont pas importantes, par contre la rapidité est primordiale.

3.6 Couche d'application : HTTP

Enfin, cette dernière couche est utilisée pour transmettre les informations réellement utiles à l'utilisateur. Il existe par exemple SMTP, utilisé pour le transfert d'emails, ou FTP, pour le téléchargement de fichier, ainsi que bien d'autres encore.

Le protocole http est en fait le cœur de ce que l'on nomme généralement "Internet". En effet, le Web fonctionne à l'aide de ce protocole ainsi que de ceux cités ci-dessus. Il très important de comprendre le fonctionnement de ce protocole.
Le protocole http est toujours implémenté en deux parties : le client et le serveur. En plus, il peut être lu de tout système d'exploitation. Ce protocole utilise TCP et est "stateless", c'est-à-dire que le serveur ne garde aucune information sur le client et vice-versa. Ce n'est par contre pas le cas des couches au-dessous. Voici le déroulement des différentes étapes durant une connexion.

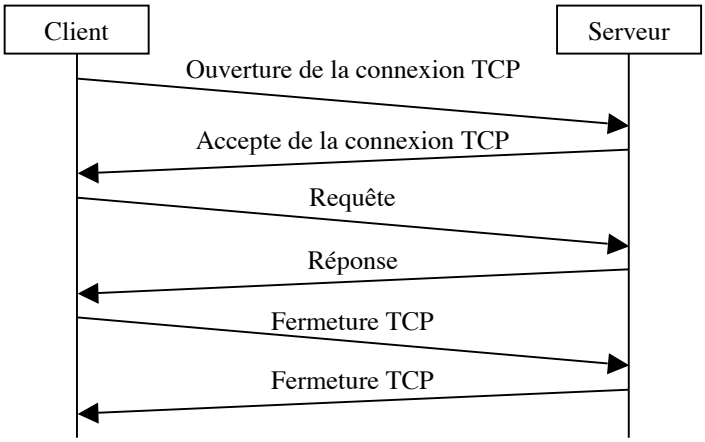

Figure 3.9 : Connexion http

Tout d'abord, une connexion TCP doit être ouverte. Une fois ceci réalisé, le client envoi une requête http. Le serveur lui répond par une réponse http. Lorsque le client a reçu le paquet, il ferme la connexion TCP.
En principe, la réponse du serveur est une page html. Dans le cas où celle-ci contiendrait des références, comme par exemple des images, le client réalise les étapes ci-dessus pour chacune des références. Exemple d'une page html avec 3 images JPEG.

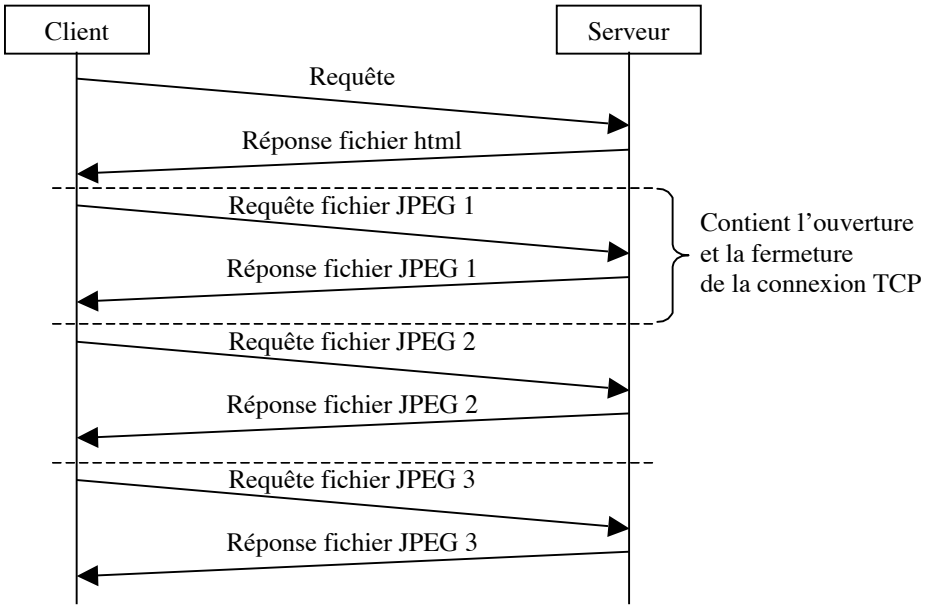

Figure 3.10 : Exemple requête d'une page html et de ses références

Chapitre 4

Architecture générale du système

4.1 Schéma bloc

Il est préférable de visualiser l'ensemble de l'architecture avant d'entrer dans les détails du projet. Le cœur de la carte mère est évidemment la logique programmable. Ce chip est, tout comme la carte de développement, produit par Altera. Il contiendra le processeur Nios, la mémoire ROM ainsi que toutes les interfaces permettant au processeur le dialogue avec le monde extérieur. Parmi celles-ci les deux plus importantes sont les interfaces de communication avec la caméra et le module Ethernet. Ces deux modules externes sont décrits plus en détails dans les chapitres suivants. Il ne faut également pas oublier la mémoire située à l'extérieure de la logique programmable. Il y a une grande quantité de mémoire Flash ainsi qu'un peu de mémoire RAM.

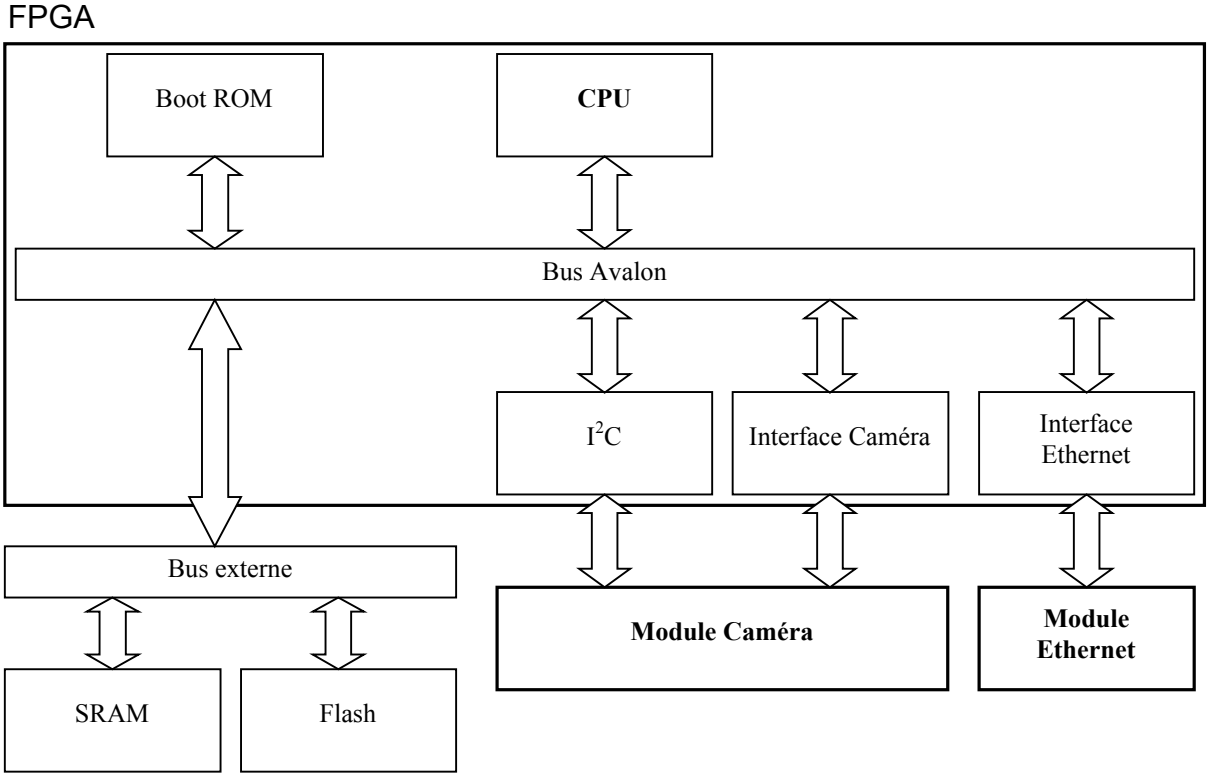

Figure 4.1 : Schéma bloc global

Ce schéma permet entre autre de bien distinguer la caméra et le module Ethernet qui sont en réalité deux cartes filles distinctes. La séparation des interfaces de configuration et de données de la caméra est mise en évidence.

Afin de bien comprendre l'assemblage des différents éléments, une photo permet de les mettre en évidence. De plus, la compréhension des différents blocs est facilitée.

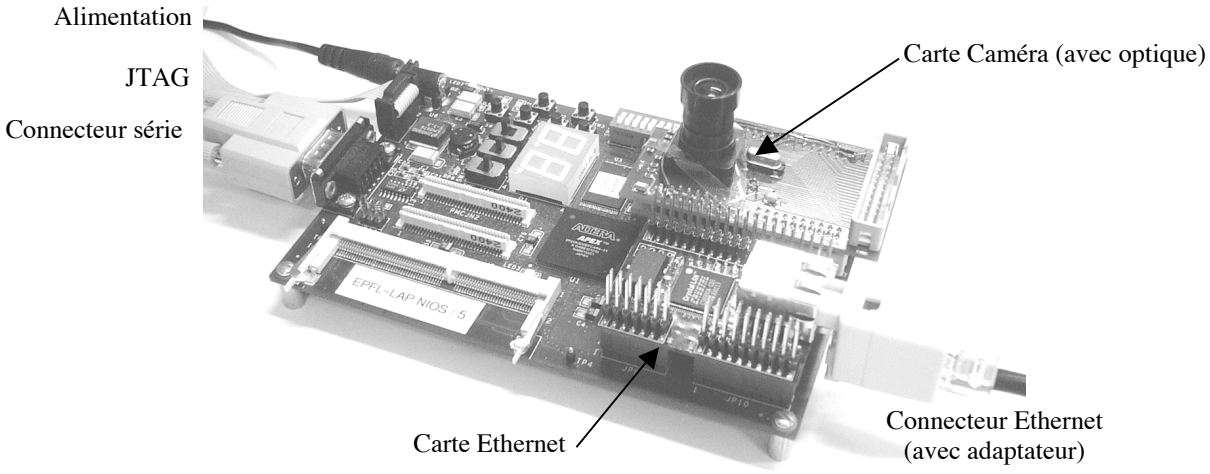

Figure 4.2 : Photo de la carte prête à l'emploi

4.2 Les différents composants en détails

4.2.1 Module sous SoPc Builder

Voici l'interface gérant les différents éléments devant être accessibles par le processeur Nios (en surligné).

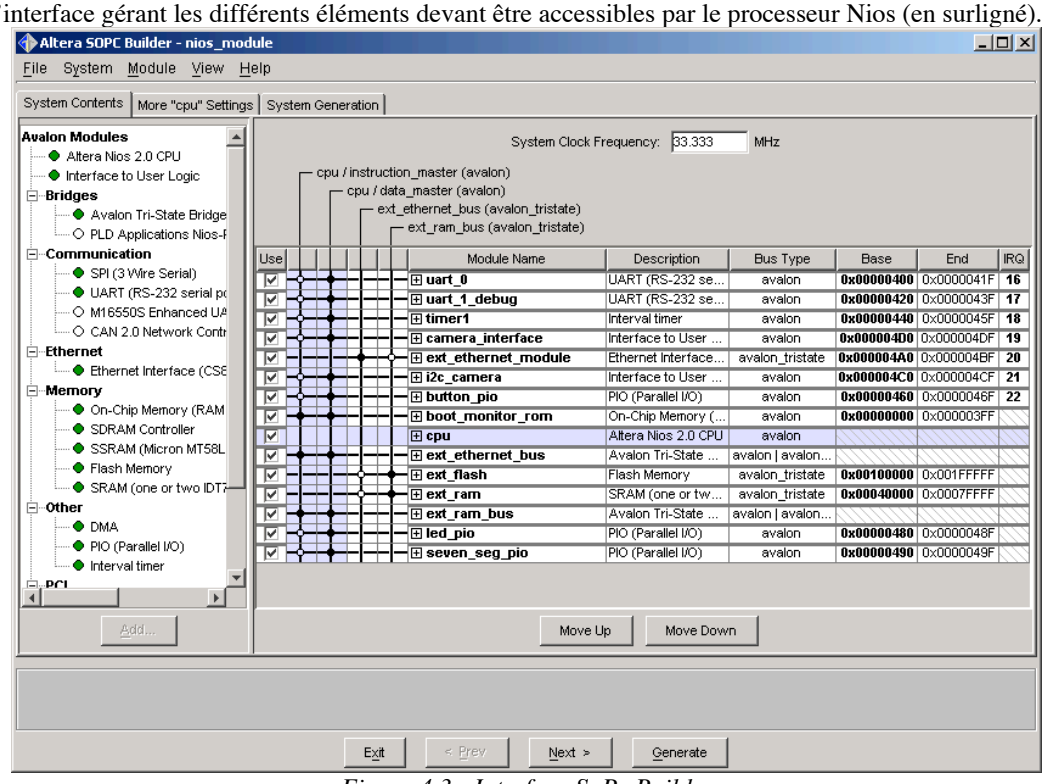

Figure 4.3 : Interface SoPc Builder

Les adresses de début et de fin sont automatiquement gérées. Les niveaux d'interruptions également. Il est possible d'apercevoir d'autres bus que le bus Avalon. Ces bus sont utilisés pour communiquer avec l'extérieure de la FPGA. Ils sont donc physiquement différents, mais en réalité ils sont utilisés de manières similaires.

Les quelques paragraphes suivants décrivent les différents paramètres nécessaires à la configuration des éléments. Seuls les parties concernant l'interfaçage avec le module Ethernet et le module Camera ne sont pas expliqué ici. Ils feront l'objet d'un chapitre chacun.

4.2.2 Processeur Nios

La création d'un processeur Nios est simple. Il suffit de remplir quelques paramètres. Ce processeur existe sous deux versions : une de 16 bits et une de 32 bits. Dans ce projet, la version choisie est celle de 32 bits. La raison est simple. La FPGA à disposition est énorme comparée à la capacité nécessaire. Il n'est donc pas nécessaire d'économiser de la place, alors pourquoi se priver d'un processeur de 32 bits ?

