Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2010

Bastien Praplan

Composant ethernet en FPGA

Professeur François Corthay

Expert Claude Magliocco

Sion, le 12 juillet 2010

Données du travail de diplôme
Daten der Diplomarbeit

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2009/10**	No TD / *Nr. DA* **it/2010/28**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire	Etudiant / *Student* **Bastien Praplan**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire
Professeur / *Dozent* **François Corthay**	Expert / *Experte* *(données complètes)*	
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein		

Titre / *Titel*

Composant Ethernet en FPGA

Description et Objectifs / *Beschreibung und Ziele*

Le but de ce projet est d'utiliser une carte FPGA comme périphérique Ethernet. Un bloc d'interface Ethernet a déjà été développé : il enregistre les trames Ethernet arrivantes dans une mémoire et — pour l'autre direction — il émet des trames en parcourant le contenu d'une autre mémoire. Par ailleurs, ce bloc effectue un premier décodage des trames reçues et transmet les données utiles au bloc qui devra les traiter.

Ce projet a pour but d'implémenter les protocoles ICMP (commande "ping") et DHCP de manière à pouvoir être reconnu automatiquement au moment où le système est connecté au réseau Ethernet.

Les objectifs du projet sont de :

— mettre en route le circuit actuel et tester la fonction d'écho de trames UDP déjà à disposition
— réaliser un bloc implémentant le protocole ICMP, l'ajouter au système et le tester
— réaliser un bloc implémentant la fonction de demande d'adresse IP (DHCP), l'ajouter au système et le tester
— réaliser une application de démonstration du système.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
22.02.2010

Remise du rapport intermédiaire / *Zwischenbericht:*
07.05.2010, 17:00

Défense intermédiaire / *Zwischenverteidigung:*
21.05.2010

Remise du rapport / *Abgabe des Schlussberichts:*
12.07.2010, 12:00

Exposition publique / *Ausstellung Diplomarbeiten:*
27.08.2010

Défense orale / *Mündliche Verteidigung:*
Semaine 35 / *Woche 35*

Signature ou visa / *Unterschrift oder Visum*

Responsable de la filière
Leiter des Studiengangs: .....

[1] Etudiant/*Student*: .....

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Composant Ethernet en FPGA

Diplômant/e Bastien Praplan

Objectif du projet

Le but de ce projet est d'implémenter sur une FPGA le traitement du protocole ICMP pour répondre à une commande PING et le protocole DHCP. Une application communiquant par RS232 avec des modules de contrôle d'éclairage public est réalisée.

Méthodes | Expériences | Résultats

Les paquets arrivants sur la ligne Ethernet sont enregistrés dans une mémoire. Les en-têtes des protocoles sont ensuit enlevés et le données sont traitées. Dans le sens de l'envoi, les données sont enregistrées dans un autre mémoire et les en-têtes des protocoles sont rajoutés. Après quoi, le tout est envoyé sur le réseau Ethernet. La configuration réseau de la carte se fait automatiquement grâce au DHCP et une partie d'auto-détection envoie des messages à un serveur pour s'identifier et communiquer la configuration de la carte. Sur le serveur, une application enregistre les informations communiquées par ces messages pour permettre à d'autres applications de les utiliser.
La carte FPGA et la structure de base ont été fournies au début du projet. Un module de test a été donné pour permettre de tester la communication avec les modules.
Toutes les parties implémentées sur la FPGA ont été testées en simulation, puis dans le monde réel et ont montré un comportement correspondant aux demandes du cahier des charges. L'application qui enregistre les configurations des cartes sur le serveur répond également au cahier des charges.

Travail de diplôme
| édition 2010 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
François Corthay
francois.corthay@hevs.ch

Partenaire
-

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

1 TABLE DES MATIÈRES

2 TABLE DES FIGURES

3 INTRODUCTION

3.1 CONTEXTE DU PROJET

3.1.1 Objet du document

Ce rapport explique les développements apportés au projet Composant Ethernet en FPGA effectué dans le cadre du projet de diplôme du 6ème semestre, lors de l'année académique 2009/10.

3.1.2 Equipe de développement

Professeur : François Corthay francois.corthay@hevs.ch
Collaborateur : Sébastien Farquet sebastien.farquet@hevs.ch
Diplômant : Bastien Praplan bastien.praplan@students.hevs.ch

3.2 DESCRIPTION DU PROJET

3.2.1 Idée

Ce projet a pour but de programmer une FPGA pour qu'elle réagisse comme un périphérique Ethernet. Les paquets arrivants depuis Ethernet sont enregistrés dans une mémoire, décodés selon les types de protocole utilisés par ceux-ci et traités. Dans l'autre direction, les en-têtes des protocoles sont rajoutés aux données qui sont enregistrées dans une autre mémoire et, ensuite, envoyées sur le réseau Ethernet.

3.2.2 Etat initial

La carte FPGA et le module de test qui émule le contrôleur d'éclairage ont été fournis. Le module était programmé.
La structure de base était faite. La réception, l'envoi, le stockage des trames Ethernet, les blocs des protocoles Ethernet, IPv4 et UDP, ainsi que les blocs de traitement pour l'écho et le time UDP ont déjà été implémentés.

3.2.3 Cahier des charges

Ce travail a les objectifs suivants :

- Installation d'un environnement de test
- Mise en route du circuit existant
- Tests des fonctionnalités écho et time déjà existantes
- Implémentation et test de la partie protocole ICMP
- Implémentation et test de la partie de traitement des données de la commande PING
- Implémentation et test de la partie protocole DHCP
- Réalisation sur la FPGA d'une application de démonstration contenant les parties
  - Auto-détection
  - Transfert de données entre Ethernet et les ports RS232
- Réalisation sur le PC d'une application qui enregistre les données d'auto-détection
- Réalisation sur le PC d'une application qui discute avec les modules par Ethernet

3.3 ENVIRONNEMENT DE DÉVELOPPEMENT

Cross development sur FPGA :

- Mentor Graphics, HDL Designer, version 2007.1a
- Xilinx, ISE, version 10.1
- Xilinx, ISE, version 11.1

Linux :

- Debian 5
- libxml-simple-perl 2.18-1
- libdate-manip-perl 5.54-1
- dhcp3-server 3.1.1-6

Documentation :

- Microsoft Word
- Microsoft Excel
- Microsoft Visio

4 DESCRIPTION DU SYSTÈME

4.1 PROTOCOLES

4.1.1 Protocole ICMP

Le protocole ICMP (Internet Control Message Protocol) permet de véhiculer les informations de contrôle ou d'erreur du protocole IP. Il est encapsulé dans les paquets IPv4. La configuration du paquet est présentée dans la Figure 1.

Figure 1 : Trame ICMP

Les données ICMP sont composées de la manière suivante :

Bit 0 à 7	Bit 8 à 15	Bit 16 à 23	Bit 24 à 31
Type de message	Code	Somme de contrôle	
Bourrage			
Données			

Figure 2 : Données ICMP

- Les champs type et code représentent le message contenu. Une partie de la liste des combinaisons entre ses champs et leur signification se trouve en annexe 10.1.
- La somme de contrôle représente la validité du paquet ICMP. Le calcul de cette somme est le même que celui pour IPv4, mais se fait sur l'en-tête ICMP et ses données.
- La partie bourrage dépend du type du paquet. Si cette partie n'est pas utilisée, elle est remplacée par les données.

4.1.2 PING

La commande PING est un outil d'administration de réseau informatique qui permet de tester si une machine est atteignable et de mesurer le temps d'aller-retour. Un paquet ICMP contenant une demande d'écho est envoyée et une réponse d'écho est attendue.

4.1.2.1 Demande d'écho

La demande d'écho se fait par un paquet ICMP de type 8 et de code 0. La partie bourrage est composée d'un identifiant et d'un numéro de séquence qui permettent au client de déterminer quelle demande d'écho est la cause de telle réponse d'écho. Les données et le bourrage doivent être répétés dans la réponse d'écho.