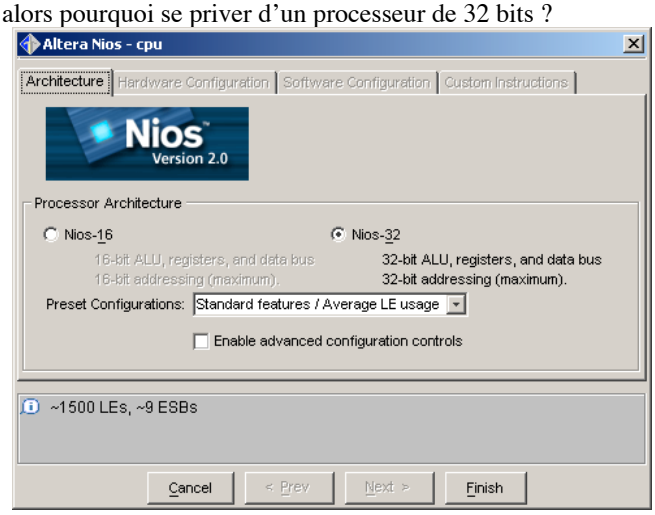

Figure 4.4 : Importation d'un processeur Nios

4.2.3 La mémoire de boot (ROM)

Cette mémoire est très importante. Elle s'occupe de configurer la logique programmable lors de la mise sous tension ou lors d'un reset matériel. La largeur choisie est de 32 bits et la capacité de 1Kbytes. Le Germs y est écrit afin d'avoir un petit programme de test lors du démarrage de la carte.

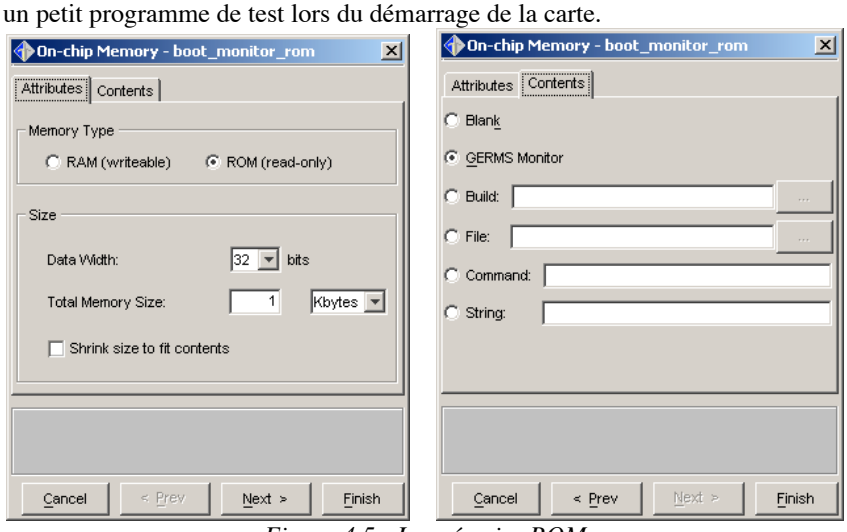

Figure 4.5 : La mémoire ROM

4.2.4 UART

En réalité deux uart sont intégrées à l'architecture. Elles permettent au processeur Nios de dialoguer avec le Nios Shell en passant au travers du port RS-232.

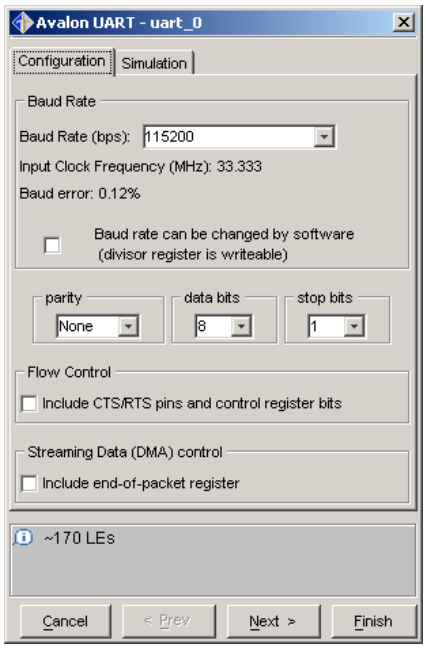

Figure 4.6 : Les UART's

Cette unité sera également utilisée dans un premier temps pour transmettre les images au logiciel CamAff. Il est plus facile de cette manière de trouver les erreurs sans avoir à s'occuper du module Ethernet.

4.2.5 Timer

Un timer permet au processeur de pouvoir mesurer le temps. Il est possible de générer des interruptions. Plusieurs registres permettent de le configurer de la manière voulue.

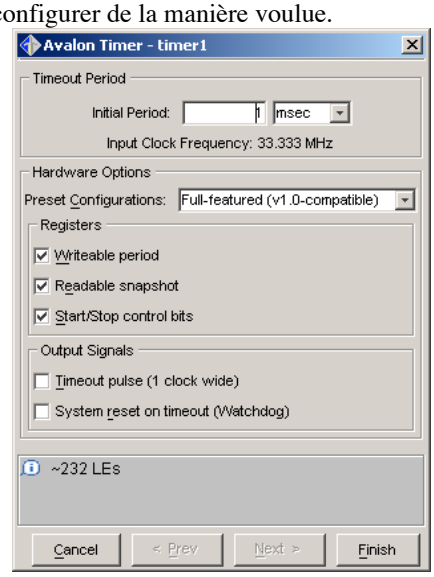

Figure 4.7 : Le timer

4.2.6 Boutons

Les boutons permettent à l'utilisateur de pouvoir créer des événements en entrée. Ils sont au nombre de 12, c'est pourquoi la largeur est de 12 bits.

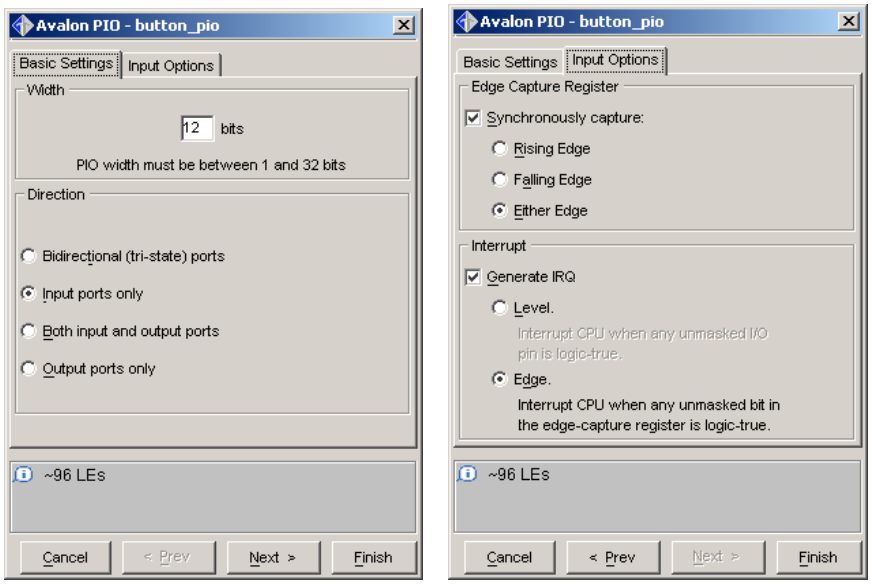

Figure 4.8 : Les boutons

4.2.7 Les LED's

Elles donnent la possibilité au processeur d'émettre un signal à l'utilisateur. Elles peuvent par exemple être utilisée afin de réaliser des mesures. Il est possible d'allumer, puis d'éteindre une LED lors d'événements voulus. A l'aide d'un oscilloscope une mesure temporelle peut être effectuée sur la pin de celle-ci.

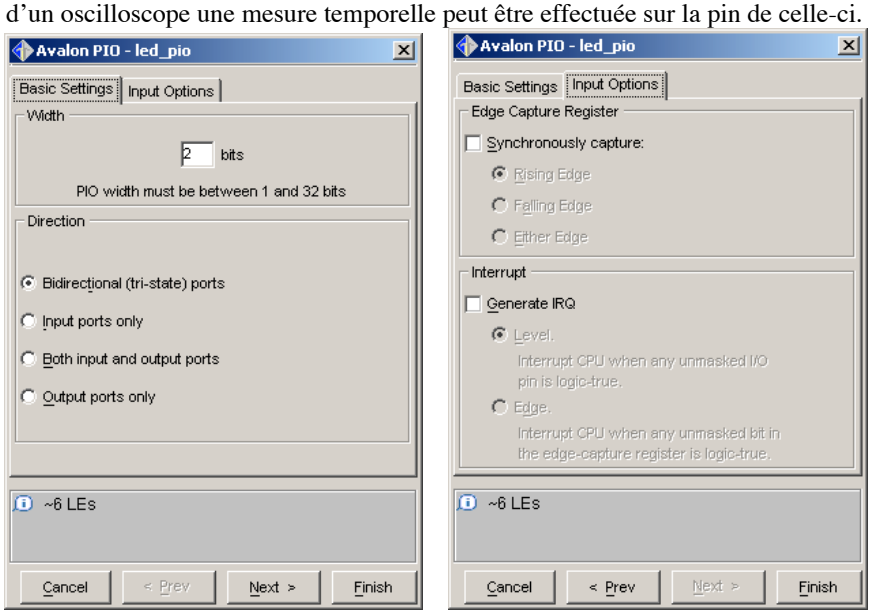

Figure 4.9 : Les deux Led's

4.2.8 Affichage 7 segments

L'affichage 7 segments est utilisé par le processeur pour pouvoir indiquer à l'utilisateur un nombre qui peut être sur deux digits. Il peut également être utilisé simplement comme un ensemble de LED's.

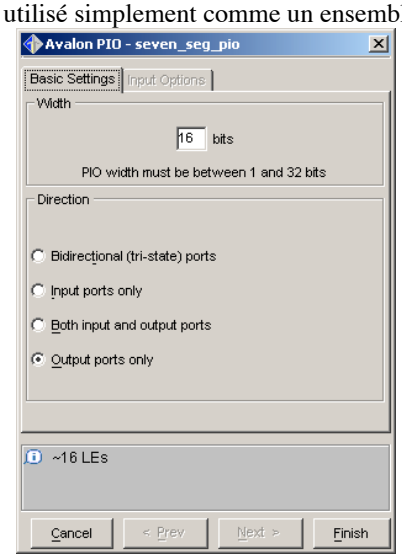

Figure 4.10 : L'affichage 7 segments à 2 digits

4.2.9 La bus de mémoire externe

Un bus a du être ajouté à l'intérieur de la logique programmable. Il permet la liaison du bus Avalon avec les pins externes voulues.

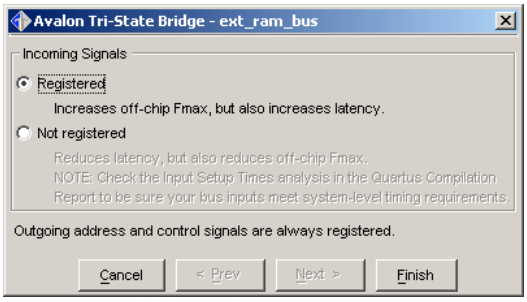

Figure 4.11 : La bus de mémoire externe

4.2.10 La mémoire statique (SRAM)

La mémoire statique est utilisée par le processeur comme mémoire vive. Elle contient les instructions nécessaires pour le déroulement du programme ainsi que toutes les variables stockées en mémoires.

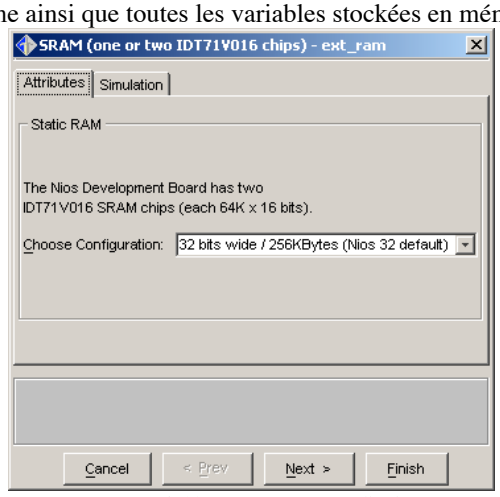

Figure 4.12 : La mémoire SRAM

4.2.11 La mémoire flash

Enfin, la mémoire flash permet l'enregistrement de données de façon "fixe". C'est-à-dire que même lors d'une coupure de courant, le contenu reste intacte.

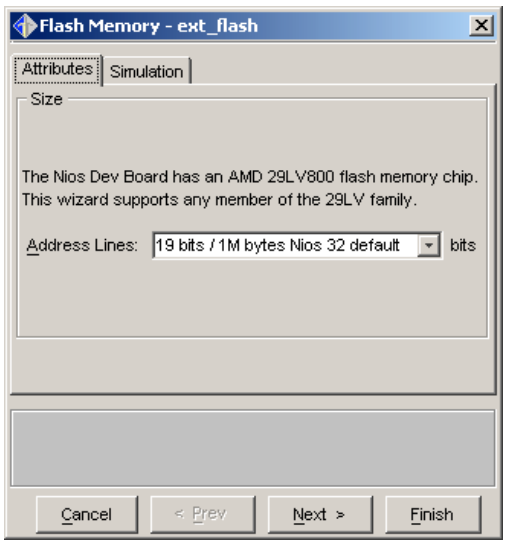

Figure 4.13 : La mémoire Flash

4.3 Schéma logique

En entrant un plus dans les détails, il est possible de voir sur le schéma ci-dessous des informations supplémentaires. Par exemple, la taille des bus utilisés pour chaque périphérique.

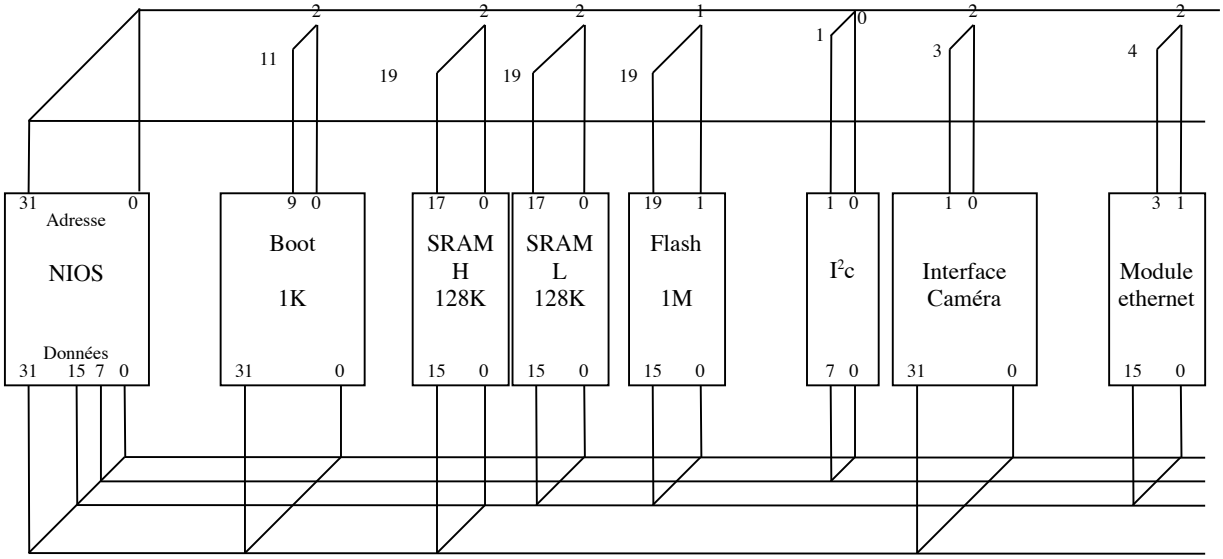

Figure 4.14 : Schéma logique global

La connexion du module Ethernet peut paraître inexacte. Elle sera expliquée au chapitre 5 en détails et est correcte.
La largeur des différents périphériques est variable. Cet aspect ne dérange pas le processeur, car le bus Avalon est un bus dynamique. Dans le cas d'accès inférieurs à 32 bits, le processeur détecte qu'une partie des données n'est pas présente. Ceci est réalisé à l'aide des signaux de quittance. Il effectue alors un cycle supplémentaires selon si le périphérique à une largeur de 16 bits. Dans le cas où la largeur est de 8 bits, le processeur doit effectuer quatre cycles au total pour une lecture de 32 bits.

4.4 Plan mémoire et interruptions

Lors du développement, d'une application, il est toujours utile de concevoir préalablement un plan mémoire. Dans le cadre de ce projet, le logiciel Quartus a été utilisé pour créer l'architecture. Il gère lui-même la mémoire ainsi que le niveau des requêtes d'interruptions. Le plan a donc été conservé tel quel.

Description	**Départ**	**Fin**	**Taille$_{CPU32}$[Bytes]**	**IRQ**
Boot ROM	0x0	0x3FF	1 K	
UART_0	0x400	0x41F	32	16
UART_1	0x420	0x43F	32	17
Timer	0x440	0x45F	32	18
Button_pio	0x460	0x46F	16	19
LED_pio	0x480	0x48F	16	
7 segments_pio	0x490	0x49F	16	
Module Ethernet	0x4A0	0x4BF	32	20
Interface caméra de configuration	0x4C0	0x4CF	16	21
Interface caméra de donnée	0x4D0	0x4DF	16	22
SRAM (2 chips)	0x40000	0x7FFFF	256 K	
Flash	0x100000	0x1FFFFF	1 M	

Figure 4.15 : Plan mémoire

Il faut noter que la SRAM est répartie en deux chips. Chacun a une largeur de 16 bits. Ils sont connectés de manière à former une zone mémoire sur une largeur de 32 bits.

Adresse départ	**31 24**	**23 16**	**15 8**	**7 0**
0x0	Boot ROM			
0x400	Différents périphériques (uart, timer, …)			
0x4A0			Module Ethernet	
0x4C0				I^2C
0x4D0	Caméra interface			
0x40000	SRAM 1		SRAM 2	
0x100000	Flash			

Figure 4.16 : Graphique des zones mémoire

Chapitre 5

Interface avec la caméra

5.1 Remarques préalables

La totalité de l'interface avec la caméra a été réalisée durant un projet précédent. Un logiciel permettant d'afficher les images a également été développé. Il se charge de communiquer avec le processeur afin d'acquérir des données. Les informations passent au travers du port RS-232, ce qui impose un débit relativement lent. Grâce à ce logiciel, il est possible d'obtenir des images, mais également les registres de contrôle et d'état.
A ce stade d'avancement, les images affichées sont encore incompréhensibles. Il reste à déterminer d'où proviennent les erreurs.

L'interface est découpée en deux parties. La première est l'interface de configuration. Elle permet d'accéder aux registres internes à la caméra, tel que les registres de contrôles, d'états ainsi que de configuration. Elle fonctionne sur le principe de l'I^2C et ne contient par conséquent que deux fils. Il est possible de décuire que cette interface fonctionne correctement. En effet, après une écriture, la lecture donne une bonne confirmation. De plus, les registres d'états sont cohérents.

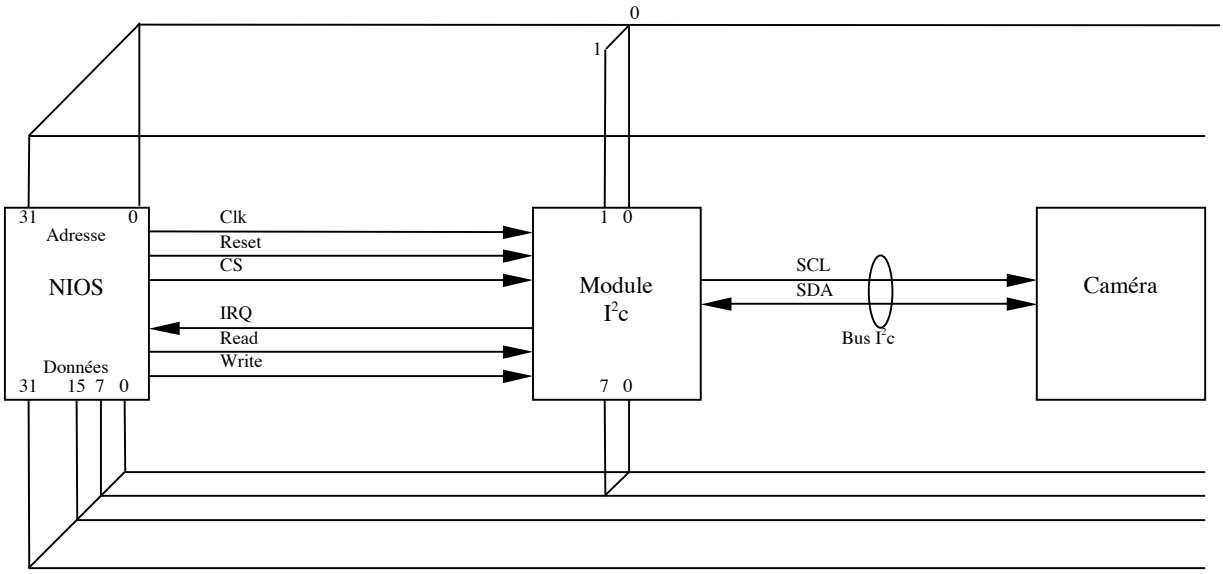

Figure 5.1 : Schéma logique de l'interface de configuration

La seconde interface est responsable de l'acquisition des données. Afin de tester cette partie indépendamment du programme d'affichage, on affiche directement les données sous forme de caractères dans le terminal du Nios. Ici, des phénomènes incompréhensibles sont visibles. Tout d'abord, la caméra est soit en surexposition, soit en sous-exposition. Ceci peut éventuellement provenir d'un mauvais réglage de la lentille. Le second phénomène visible est que le nombre de pixels capturés pour une image semble aléatoire d'une acquisition à l'autre.

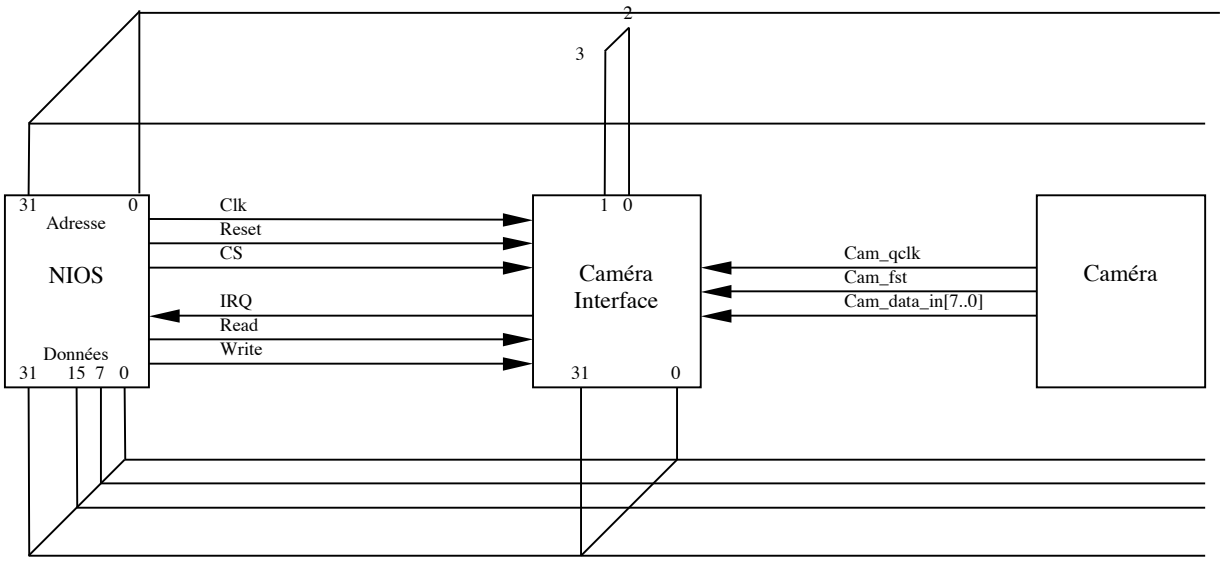

Figure 5.2 : Schéma logique de l'interface de données

Les erreurs proviennent certainement de l'interface de données. Evidemment, le logiciel d'affichage n'est pas exclu, il peut tout à fait contenir des erreurs supplémentaires.

5.2 Synchronisation sur les données de la caméra

L'interface de donnée reçoit 2 signaux et un bus. Ce dernier est évidemment le bus de données. Il contient 8 bits. Les deux autres signaux servent à ce synchroniser sur les images. Il y a évidemment l'horloge. En plus, un signal permet de connaître le début de la trame de données.
Plusieurs configurations sont possibles. Elles sont déterminées en écrivant dans les registres de la caméra au travers de l'interface de configuration. Le mode "Control plus Data" a été choisi. Il a plusieurs avantages :
Deux lignes noires sont ajoutées en début de trame. En réalité, il y a une série de lignes supplémentaires mais elles sont masquées. Au total, la trame a une taille de 147 lignes. 126 sont transmises, mais uniquement les 124 dernières contiennent l'image.
Le signal de début de trame indique suffisamment tôt l'arrivée des données. Le signal d'horloge est actif uniquement s'il y a des données à lire. C'est-à-dire que l'horloge est inactive lorsqu'il y a des lignes masquées, ce qui n'est pas le cas des autres modes.

Ligne n°	0	1 2	3 10	11 12 133 134	135 144	145	146
Descritpion	Start	2 lignes noires	Lignes masquées	Lignes visibles	Lignes masquées	Stop	Ligne masquée

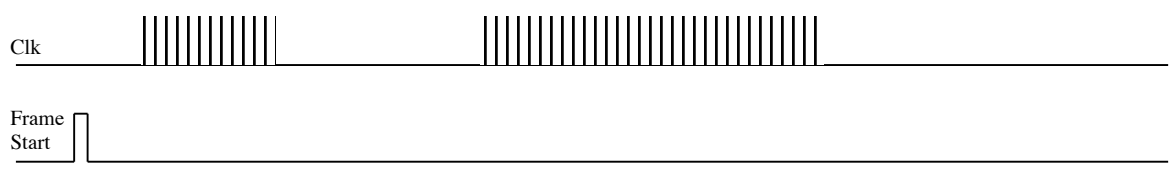

Figure 5.3 : Signaux de synchronisations

Avec ce mode de fonctionnement, il est facile d'extraire l'image. Il suffit d'attendre le début de la trame. Ensuite à chaque flanc descendant de l'horloge, on capture les données.

5.3 Nouvelle structure

La structure qui a été développée est basée sur un FIFO. Voici le schéma bloc de l'interface de données :

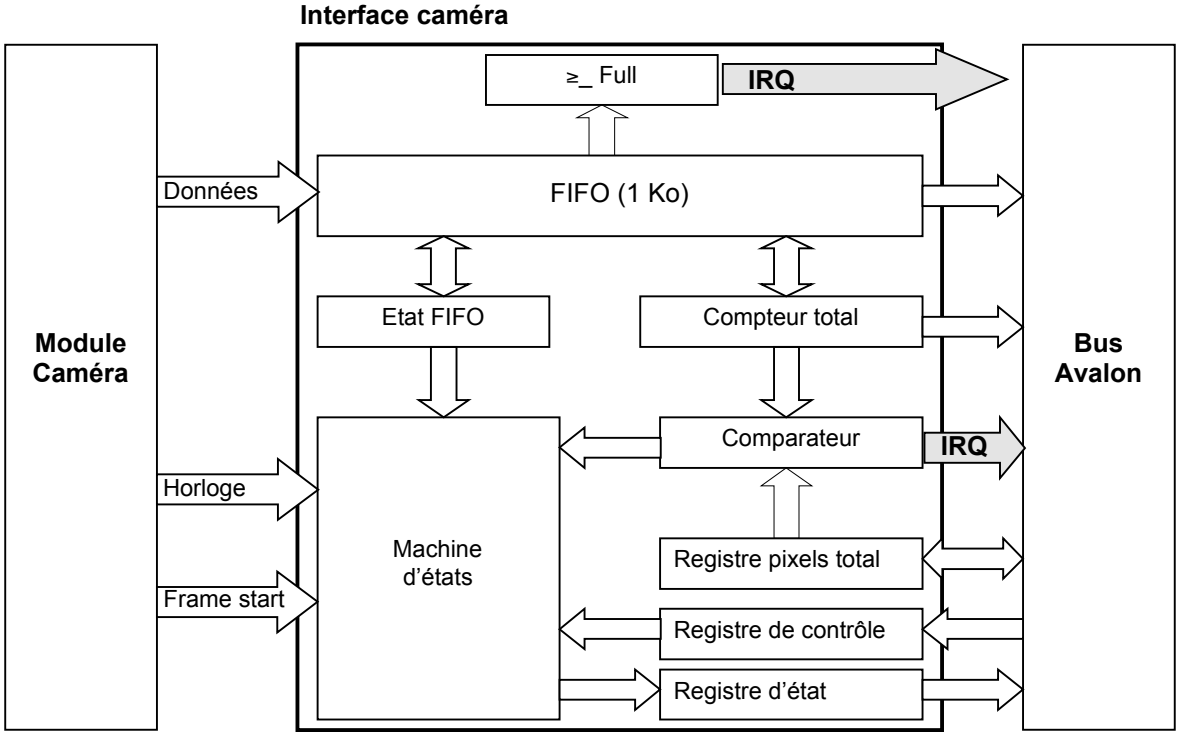

Figure 5.4 : Schéma bloc de l'interface de données de la caméra

On a tout d'abord trois registres qui permettent le dialogue entre le processeur et l'interface de données de la caméra. Le registre de contrôle permet de commander l'interface. Le registre d'état permet de connaître diverses informations sur la situation actuelle de l'interface. Enfin, le registre de pixels total indique à l'interface la taille de l'image à acquérir.
Les données sont misent en mémoire dans le FIFO. Le registre "état FIFO" permet de connaître le nombre de mots présents à l'intérieur. Le registre "Compteur total" permet de connaître combien de mots ont étés capturés durant toute l'acquisition.
Il faut évidemment informer le processeur lorsqu'il doit vider le FIFO. Ces avertissements se font à l'aide d'interruptions. Le premier type d'interruptions est lorsque le FIFO est à moitié plein. Pratiquement, il suffit de tester le bit de poids fort du registre "Etat FIFO". Ceci laisse suffisamment de temps au processeur pour éviter un overflow. Evidemment une interruption doit être levée lorsque l'acquisition est terminée. De cette manière le processeur vient également vider le FIFO et l'image est complètement acquise.
Il ne reste qu'à étudier la machine d'états.