4.1.2.2 Réponse d'écho

La réponse d'écho est envoyée dans un paquet ICMP de type 0 et de code 0. Dans la partie bourrage, se trouve l'identifiant et le numéro de séquence qui étaient dans la demande d'écho. Les données sont les mêmes.

4.1.3 Protocole DHCP

DHCP (Dynamic Host Configuration Protocol) est un protocole qui permet à un ordinateur de récupérer des informations pour sa configuration réseau. Les données de ce protocole sont dans des paquets UDP. La Figure 3 montre la configuration d'un paquet Ethernet contenant des données DHCP.

Figure 3 : Trame DHCP

Le client reçoit les données sur le port 68 et le serveur sur le port 67. Ces paquets sont envoyés par broadcast sur le réseau.

4.1.3.1 Format d'une trame DHCP

Les données DHCP ont le format suivant (les chiffres indiquent la taille, en byte, des champs) :

op (1)	htype (1)	hlen (1)	hops (1)
xid (4)			
secs (2)		flags (2)	
ciaddr (4)			
yiaddr (4)			
siaddr (4)			
giaddr (4)			
chaddr (16)			
sname (64)			
file (128)			
mcookie (4)			
options (variable)			

Figure 4 : Données DHCP

- op : type de message BOOTP (1 : requête client, 2 : réponse serveur)
- htype : type d'adresse hardware (1 : Ethernet)
- hlen : longueur de l'adresse hardware
- hops : nombre de relais DHCP
- xid : identifiant de la transaction (nombre aléatoire choisi par le client)
- secs : temps écoulé en secondes depuis que le client a commencé sa requête
- flags : flags
- ciaddr : adresse IP du client (0.0.0.0 si il n'en a pas)
- yiaddr : adresse IP proposée
- siaddr : adresse IP du prochain serveur (0.0.0.0 si il n'en a pas)
- giadddr : adresse IP du relais (0.0.0.0 si il n'en a pas)
- chaddr : adresse hardware du client
- sname : nom du serveur (tout à 0 si non donné)
- file : nom du fichier de boot (tout à 0 si non donné)
- mcookie : magic cookie (toujours 0x63825363)
- options : champs réservé pour les options

4.1.3.2 Format des options

Le passage de paramètre (exemple : adresse IP du serveur,…) se fait par l'intermédiaire d'option. Plusieurs options peuvent être envoyées dans le même message DHCP. Voici leur format :

code (1)	longueur (1)	données (variable)

Figure 5 : Format des options DHCP

- code : code identifiant l'option
- longueur : longueur des données de l'option
- données : données de l'option

Certaines d'entre elles n'ont ni longueur, ni donnée, comme par exemple l'option end (code 255) qui doit toujours finir la zone d'option.
Les options utilisées par la FPGA sont :

- DHCP Message Type (code 53)
- Requested IP Address (code 50)
- Server Identifier (code 54)
- Parameter Requst List (code 55)

4.1.3.3 Requêtes et message DHCP

Il existe plusieurs messages DHCP qui permettent, entre autre, de configurer et renouveler les adresses IP. L'identifiant du type de message se trouve dans l'option numéro 53. Ces messages peuvent être émis par le client ou par le serveur.

Nom	Identifiant	Description	Direction
DHCPDISCOVER	1	Localisation des serveurs DHCP disponibles	Client -> Serveur
DHCPOFFER	2	Réponse au message DHCPDISCOVER contenant les premiers paramètres	Serveur -> Client
DHCPREQUEST	3	Demande de réservation d'adresse IP ou prolongement du bail	Client -> Serveur
DHCPDECLINE	4	Annonce que l'adresse est déjà utilisée	Client -> Serveur
DHCPACK	5	Réponse contenant les paramètres pour le client	Serveur -> Client
DHCPNAK	6	Annonce que le bail est terminé ou une mauvaise configuration	Serveur -> Client Client -> Serveur
DHCPRELEASE	7	Libération de l'adresse IP	Client -> Serveur
DHCPINFORM	8	Demande de paramètres (adresse IP connu)	Client -> Serveur