- L'interface se trouve dans l'état "oisif".
- Lorsque le signal de départ est activé la machine d'état passe dans un état d'attente de début de trame. Ceci permet de se synchroniser sur la trame. Cette étape peut-être évitée en le spécifiant dans le registre de contrôle à l'aide du fanion "wait_fst".
- Lorsque le début de trame est détecté, il faut passer à l'état d'attente d'un flanc descendant de l'horloge. Ceci permet de se synchroniser sur l'horloge.
- Le flanc détecté, on passe dans un état de lecture et les données sont écrites dans le FIFO.
- a) Si la limite n'est pas atteinte, on retourne se synchroniser sur l'horloge.
  b) Dans le cas contraire, c'est-à-dire que l'image est totalement acquise, il faut alors retourner à l'état "oisif".

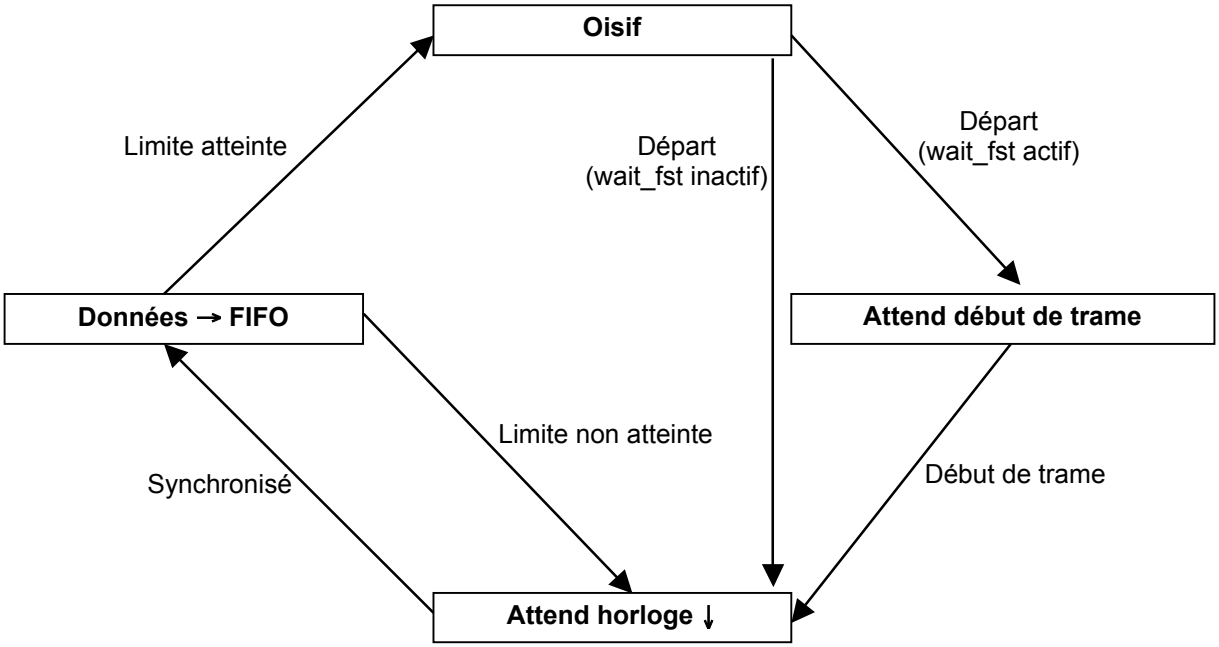

Figure 5.5 : Machine d'états

5.4 Horloge métastable

Les pixels sont capturés en utilisant l'horloge du processeur. Il faut donc échantillonné l'horloge de la caméra en se synchronisant sur l'horloge du processeur. Ce procédé conduit à des signaux métastables.
Pour éliminer ce phénomène, le signal passe au travers de 4 bascules. Celles-ci créent un retard entre l'entrée et la sortie. Elles permettent une plus grande stabilité lors de l'échatillonnage.

5.5 Intégration des interfaces dans la logique programmable

Afin que le processeur Nios puisse reconnaître les deux interfaces, il faut les intégrer dans le module Nios. Ceci est réalisé à l'aide du logiciel SoPc Builder au travers de Quartus.
Il faut sélectionner "Interface to User Logic" dans le volet de gauche puis cliquer sur le bouton "Add". A ce moment, une fenêtre contenant deux onglets s'ouvre :

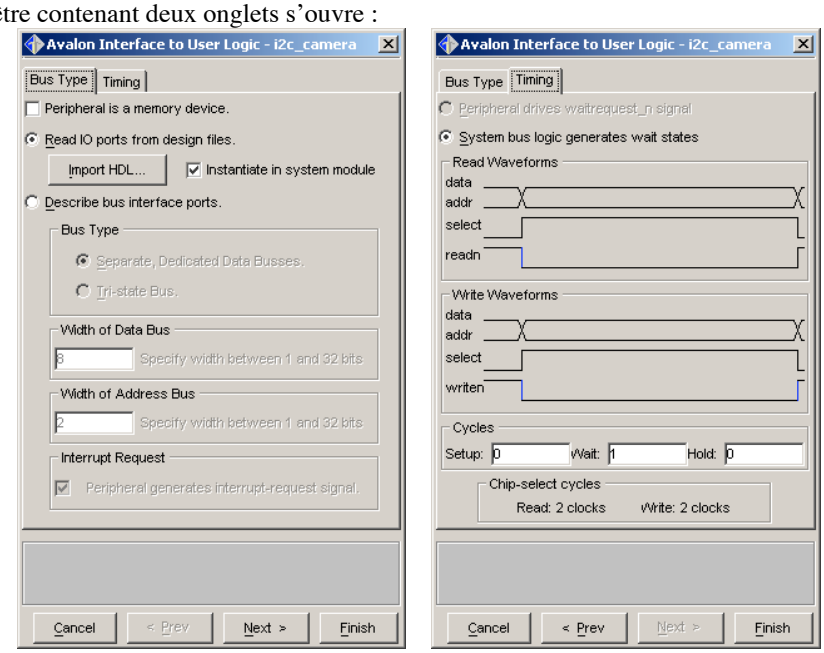

Figure 5.6 : Installation des l'interfaces dans SoPc

Dans le volet "Bus type", il faut sélectionner "Read IO ports from design files". Il faut alors cocher la case "Instantiate in system module" puis cliquer sur le bouton "import HDL". A ce moment, la fenêtre suivante s'ouvre :

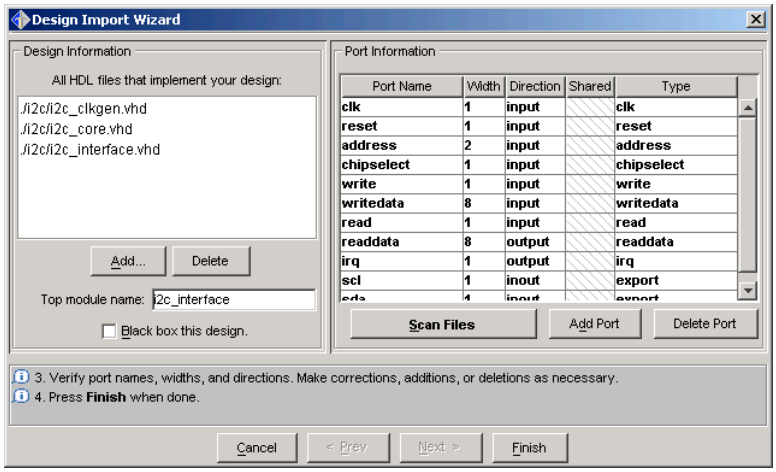

Figure 5.7 : Installation de l'interface de configuration

Il faut alors sélectionner les 3 fichiers voulus, puis cliquer sur le bouton "scan files". Il reste alors qu'à définir le type pour chaquun des signaux trouvés.
Finalement, il suffit de cliquer sur le bouton "finish" et les fichiers du module de configuration sont installés.

Une dernière étape consiste à imposer les temps lors des accès. Ceci est réalisé dans le volet "timing". Il faut mettre un coup d'horloge d'attente pour chaque cycle. Voir figure 5.6.

En ce qui concerne l'interface de données, il faut réaliser les mêmes démarches mais en changeant les fichiers à importer. Evidemment, il faut sélectionner le fichier décrivant cette interface.

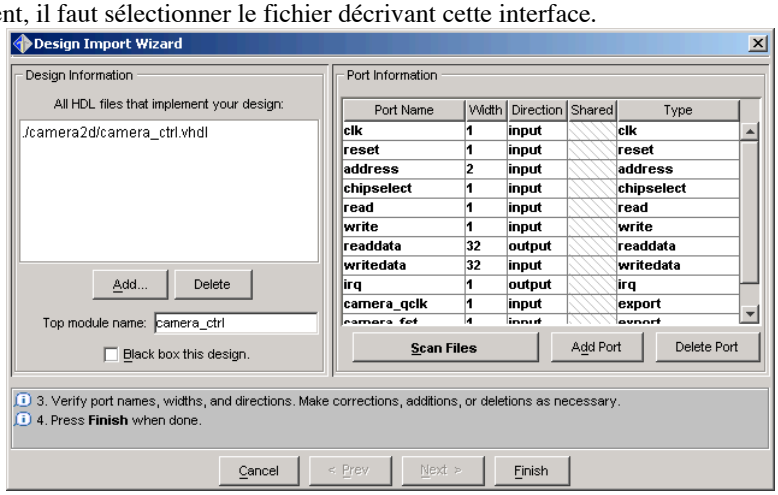

Figure 5.8 : Installation de l'interface de données

5.6 Registres de l'interface

Comme expliqué précédemment, il existe trois registres. Les trois registres ont une taille de 32 bits. La plupart des bits ne sont cependant pas utilisés. Cette taille est surtout utilisée pour permettre des cycles d'accès depuis le processeur sur 32 bits.

5.6.1 Registre de contrôle

Le premier registre est évidemment le registre de contrôle. Il permet de donner des ordres à l'interface de données de la caméra.

Unused	31 - 4	3	2	1	0
Fonction	Unused	fifo_clear	Irq_enable	Wait_fst	start

Figure 5.9 : Registre de contrôle

Le bit “start” permet d’indiquer à l’interface qu’elle doit débuter l’acquisition.
Le bit “wait_fst” indique que lors d’une acquisition, il faut attendre le signal de synchronisation de la trame.
Le bit “irq_enable” autorise les interruptions.
Le bit “fifo_clear” ordonne à l’interface de vider le fifo.

5.6.2 Registre d’état

Le second registre permet de connaître la situation de l’interface de donnée de la caméra.

Unused	31-18	17 - 8	7-6	5	4	3	2	1	0
Fonction	-	Fifo_usedw	-	Fifo_empty	Data_counter_flag	Fifo_overrun	Fifo_flag	Intr_pending	running

Figure 5.10 : Registre d’état

Le bit “running” indique que l’interface est en train de réaliser une acquisition.
Le bit “intr_pending” indique qu’une interruption n’a pas été traitée. Une écriture sur ce registre permet d’effacer l’interruption.
Le bit “fifo_flag” indique que le fifo est plein de plus de la moitié de sa capacité. Il faut donc le vider.
Le bit “fifo_overrun” permet de savoir s’il y eu un overflow et donc une perte de données.
Le bit “data_counter_flag” indique si l’acquisition est arrivée à terme.
Le bit “fifo_empty” permet de savoir si le fifo est vide.
Les bits “fifo_usedw” donnent exactement le nombre de mots présents dans le fifo.

5.6.3 Registre des compteurs

Le troisième registre permet d’indiquer à la caméra le nombre de pixels à acquérir. De cette manière, n’importe quelle résolution peut être utilisée par cette interface.

N° bit	31 - 16	15 - 0
Fonction	Data_counter	Data_counter_limit

Figure 5.11 : Registre de limite et de compteur

Les bits “ Data_counter_limit ”permettent au processeur d’indiquer à l’interface le nombre de pixels à acquérir. En principe cette valeur représente la taille de l’image en nombre de pixels.
Les bits “ Data_counter” permettent de savoir combien de pixels ont déjà été capturés durant cette acquisition.

5.6.4 Registre de données

Le dernier registre permet au processeur de lire les données.

N° bit	31 – 8	7 – 0
Fonction	Unused	Fifo_data

Figure 5.12 : Registre de données

Les bits “ Fifo_data” sont simplement les données extraites du fifo. Cette lecture correspond donc au niveau d’intensité de la couleur du pixel en question. Chaque lecture efface automatiquement du fifo le mot dont l’accès a été effectué.

5.7 Utilisation au travers du processeur

Cette interface est très facilement utilisable au travers du processeur. Il faut écrire deux morceaux de code, l’un étant le programme principal et l’autre étant le traitement des interruptions.
Il est préférable de commencer par définir la routine d’interruptions. On se souvient que deux types d’interruptions peuvent être levées : vider le FIFO avant l’overflow et vider le FIFO dans le cas où l’acquisition est terminée. A chaque interruption, il suffit donc de vider le FIFO. Ceci est réalisé simplement par une boucle qui tourne tant que le FIFO n’est pas vide. Lorsque la limite est atteinte, c’est-à-dire que l’acquisition est terminée, il faut lever un flag permettant au programme principal de savoir que l’acquisition est terminée.
Le programme principal commence tout d’abord par initialiser la caméra avec la configuration voulue. Il va configurer les interruptions et enfin initialiser l’interface de données. Ensuite, il lance l’acquisition d’une image. Il suffit ensuite d’attendre l’activation du flag indiquant que l’acquisition est terminée pour savoir que l’image est entièrement en mémoire.

5.8 Limite du système

Afin de connaître exactement le temps mis par le processeur pour vider le fifo, une mesure a été réalisée. A chaque entrée dans la routine d'interruptions, un signal est activé. Evidemment, à la sortie de la routine, le signal est désactivé. Pour des raisons de simplicité, le signal choisit est la sortie sur une led. Ceci permet d'y brancher facilement un oscilloscope. De plus, un affichage permet de savoir combien de mots ont été lus pour chaque réalisation de la routine.
Le diagramme ci-dessous représente l'acquisition d'une image complète.

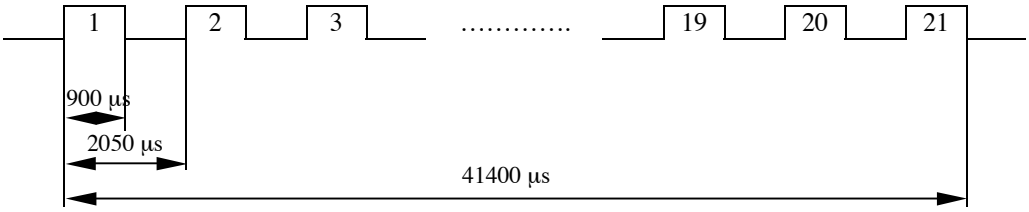

Figure 5.13 : Diagramme des temps de la routine d'interruption

Ceci montre que pour le capteur utilisé, qui à une résolution de 164 par 126 pixels, le processeur doit réaliser 21 interruptions. En affichant le nombre de mots vidés pour chaque interruption, on voit qu'il lit en moyenne 984 mots pour chacune. Ceci est tout à fait normale car :

984[mots/irq] x 21[irq] = 164[colonnes] x 126[lignes] x 1[mot/pixel] = 20'664 mots

La question est pourquoi le processeur vide 984 mots alors que l'interruption est levée à 512 ?
La réponse est simple. Tout d'abord il y a le temps d'entrer dans la routine d'interruption, déjà à ce moment quelques mots ont été ajoutés au contenu du fifio. En plus, pendant le temps de lecture, le fifo continue à se remplir. Il est donc possible de conculre qu'il y a suffisamment de marge avant d'avoir un dépassement de capacité.

Il est également possible d'en déduire la vitesse d'acquisition. En effet, la durée d'une acquisition est d'environ 41.4[ms]. Ce nombre indique qu'il y a un taux de rafraîchissement de 25 images par secondes.

Le problème vient du rapport entre le temps occupé et le temps libre du processeur. Le rapport du temps de traitement des interruptions avec le temps total vaut environ 45%. C'est-à-dire que 45 % du temps du processeur durant une acquisition est consacré à la lecture des données. Ceci donne une limite de débit à l'interface. Il faudra donc réaliser un compromis entre le taux de rafraîchissement et la taille de l'image. Le débit maximum est facilement déterminable. Il vaut :

900 us -> 987 bytes => ~1.1 Mbytes/s

En utilisant ce capteur, chaque image a une taille d'environ 20 Kbytes. Le taux de rafraîchissement est de 25 images par seconde. Donc le débit est d'environ 500 Kbytes/s.
Dans le cas où l'image est trop volumineuse, le processeur n'aurait pas le temps de vider le fifo dans les 40 ms lui étant alloué. Ces 40 ms sont en principe fixe et proviennent du taux de rafraîchissement de la caméra. Il est donc impossible de ralentir la caméra. L'interface est suffisamment rapide pour accepter cette vitesse, mais le processeur pas. La seule solution est d'augmenter la taille du fifo afin d'éviter un overflow.
Une solution permettant de quadrupler la vitesse de transmission entre le processeur et l'interface est d'envoyer 4 mots à la fois. En effet, une lecture de 32 bits est réalisée, mais uniquement 8 bits sont utilisés. Cette amélioration est facilement réalisable. Le débit maximum serait alors d'environ 4.4 Mbytes/s.

Le processeur réalise donc une lecture en environ 1 us. Ceci s'explique par le fait qu'il doit réaliser une douzaine d'instructions. Il doit en plus réaliser 4 accès mémoire pour la lecture et l'écriture de données. Ce processeur réalise une opération tous les 30 ns. Chaque opération est réalisée en 3 cycles. Ces différents paramètres conduisent à un temps de lecture du FIFO d'environ 960 ns.

Chapitre 6

Module Ethernet

6.1 Remarques préalables

Comme expliqué précédemment, le kit de développement comprend trois parties. La première est évidemment un module physique. Il vient s'enficher sur la carte fournie avec le kit de développement Nios.

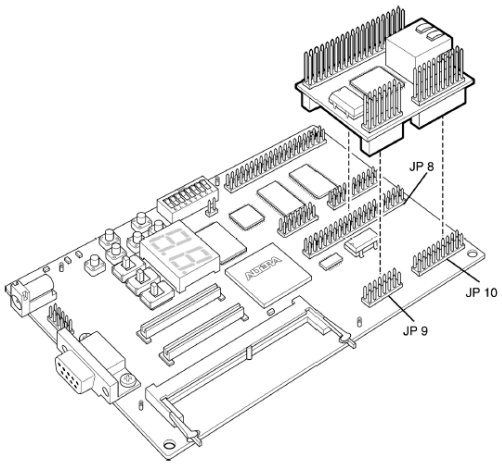

Figure 6.1 : Installation du module Ethernet sur la carte Excalibur, source [4]

Une partie très importante qui facilite énormément le travail est l'ensemble de librairies. Il y a évidemment les librairies permettant d'utiliser le module. Il y a en plus une partie qui gère les protocoles réseaux du type Ethernet. Cette librairie permet en particulier d'utiliser TCP/IP.
La dernière partie est une petite documentation. Elle est un peu restreinte, mais quelques exemples d'architectures ainsi que de programmes sont livrés et permettent de mieux comprendre les détails d'implémentation.
Les généralités d'utilisation de ce module sont expliquées dans ce chapitre. Pour plus de détails, il est préférable de lire directement la documentation du Kit de développement Ethernet.

6.2 Interface avec le processeur Nios

La connexion du module Ethernet peut paraître inexacte. Cependant, ce branchement est conforme et fonctionne correctement. La documentation fournit un branchement similaire. Il est important de savoir qu'il est possible d'enficher deux cartes Ethernet l'une sur l'autre. Le bit '0' du bus d'adresse de ce module est utilisé pour sélectionner l'une des deux cartes filles. Dans notre cas, nous travaillons uniquement avec une seule carte. Par conséquent, ce signal doit être connecté à la masse. Le bit '1' du bus d'adresse n'est pas connecté aux adresses du module car les accès se font sur une largeur de 16 bits. Il est par contre nécessaire à la sélection des poids faibles ou forts, ce bit est donc utilisé par le bus Avalon.

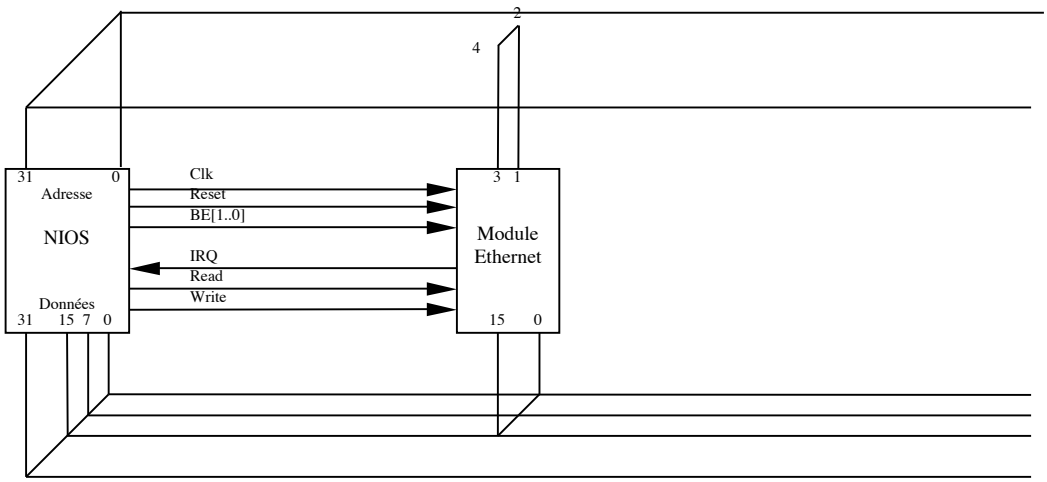

Figure 6.2 : Schéma logique de la connexion du module Ethernet

En ce qui concerne les autres signaux, il n'y a rien de spécial à signaler. Ils sont connectés comme n'importe quel autre périphérique du processeur.

6.3 Les "plugs"

Le cœur du fonctionnement de ce système est ce que l'on appelle les "plugs". Ces objets virtuels sont équivalents aux "sockets" de Unix. Ils permettent au développeur de l'application d'écrire un programme sans avoir à s'occuper des protocoles. Il existe un type de plug pour chaque couche réseau. Il existe donc les types suivants :

- plug TCP
- plug UDP
- plug IP
- plug Ethernet
- etc.

Il faut savoir que chaque plug est associé à une carte réseau. Il faut donc avoir préalablement initialiser chaque carte. Il est ensuite possible de créer et de lui associer un ou plusieurs plugs.
L'avantage est énorme. Le programmateur s'occupe simplement d'envoyer un paquet au plug désiré. Le système gère ensuite automatiquement l'ajout des différents en-têtes, les connexions, les retransmissions, etc.
Lors de la réception d'un paquet, l'interface Ethernet extrait les données de chaque couche réseau. Lorsque chaque protocole a été identifié, le paquet est envoyé à chaque plug concerné. Par exemple, prenons le cas où deux plugs ont été créés, un TCP et un Ethernet. Un paquet TCP arrive. Il est décortiqué et ensuite il est transmis au plug TCP ainsi qu'au plug Ethernet.
Les différents protocoles supportés par la libraire fournie par Altera se résument de la manière suivante :

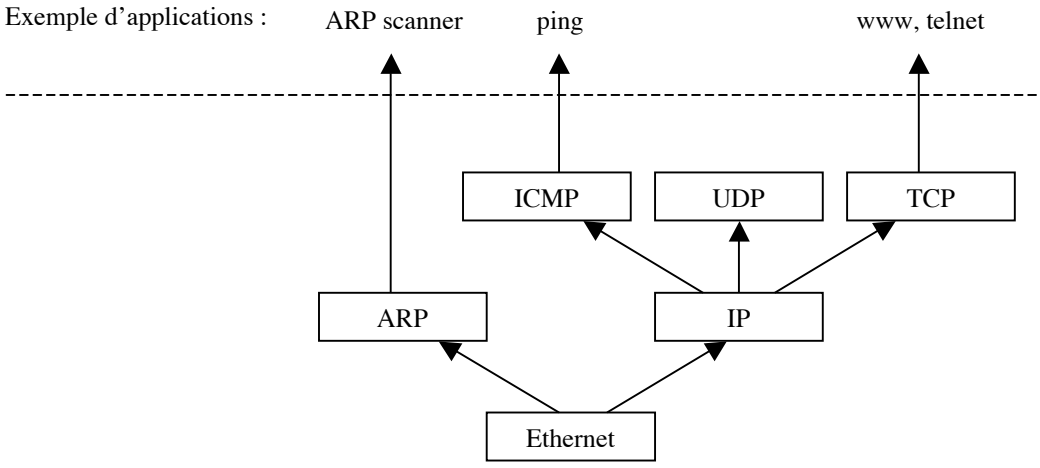

Figure 6.3 : Différents protocoles supportés par la librairie

En résumé, les paquets reçus sont d'abord traités afin d'extraire les différentes informations comprises dans les en-têtes de chaque couche réseau. Ceci réalisé, le paquet est envoyé au plug voulu sous forme d'une structure. L'en-tête est divisé en plusieurs attributs, comme par exemple, le port source, destination, la taille, etc. Les données sont également stockées dans un attribut. La structure d'un paquet TCP est la suivante :

- Port de source
- Port de destination
- Numéro de la séquence
- Numéro quittancé
- Taille de l'en-tête
- Taille de la fenêtre
- Checksum

De cette manière les informations utiles sont directement accessibles. A noter que les informations de l'en-tête IP sont également accessibles.

6.4 Insertion de l'interface dans l'architecture

Comme dans le cas des interfaces de la caméra, cette étape est réalisée à l'aide du logiciel "SoPC Builder" au travers de Quartus. Dans le kit de développement, le logiciel d'installation s'occupe d'ajouter les librairies pour que "SoPc Builder" reconnaisse l'interface Ethernet. Avant d'installer ce dernier, il faut installer un bus. Ceci permet de faire correspondre le bus Avalon avec le bus du module Ethernet, ceci dans le but de pouvoir choisir les pin's de sortie sur la logique programmable.

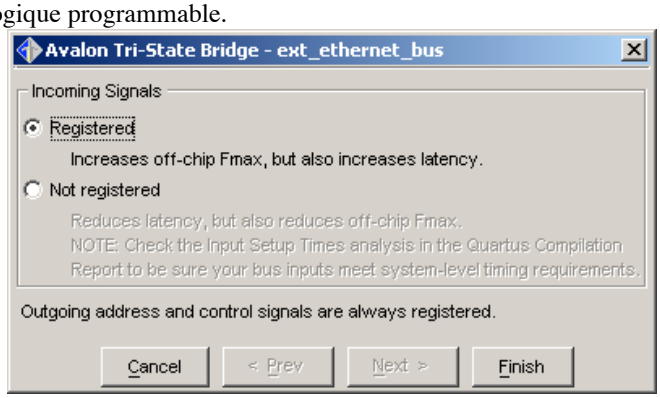

Figure 6.4 : Installation d'un bus

Ceci réalisé, il est possible d'additionner au schéma l'interface Ethernet. Il faut alors choisir quelques paramètres. L'interface est la suivante.

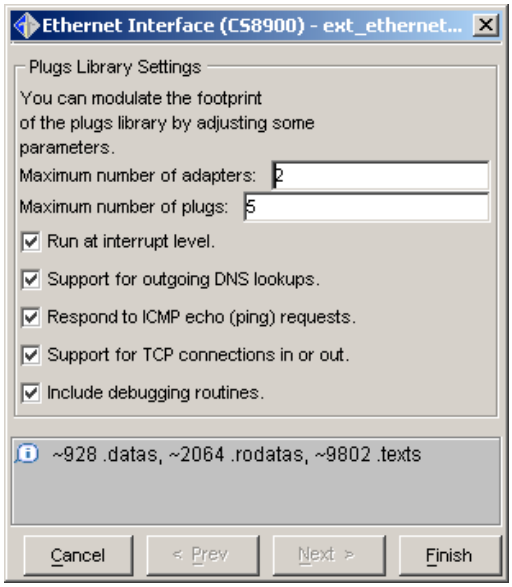

Figure 6.5 : Installation de l'interface Ethernet

Avec le kit de développement Ethernet, des exemples d'architectures sont livrés. Il est plus facile de démarrer de l'une de ces architectures que directement de la documentation de ce kit.

6.5 Utilisation

Avant de pouvoir utiliser la carte, il faut tout d'abord l'initialiser. Ceci est réalisé à l'aide de la commande "nr_plugs_initialise". Il faut en particulier décrire la carte à l'aide d'une structure. Cette structure contient :

- l'adresse Ethernet de la carte (48 bits)
- l'adresse IP de la carte (32 bits)
- le nom carte
- le masque de sous-réseau (32 bits)
- l'adresse IP du routeur par défaut (32 bits).