Figure 6 : Type de message DHCP

4.1.3.4 Demande d'une nouvelle adresse IP

Le client envoie en broadcast un message DHCPDISCOVER contenant une liste des options qu'il désire recevoir. Le serveur répond avec un DHCPOFFER pour transmettre une adresse IP. Comme plusieurs serveurs peuvent répondre, le client prend la première réponse et transmet un message DHCPREQUEST qui contient l'adresse du serveur. Ce message est toujours expédié en broadcast. Si l'adresse IP peut être réservée, le serveur confirme avec un message DHCPACK, si non, un DHCPNAK est envoyé.
Lorsqu'un DHCPNAK est reçu, le client recommence depuis le début. Quand le client a reçu son adresse IP, il peut utiliser ARP pour prévenir un conflit d'adresse. Le protocole ARP n'étant pas encore implémenté, la FPGA ne l'utilise pas.
Le chronogramme de la Figure 7 montre une demande pour une nouvelle adresse.

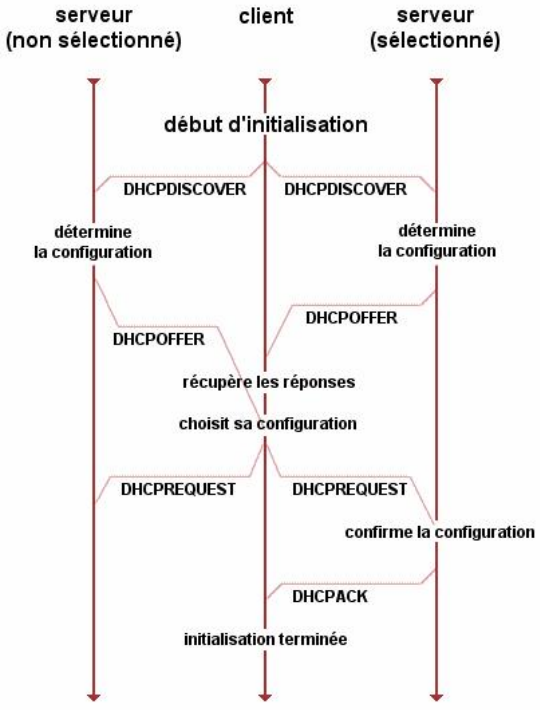


Figure 7 : Demande d'une adresse IP

4.1.3.5 Renouvèlement de bail

Lorsque le bail arrive à 50%, le client envoie du message DHCPREQUEST au serveur DHCP qui lui a attribué son adresse IP. Si aucun problème n'est détecté, le serveur réinitialise le bail et envoie un DHCPACK.
Si le serveur n'est pas joignable, le client attend 87.5% du bail et transmet un message DHCPREQUEST à tous les serveurs. Si le client ne reçoit pas de DHCPACK, mais un DHCPNAK ou pas du tout de message, le client fait une demande pour une nouvelle adresse IP.
La Figure 8 montre un renouvellement réussi d'une adresse IP.

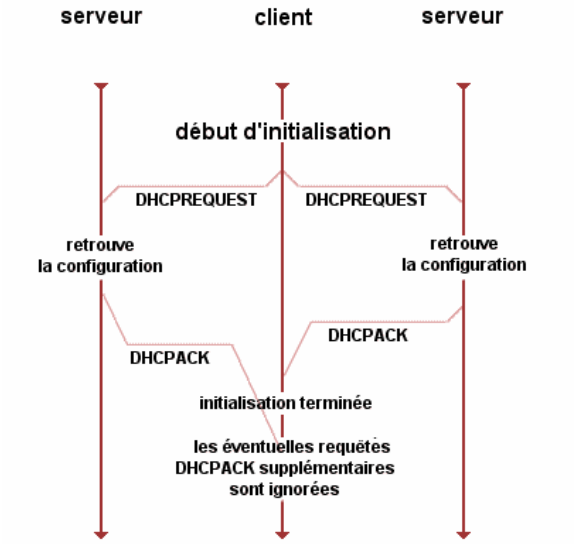


Figure 8 : Renouvellement d'une adresse IP

4.1.3.6 Libération de l'adresse IP

Avant de s'éteindre, le client devrait envoyer un message DHCPRELEASE au serveur DHCP qui lui a attribué son IP. Cette façon de faire n'est pas implémentée dans la FPGA, car elle ne devrait pas s'arrêter.
Si le serveur ne reçoit pas de message et que le bail arrive à son terme, l'adresse IP est libéré.
La Figure 9 montre la manière propre de libéré l'adresse IP.