Cette structure est nommée "ns_plugs_network_stting. Elle définit en réalité une configuration réseau standard.
Il existe une commande inverse permettant de désinitialiser une carte. Elle est en réalité nommée "nr_plugs_terminate".

Avant d'utiliser la carte, il est obligatoire de créer au moins un plug. Ceci est réalisé à l'aide de la commande "nr_plug_create". Cette fonction nécessite différents paramètres dont le type de protocole afin de savoir quels paquets lui seront transmis.
Comme dans le cas de l'initialisation de la carte, il existe une commande permettant de détruire le plug. Cette fonction est "nr_plugs_destroy". Le déroulement est identique aux sockets de Unix. Il faut les créer, puis les détruire.

Une fonction très importante est "nr_plug_idle". Elle attend des packets entrants et les distribue vers les plugs concernés.

Deux autres routines utiles sont "nr_plugs_connect" et "nr_plugs_listen". La première permet d'ouvrir une connexion TCP. La seconde réalise l'opération inverse. Elle permet d'attendre une requête de connexion TCP. Ces deux commandes sont très utilisées dans le cas d'un serveur Web par exemple.

Enfin, une routine permettant d'envoyer un paquet est fournie. Elle se nomme "nr_plugs_send". Il faut la manipuler avec précautions, car les paquets ne sont pas découpés par l'interface. Le découpage doit être réalisé par le programme principal.

Il existe bien d'autres routines, mais seules les plus importantes et les plus utilisées ont été décrites.

Chapitre 7

Serveur Web

7.1 Pourquoi utiliser un serveur Web ?

Plusieurs solutions auraient été envisageables afin de transmettre des images à un utilisateur réseau. La première solution aurait été de développer un logiciel optimisé pour cette fonctionnalité. L'utilisateur est alors obligé de posséder ce logiciel pour pouvoir lire les images. De plus, cette solution n'améliore pas réellement le débit dans le cas où les transmissions auraient utilisées le protocole http. Une autre solution aurait été d'utiliser le protocole UDP. Celui-ci permet une vitesse de transmission élevée. Cet avantage provient du fait que ce protocole n'est pas fiable, il n'y a pas d'attente à réaliser pour les quittances, il n'y a également pas de retransmission, la perte de données est autorisée, les en-têtes sont plus courts, etc. En bref, il y a énormément d'avantages qui prédise un système plus rapide. Un inconvénient majeur est le fait qu'il est nécessaire de disposer d'un logiciel approprié pour pouvoir lire les images, ce qui n'est pas le but de ce travail.
En utilisant, un simple navigateur Web, il est possible à n'importe quel utilisateur de visionner les images de n'importe où. Il lui suffit d'avoir un ordinateur connecté au réseau possédant un navigateur Web. L'inconvénient de cette approche est la création des images. En effet, le processeur embarqué doit tout d'abord acquérir les images, il doit ensuite les convertir dans un format lisible par un navigateur, pour enfin créer le message http et l'envoyer. Ces étapes sont relativement lourdes pour le processeur. Un autre inconvénient sera le fait d'utiliser le protocole http. Ce protocole utilise TCP. A chaque perte de données, il faudra retransmettre le paquet. De plus, chaque paquet doit être quittancé, ce qui impose une transmission bidirectionnelle presque en permanence. Le taux de rafraîchissement ne sera alors pas très élevé.

7.2 Rôle d'un serveur Web

Avant de partir dans les détails, il est important de savoir quel est le rôle d'un serveur Web. Prenons comme exemple le cas de notre système. Pour ne pas compliquer, le protocole http est expliqué sans ce soucier des couches inférieures.

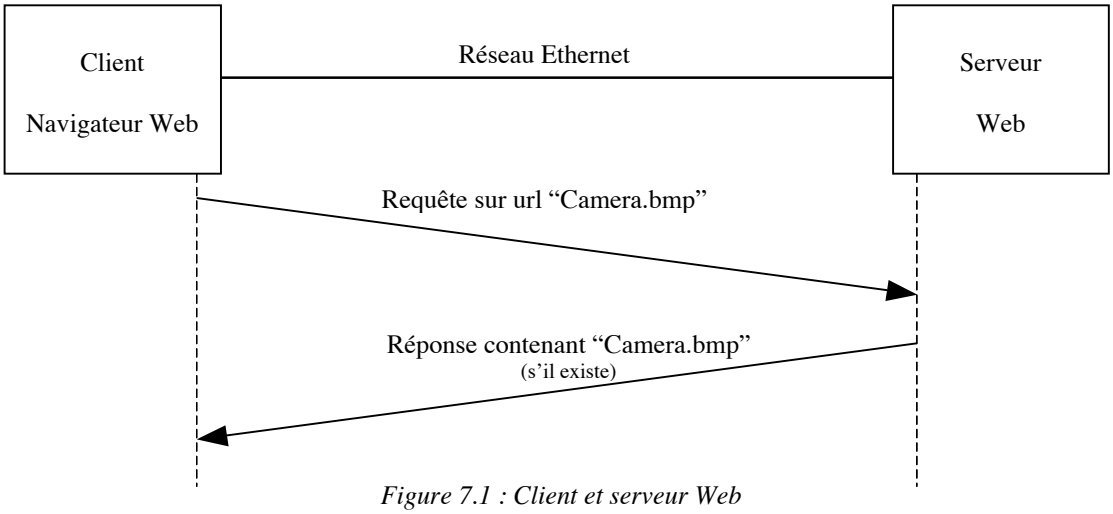

Figure 7.1 : Client et serveur Web

Le système est simplement composé de 3 éléments. Le premier est le serveur Web, il est en fait l'objet de ce travail. Il y a également le client, dans ce cas il s'agit d'un simple navigateur Web. Enfin, il y a le réseau Ethernet permettant de connecter les deux. Ces trois éléments froment la base de tout réseau informatique. Dans la réalité, il n'y aura pas uniquement une connexion Point-to-point, mais un réseau complet.
Le rôle du serveur est simple. Il doit attendre une requête http lui étant destinée. Le requête est crée par l'utilisateur simplement en cliquant sur le lien de la caméra ou encore en tapant l'adresse de la caméra. Une fois reçue, la requête doit être traitée afin de pouvoir émettre une réponse. Plus tard, il sera possible de voir les différentes sources afin de créer une réponse. Une fois le message créé, la réponse est envoyée sur le réseau.

7.3 Le serveur fournit par Altera

Dans le kit de développement Ethernet, il est fournit un exemple de serveur Web. Plusieurs types de pages Web peuvent être fournies. Ces différentes réponses se différencient par la source des données que le serveur va transmettre. Sur les schémas caractérisants chacune de ces réponses, la source est identifiée sous forme d'un bloc assombri.

7.3.1 Pages statiques

Le serveur est capable de fournir des pages html provenant de la mémoire. En effet, il est possible de stocker des fichiers dans la mémoire Flash. Ces fichiers peuvent être de toutes sortes, comme par exemples une page html, une image gif ou bmp, etc.

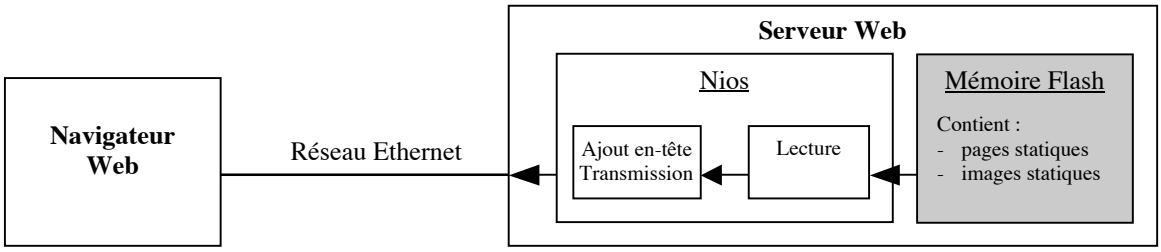

Figure 7.2 : Pages statiques

Pour stocker des fichiers, il suffit d'utiliser la commande "wosfs_maker" permettant de convertir des fichiers en une zone mémoire. Le résultat de la conversion est stockée ici dans le fichier "pages.flash" :

wosfs_maker.pl [liste de fichiers source] > pages.flash

Ceci réalisé, il faut encore écrire cette zone dans la mémoire. Pour cela, il faut utiliser la commande suivante :

Nr –x pages.flash

Le serveur peut alors retrouver chaque fichier uniquement en connaissant son nom.

Ces réponses sont nommées "pages statiques", car aucune transformation est réalisée. Le serveur transmet simplement l'équivalent de la zone mémoire. Il doit en réalité ajouter l'en-tête http.

7.3.2 Pages dynamiques

Ces pages sont entièrement crées par le serveur Web, donc par le processeur Nios.

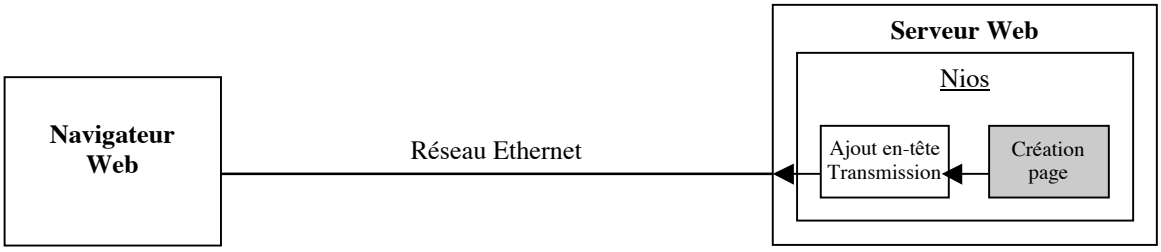

Figure 7.3 : Pages dynamiques

L'avantage est qu'il est possible de créer des pages modifiables selon le contexte. L'inconvénient est que le processeur doit travailler et il ralentit le temps de réponse. Ces pages sont simplement générées en imprimant du texte dans un buffer. Une fois la page terminée, le buffer est transmit au réseau sous forme d'une réponse http.

7.4 Création dynamique des images

La capture des images se fait de la même manière que décrite dans le chapitre 5. Trois fonctions ont été écrites. La première lance la capture d'une image. La seconde traite les interruptions. La dernière s'occupe d'attendre que la caméra aie terminée. Ses trois fonctions permettent par exemple, de lancer une acquisition. Il est alos possible de traiter d'autres informations durant la capture. Les données sont automatiquement gérées par la routine d'interruptions. Lorsque l'image est réellement nécessaire, il faut attendre, si besoin, que la caméra ait terminé sont acquisition.
Une fois l'image en mémoire, il est possible de la convertir de manière à ce qu'elle soit lisible par un navigateur Web. Il faut tout d'abord choisir le format de l'image qui sera transmise. Afin de ne pas trop charger le processeur, le format bitmap (BMP) a été sélectionné. Les images sont plus grosses qu'un format JPEG par exemple, donc le temps de transmission sera également plus grand. En contre partie, les temps de compression sont nuls. Finalement, il ne doit donc pas y avoir une grande différence de débit.
Le format bitmap a l'avantage d'être simple. Ce format est en fait codé en RGB, c'est-à-dire que chaque pixel est codé à l'aide d'un mot de 8 bits pour chacune des trois couleurs. Il y a donc 24 bits par pixel. Il est également nécessaire de générer un en-tête. Ce dernier contient une dizaine de paramètres, comme par exemple, le format, la compression, le nombre de bits/pixel, la taille de l'image, la taille du fichier (en-tête compris), etc.

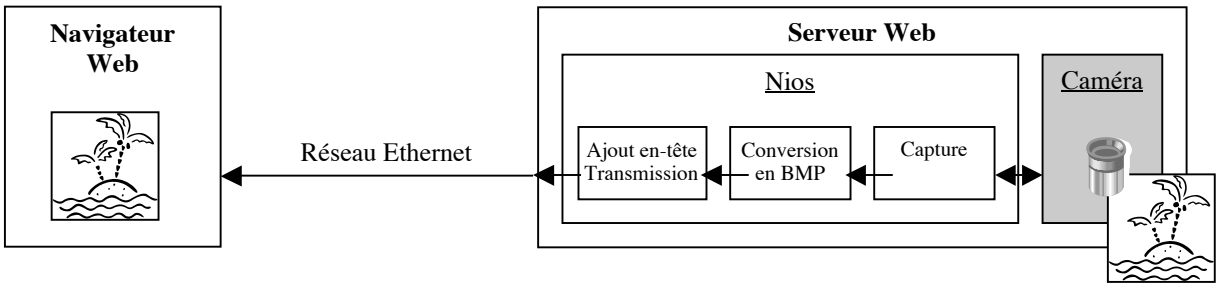

Figure 7.4 : Images dynamiques

Il faut noter que l'image est enregistrée dans une zone mémoire en BMP à l'aide d'une structure définissant chacun des paramètres de l'en-tête ainsi que le champ de données.
Une fois l'image en mémoire, il est possible de l'envoyer simplement en transmettant la zone mémoire sous forme d'une réponse http. Il est important également de découper le message en morceaux. En effet, le protocole Ethernet impose un paquet d'une taille maximum de 1500 bytes. Les librairies implémentés par Altera ne tiennent pas comptent de la segmentation. Il faut donc la réaliser soi-même dans le logiciel principal, ici dans le serveur Web.

7.5 Réponse à une requête

Afin de déterminer les étapes à réaliser lors de la création de la réponse, le serveur commence par déterminer si la réponse demandée est une page HTML, une image GIF ou une image BMP. Ceci est réalisé simplement en vérifiant le dernier caractère de l'URL.
Dans le cas d'une image GIF, il n'y pas d'autre solution que d'aller la chercher en mémoire.
Dans le cas d'une image BMP, il faut commencer par voir si elle est crée dynamiquement par le processeur. Si ce n'est pas le cas, il faut également aller la chercher en mémoire. Il est important de réaliser d'abord le test sur l'image dynamique car de cette manière lors d'une séquence d'images, le rafraîchissement sont plus rapide du fait qu'il n'y pas besoin de chercher en mémoire.
Dans le cas d'une page HTML, il faut commencer par vérifier s'il elle existe en mémoire. Si ce n'est pas le cas, il faut vérifier s'il est possible de la créer dynamiquement.
Dans le cas où la source n'a pas été trouvée, le serveur Web devra envoyer la page d'erreur 404.

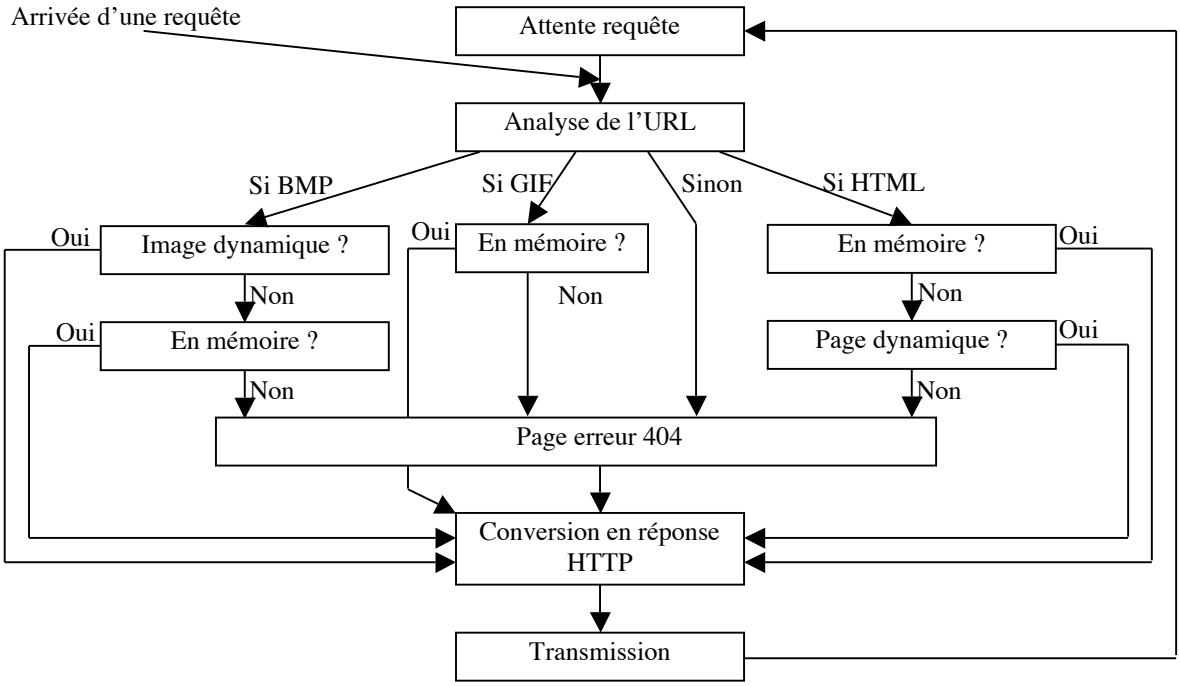

Figure 7.5 : Traitement d'une requête

Dans tous les cas, le Serveur devra convertir l'objet en une réponse HTTP. Il devra finalement terminer par l'envoi de celle-ci.
Evidemment, dans chacun des blocs la création de l'objet en question est réalisée. Ces instructions n'ont pas été ajoutées pour ne pas charger le schéma.

7.6 Concurrence : stratégies de capture et de transmission

Ce paragraphe traite de la concurrence entre la caméra et le traitement des requêtes. L'idéal en réalité serait de réaliser une capture juste avant de recevoir un requête demandant une image de la caméra. Il est évidemment impossible de prédire l'arrivée d'une requête. Différentes solutions sont proposées permettant la capture d'images de manière plus ou moins performante.

7.6.1 Capture pendant le traitement de la réponse

La première solution venue à l'esprit est de capturer une image lorsqu'on a reçu la requête. Cette manière à l'avantage d'avoir une image du moment exacte. Ceci impose donc de capturer l'image dans la routine d'interruption de la requête. Le problème est que cette méthode ralentit beaucoup le traitement de la requête. Il est en principe mauvais de passer trop de temps dans une routine de ce genre. Le deuxième inconvénient est que le navigateur reçoit la réponse "longtemps " après la requête. Il est possible qu'il y ait déjà eu le "time to live" de la requête envoyée et le navigateur ne tient pas compte de la réponse.

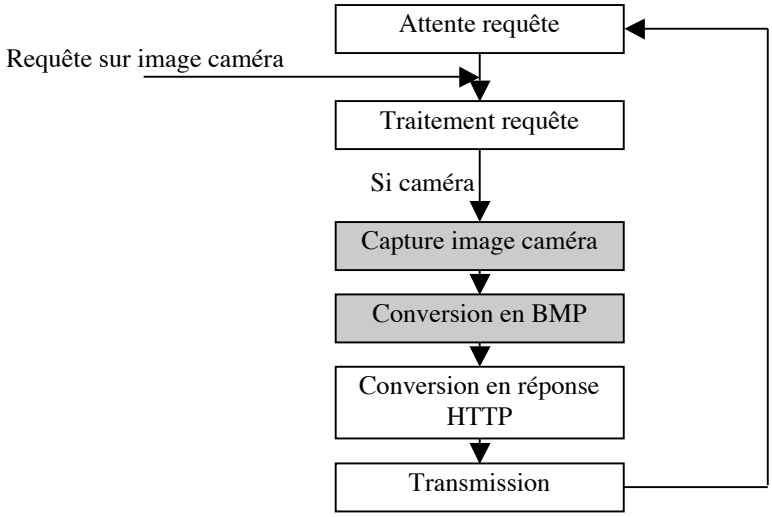

Figure 7.6 : Capture pendant le traitement de la réponse

7.6.2 Capture en permanence, transmission de la dernière

Afin de ne pas trop ralentir le traitement d'une requête, il faut donc réaliser la capture de l'image avec sa conversion en dehors du traitement de la requête. Il suffit pour cela de capturer des images en permanence. Lors d'une requête, la dernière image est envoyée.

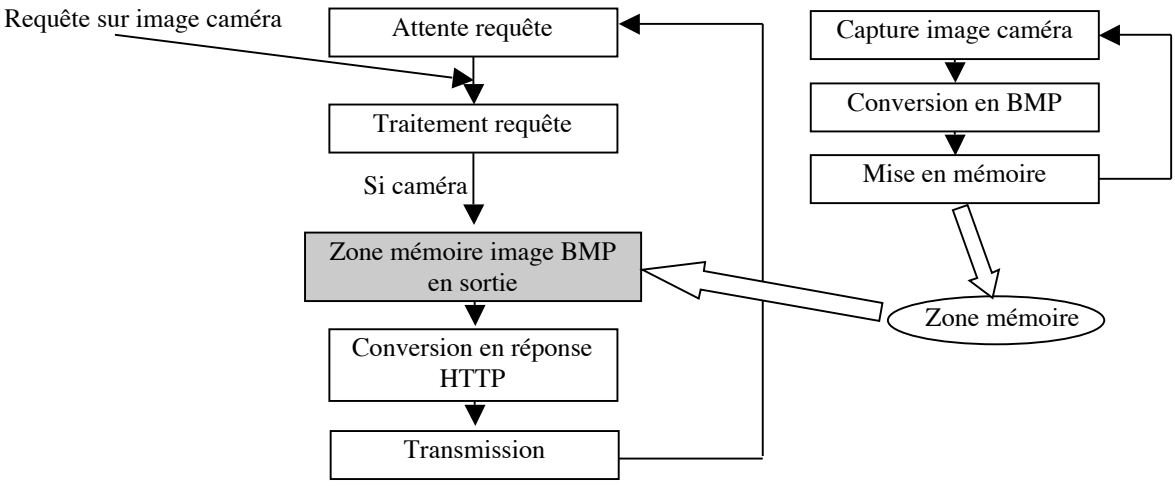

Figure 7.7 : Capture en permanence, transmission de la dernière

Cette méthode pose un énorme problème. En réalité elle ralentit la création d'une réponse. Lors de la capture d'une image, la caméra démarre une acquisition. A chaque fois qu'il faut vider le FIFO de l'interface de données de la caméra, une interruption est levée. Entre chaque interruption, il y a du temps libre. Si une requête HTTP arrive pendant ce temps, elle est traitée. Le problème arrive maintenant car la caméra va faire une interruption. Elle a un niveau prioritaire et sera traitée. Par conséquent, la requête devient excessivement lente, car elle est à tout moment interrompue. Une solution serait de donner la priorité à l'interface Ethernet. Dans ce cas, il y aurait des pertes de données du aux overflows de FIFO contenu dans l'interface caméra.

7.6.3 Capture juste après une réponse

Comme il n'est pas possible de prédire l'arrivée d'une requête et qu'il n'est pas possible n'ont plus d'intégrer la capture d'une image dans le traitement de la requête, une dernière méthode à été développée.
En fait, la capture d'une image est réalisée juste après une lecture de celle-ci. Si une requête arrive, elle est traitée. Dans le cas où l'image a été lue, une nouvelle capture sera réalisée. En fait, une image ne sera jamais envoyée plusieurs fois. Par contre, si une image a été lue il y a dix minutes. La requête suivante aura une image datant de 10 minutes. Ceci est un inconvénient, mais il n'est pas gênant pour l'utilisation. En réalité, il y aura des requêtes toutes les demis seconde. Que la capture soit réalisée juste avant ou juste après le traitement de la requête ne change rien. L'image aura de toute manière un retard du au temps de transmission.

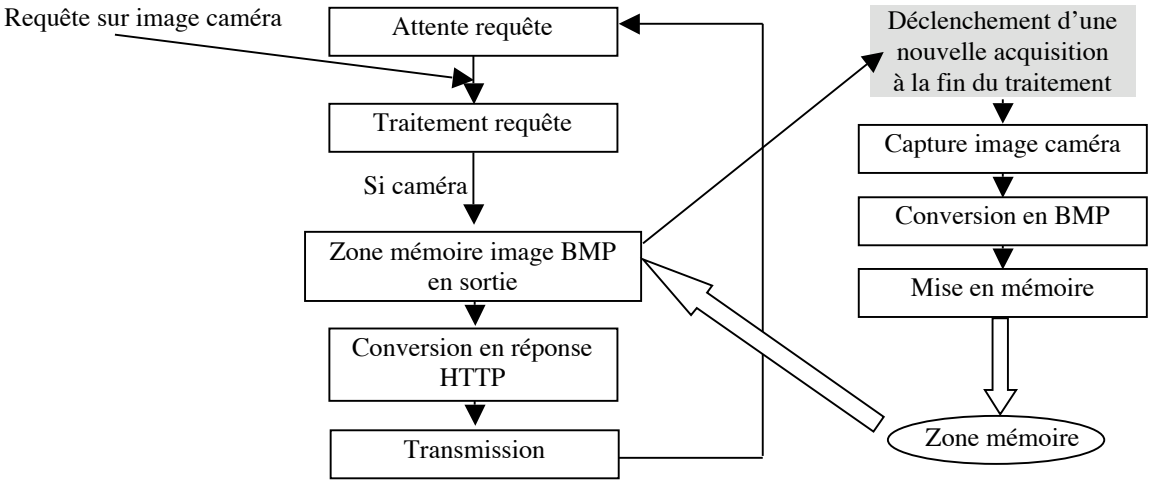

Figure 7.8 : Capture juste après une réponse

La seule différence est que la capture est réalisée qu'une seule fois entre chaque lecture. Par conséquent, si l'on laisse suffisamment de temps au système entre deux requêtes pour traiter la requête et capturer une image, le système sera très performant.
Si au contraire, le temps est trop faible, une requête sera envoyée avant que la capture soie terminée et les interruptions de la caméra interrompront la routine de traitement des requêtes. A ce moment le système devient très lent.

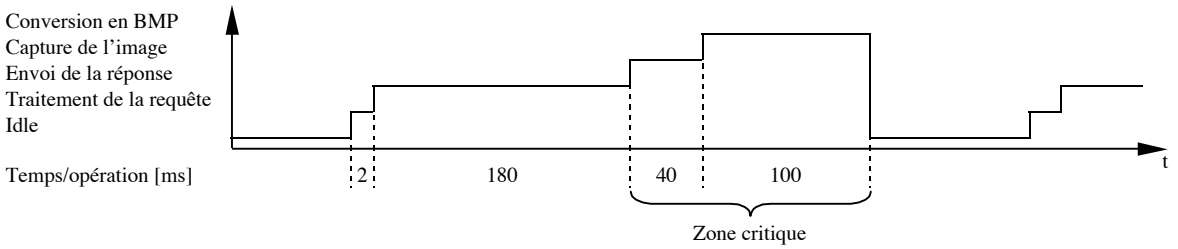

Figure 7.9 : Mesure du temps par opérations

En résumé, la dernière solution est idéale pour autant que le temps entre deux requêtes soit suffisamment long. Un moyen pour accélérer ces temps serait de travailler avec deux processeurs. Le premier s'occuperait de la caméra et le second du traitement réseau. De cette manière, le temps par image serait d'environ 200 ms contre presque 400 dans le cas traité ici.

7.7 Page HTML gérant les rafraîchissements

7.7.1 Page HTML pure avec la commande “refresh”

La façon la plus simple de rafraîchir une page Web est d’entrer la ligne de commande dans le header.

```
<BODY onLoad=window.setTimeout("location.href='pic_page1b.html'",100)>
```

De cette manière, la page est appelée tous les x secondes. Cette manière de rafraîchir l’image fonctionne correctement. Cependant il y a un gros défaut. L’image clignote et la vision n’est pas agréable. Le clignotement s’explique facilement. La page est chargée. Une fois le chargement terminé, l’image à son tour doit être téléchargée. Lorsque l’image est complète, la nouvelle page ainsi que la nouvelle image doivent à nouveau être actualisée. Pendant le temps de chargement de l’image, le navigateur n’a rien à afficher, par conséquent la page est blanche. Puis l’image est en mémoire, alors elle est affichée. Ensuite elle est rafraîchie, la page est blanche, et ainsi de suite.
Ce type de rafraîchissement est utilisé pour de longue période, de l’ordre de quelques dizaines de secondes. De cette manière, l’image est affichée pratiquement la totalité du temps. Lors de la transition, il y a aura durant une petite fraction de seconde une image blanche, mais cela est négligeable. C’est par contre inutilisable dans notre cas, car le temps d’affichage d’une image est environ égal au temps de chargement de cette image. Le temps affiché est par conséquent presque nul.

7.7.2 Page HTML utilisant Javascript

L’une des autres manières possibles est simplement d’utiliser le langage Javascript. Il est en principe utilisable sur tous les navigateurs Web. Il a en plus l’avantage de ne pas trop ralentir le navigateur.
Pour réaliser un rafraîchissement, il suffit de réaliser une nouvelle requête en ajoutant à l’URL, la date, comprenant l’heure actuelle. De cette manière, le navigateur sait que c’est une nouvelle image, car l’URL est différente. Le serveur Web quand à lui élimine cette partie ajoutée et ne s’occupe que de répondre par une nouvelle image.
Ce langage permet également de savoir si l’image a totalement été chargée. De plus, il permet de gérer un timer.
A l’aide de ces trois aspects, il est possible de réaliser un petit script permettant le rafraîchissement de l’image. Le navigateur doit en premier lancer le téléchargement de l’image. Lorsque celle-ci est totalement en mémoire, il l’affiche. Un certain temps plus tard, il recommence l’acquisition et ainsi de suite.

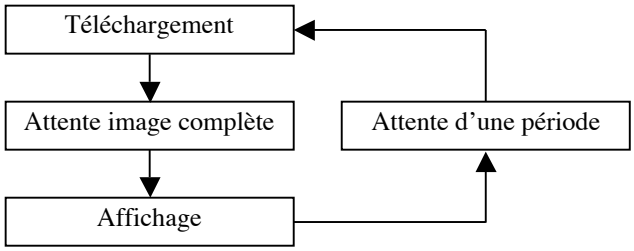

Figure 7.10 : Script de rafraîchissement

L’image est alors rafraîchie sans clignotement et de manière régulière. La période est réglable ainsi que la taille d’affichage de l’image. Ces deux réglages sont simplement réalisés à l’aide de boutons et de fonctions réalisées en Javascript puis incorporé dans la page html.

Chapitre 8

Adaptation du système à la carte RokApex

8.1 Motivations

Jusqu'à maintenant, toutes les fonctions ainsi que toutes les architectures ont été conçues pour la carte Excalibur. Cette carte est livrée par Altera, elle fait en fait partie du kit de développement Nios.

Dans le but d'installer le système sur le robot cyclope, le travail a été adapté à un nouveau design. La carte qui sera utilisée est la carte RokApex. Elle a été développée par Cédric Gaudin lors d'un projet de semestre précédent.

Au final, le système sera alimenté uniquement avec des accus. En effet, une carte d'alimentation est en train d'être développée à cet effet. Le système sera donc entièrement autonome.

8.2 Architecture

L'architecture en elle-même n'est pas modifiée. Cependant certains détails doivent être adaptés. En réalité, la base de l'architecture a été reprise du projet de Cédric Gaudin. Il suffit alors de lui ajouter les interfaces de la caméra ainsi que celle du module Ethernet.

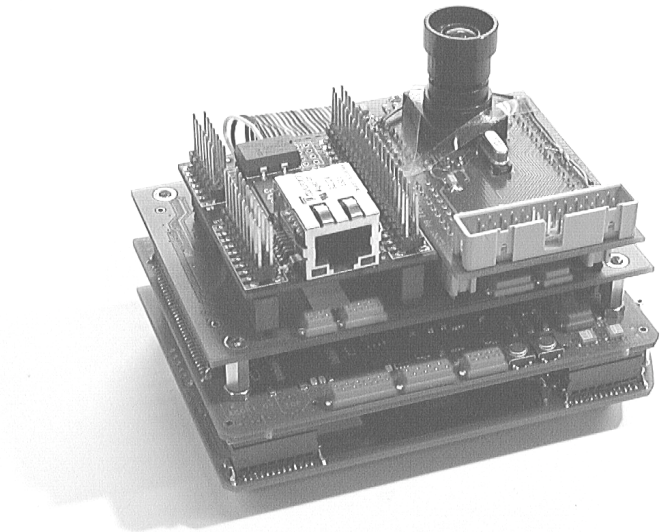

Figure 8.1 : Carte RokApex avec caméra et module Ethernet

Le point qu'il faut entièrement refaire est l'assignation des pins. Là aussi, le projet de Cédric Gaudin a été pris comme point de départ. Les pins concernant la caméra et le module Ethernet ont alors été assignées.

8.3 Le serveur Web

A ce stade du développement, il n'est pas possible d'écrire dans la mémoire Flash. Le problème est que le serveur Web nécessite de placer les pages html ainsi que les images statiques dans cette mémoire.

Deux solutions se prêtent alors. La première est de chercher un moyen d'écrire dans cette mémoire. L'inconvénient est qu'il faut entièrement reprendre le design et chercher l'erreur. Par manque de temps, ceci n'est pas possible.

Par conséquent, la seconde solution a été prise comme secours. Elle consiste à intégrer toutes les pages Web dans le serveur. Nous savons qu'il est possible de générer des pages dynamiques. Il est donc également possible d'utiliser cette méthode pour générer des pages statiques. L'avantage de cette solution est qu'elle est relativement simple et assez rapide. Il est clair que c'est une solution provisoire. Dans le cadre de développements futurs, il serait important de pouvoir écrire dans la mémoire Flash.

8.4 En résumé

Le transfert d'une carte à l'autre est relativement rapide. Seules quelques petites modifications ont du être réalisées. Le plus gros travail est l'assignement des pins. A ce stade le serveur fonctionne correctement au détail près de la mémoire flash. En temps normal, le serveur ne nécessite pas de modification. Ici quelques petits détails ont été changés afin de palier au fait qu'il n'est pas possible d'utiliser cette mémoire.

Chapitre 9

Conclusion

9.1 Rappels des objectifs

Il s'agit de développer un serveur Web. Cette application permettra d'accéder aux images provenant de la caméra. De cette manière, seul un navigateur Web sera indispensable pour la lecture des images. Ce système devra être autonome. Cela signifie qu'il ne devra pas nécessiter un appareil externe tel qu'un ordinateur.

9.2 Apport de connaissances

Durant ce travail, j'ai appris énormément de choses. Le plus grand plaisir était de voir le lien entre chaque niveau. En effet, il a fallu tout d'abord commencer par réaliser l'architecture, ainsi que l'interface de données de la caméra en VHDL. Donc un travail de bas niveau. Ensuite, il a fallu réalisé un code permettant le fonctionnement d'un serveur Web autonome. Ceci fait l'objet d'un niveau supérieur. Enfin, il a fallu écrire des pages HTML, parfois en y incorporant du Javascript. D'où un travail de très haut niveau. J'ai trouvé très intéressant le fait de pouvoir concevoir une interface, par exemple, puis de devoir l'utiliser dans un logiciel, pour enfin la voire fonctionne au travers d'un réseau.
Une autre grande partie des connaissances apportées par ce projet est l'utilisation des outils de développement. En effet, les premières semaines ont été consacrées uniquement à l'apprentissage de l'environnement. C'est-à-dire comprendre le fonctionnement et l'utilité de chaque logiciel, connaître le processeur à disposition, etc.
Ce travail était très motivant. Petit à petit le projet prenait forme. Tout d'abord, il a fallu faire fonctionner la caméra. Une fois ceci réalisé, c'était au tour du serveur Web. Pour ensuite voir des images se transmettre. Et enfin, il a fallu développé un procédé de rafraîchissements. C'est encourageant de voir une progression au fur et à mesure des semaines.

9.3 Travail effectué

Durant ce projet, les différentes étapes réalisées sont les suivantes.

1) En tout premier, l'interface de données de la caméra a été améliorée afin de faire fonctionner correctement la caméra. En réalité, l'interface de données a totalement été reconstruite.
2) Le programme CamAff a été corrigé. Il a uniquement été utilisé pour le développement de l'interface de la caméra. Il a permis d'utiliser la caméra sans le module Ethernet. Il est en réalité inutile dans ce projet.
3) Une architecture a été développée afin de rassembler les interfaces de la caméra ainsi que l'interface du module Ethernet.
4) Le serveur Web a été construit sur la base du serveur donné comme exemple par Altera.
5) La capture des images a pu y être ajoutée.
6) Il a fallu ensuite créer un convertisseur permettant de convertir ces images dans un format lisible par un navigateur Web quelconque.
7) Le serveur a été adapté afin qu'il réponde correctement aux requêtes sur une image dynamique.
8) Des pages Web ont été écrites afin de pouvoir gérer automatiquement les rafraîchissements des images. Ces pages contiennent des fonctions écrites en Javascript.
9) Pour terminer, tout le système a été adapté à la carte RokApex.

9.4 Comparaison par rapport au marché

- ✓ Une grande partie des solutions offertes sont en réalité des webcams connectées à un ordinateur au travers du port USB. Un logiciel permet alors de créer l'interface avec le réseau, en fait de jouer le rôle du serveur Web. L'avantage du système développé ici est qu'il est entièrement autonome.
- ✓ Un autre grand avantage est qu'il est accessible par navigateur Web conventionnel. Les entreprises du marché proposent souvent des solutions utilisant un logiciel spécialisé. De cette manière, ils peuvent en quelques sortent protéger leur travail. Dans notre cas, aucune protection n'est nécessaire.
- ✓ Un énorme avantage est que nous sommes en possession de tous les codes sources. De cette manière, il est possible de reprendre ce travail afin de l'améliorer, de le modifier, etc.

9.5 Travail futur

- → Une amélioration possible serait simplement de changer le capteur de la caméra. Les images pourraient être de meilleure qualité en augmentant la résolution du capteur.
- → Il serait envisageable de travailler avec deux processeurs. L'un s'occuperait de capturer les images, de les convertir puis de les mettre ne mémoire. Le second s'occuperait uniquement de traiter les requêtes. De cette manière, il y aurait un parallélisme. La vitesse de l'ensemble serait alors nettement accélérée car les temps d'attente pour la capture et la conversion seraient inexistants (vu du navigateur Web).
- → Il n'est pas possible d'augmenter le débit binaire. La seule solution pour augmenter le débit d'images est de diminuer leur taille. Il serait donc possible de créer un dispositif permettant de compresser les images. Il faut cependant faire attention au format choisi. Il faut qu'il soit lisible par un navigateur Web. Le format le plus courant est bien sûre le JPEG.
- → Enfin, il serait possible de remplacer le module Ethernet par un module Ethernet sans fil (Wireless). Ceci permettrait par exemple le pilotage d'un robot au travers du serveur Web. Il serait tout à fait imaginable de concevoir une petite interface Web, avec quelques commandes, tel que avancer, reculer, gauche, droite, etc. De plus, il serait possible de visionner les images en "direct".

Bibliographie

Projets existants

[1] Cédric Gaudin, “Caméra embarquée”, projet de semestre LAP 2002

[2] Julien Chastaing et Laurent Chatelanat, “Firewall”, projet de semestre LAP 2002

Documentation sur le matériel fournit

[3] Documentation Nios 2.0, Altera 2002

[4] Documentation Nios Ethernet 2.0, Altera 2002

Réseaux informatiques

[5] Eric A. Hall, “Internet Core Protocols : The Definitive Guide”, édition O’Reilly 2000

[6] Richard Stevens, “TCP/IP illustré volume 1”, édition Vunibert 1998

[7] Silvia Giordano, “Computer Networking I”, EPFL 2002

Caméra CMOS

[8] Documentation sur le capteur VV6301 de STMicroelectronics

Lexique

Architecture des processeurs

FPGA	Field Programmable Gate Array
RAM	Random Access Memory
ROM	Read Only Memory
Nios	Processeur soft-core, c'est-à-dire intégrable dans une logique programmable
Flash	Mémoire conservant les données même sans alimentation.

Réseaux informatiques

IP	Internet Protocol
TCP	Transfert Control Protocol
UDP	User Datagram Protocol
MAC	Media Access Control
LAN	Local Area Network
PPP	Point-to-Point Protocol
ICMP	Internet Control Message Protocol
ARP	Address Resolution Protocol
SMTP	Simple Mail Transfer Protocol
CSMA/CD	Carrier Sense Multiple Access / Collision Detection

Internet

WWW	World Wide Web
URL	Uniform Reference Location
HTTP	HyperText Transfert Protocol

Table des figures

Figure 1.1 : Structure générale 1
Figure 1.2 : Système utilisant un ordinateur 2
Figure 1.3 : Système à logiciel spécialisé (Source : Visual Inspection Systems) 3
Figure 1.4 : Organisation du travail 3
Figure 2.1 : Carte Excalibur, source [3] 4
Figure 2.2 : Carte Ethernet 5
Figure 2.2 : Carte Caméra2D (sans optique) 5
Figure 3.1 : Organisation en couches 7
Figure 3.2 : Un voyage aérien 7
Figure 3.3 : Encapsulation des données des couches hautes vers les couches basses, source [6] 8
Figure 3.4 : Paquet Ethernet 8
Figure 3.5 : Structure d'un paquet IP, source [6] 9
Figure 3.6 : Structure d'un paquet UDP, source [6] 10
Figure 3.7 : Structure d'un paquet TCP, source [6] 10
Figure 3.8 : Applications sur TCP ou UDP 10
Figure 3.9 : Connexion http 11
Figure 3.10 : Exemple requête d'une page html et de ses références 11
Figure 4.1 : Schéma bloc global 12
Figure 4.2 : Photo de la carte prête à l'emploi 13
Figure 4.3 : Interface SoPc Builder 13
Figure 4.4 : Importation d'un processeur Nios 14
Figure 4.5 : La mémoire ROM 14
Figure 4.6 : Les UART's 15
Figure 4.7 : Le timer 15
Figure 4.8 : Les boutons 16
Figure 4.9 : Les deux Led's 16
Figure 4.10 : L'affichage 7 segments à 2 digits 17
Figure 4.11 : La bus de mémoire externe 17
Figure 4.12 : La mémoire SRAM 17
Figure 4.13 : La mémoire Flash 18
Figure 4.14 : Schéma logique global 18
Figure 4.15 : Plan mémoire 19
Figure 4.16 : Graphique des zones mémoire 19
Figure 5.1 : Schéma logique de l'interface de configuration 20
Figure 5.2 : Schéma logique de l'interface de données 21
Figure 5.3 : Signaux de synchronisations 21
Figure 5.4 : Schéma bloc de l'interface de données de la caméra 22
Figure 5.5 : Machine d'états 23
Figure 5.6 : Installation des l'interfaces dans SoPc 23
Figure 5.7 : Installation de l'interface de configuration 24
Figure 5.8 : Installation de l'interface de données 24
Figure 5.9 : Registre de contrôle 24
Figure 5.10 : Registre d'état 25
Figure 5.11 : Registre de limite et de compteur 25
Figure 5.12 : Registre de données 25
Figure 5.13 : Diagramme des temps de la routine d'interruption 26
Figure 6.1 : Installation du module Ethernet sur la carte Excalibur, source [4] 27
Figure 6.2 : Schéma logique de la connexion du module Ethernet 28
Figure 6.3 : Différents protocoles supportés par la librairie 28
Figure 6.4 : Installation d'un bus 29
Figure 6.5 : Installation de l'interface Ethernet 29
Figure 7.1 : Client et serveur Web 31

Figure 7.2 : Pages statiques _____ *32*
Figure 7.3 : Pages dynamiques _____ *32*
Figure 7.4 : Images dynamiques _____ *33*
Figure 7.5 : Traitement d'une requête _____ *34*
Figure 7.6 : Capture pendant le traitement de la réponse _____ *35*
Figure 7.7 : Capture en permanence, transmission de la dernière _____ *35*
Figure 7.8 : Capture juste après une réponse _____ *36*
Figure 7.9 : Mesure du temps par opérations _____ *36*
Figure 7.10 : Script de rafraîchissement _____ *37*
Figure 8.1 : Carte RokApex avec caméra et module Ethernet _____ *38*