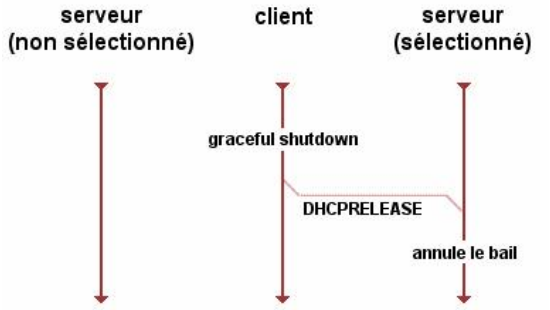


Figure 9 : Libération de l'adresse IP

4.2 MODULES

Les modules qui peuvent être raccordés à la carte FPGA permettent de commander des éclairages publics. Leurs registres sont maintenant sur 32 bits. La nouvelle documentation sur le protocole utilisé n'est pas encore fini, mais le rapport « SINE 12 Protocole de communication Master<->DSP V2.2 » fournit des renseignements. Les informations importantes sont expliquées dans ce chapitre.

4.2.1 Port RS232

Le port RS232 est utilisé pour envoyer ou recevoir les trames de communication. Les paramètres utilisés sont :

- Baudrate : 57.6 kB/s
- Parité : non
- Bits de données : 8
- Stop bit : 1
- Erreur flag : non

4.2.2 Paquet de communication

Les trames doivent avoir le format suivant (les chiffres indiquent la taille, en byte, des champs) :

En-tête (1)
Identifiant du paquet (1)
Commande (1)
Longueur (1)
Données (variable)
Somme de contrôle (1)

Figure 10 : Trames de communication avec les modules

- En-tête : Valeur 0xAA pour être valide
- Identifiant du paquet : Choisi par le PC
- Commande : Code de la commande
- Longueur : Longueur des données
- Données : Données de maximum 255 byte
- Somme de contrôle : Addition des bytes envoyés puis module 256

Tous les paquets reçus par le module sans erreur renvoient :

- L'identifiant du paquet
- La commande reçue
- Les données nécessaires

Tous les paquets reçus par le module avec des erreurs renvoient :

- L'identifiant du paquet (peut-être corrompu)
- La commande NACK (0x00)

4.2.3 Commandes

Les commandes utilisées lors de ce projet sont WRITE_MEM (0X03) et READ_MEM (0X04).

4.2.3.1 WRITE_MEM

Cette commande permet d'écrire dans la mémoire du module. Le format des données est :

Demande :

- Donné 0 : byte de poids faible de l'adresse
- Donné 1 : byte de poids fort de l'adresse
- Donné 2 : byte de poids faible de la valeur
- Donné 3 : byte de poids fort de la valeur

Réponse :

- Pas de donnée

4.2.3.2 READ_MEM

Cette commande permet de lire dans la mémoire du module. Le format des données est :

Demande :

- Donné 0 : byte de poids faible de l'adresse
- Donné 1 : byte de poids fort de l'adresse

Réponse :

- Donné 0 : byte de poids faible de la valeur
- Donné 1 : byte de poids fort de la valeur

4.3 PROGRAMME DE DÉMONSTRATION

4.3.1 Auto-détection

Un message est envoyé périodiquement pour mettre à jour les informations d'une carte. Cela permet d'avoir les informations désirées à jour automatiquement.
Pour faire cela, un byte est envoyé, périodiquement, sur tous les ports RS232 dont dispose la carte FPGA. Si un module est branché, un message d'erreur (NACK) est transmis à la carte. Ensuite, un paquet UDP est envoyé en broadcast sur le port 4321. Le port source de ce message est le port UDP pour communiquer avec les modules. Les informations se trouvant dans cette trame sont, dans l'ordre :

- l'adresse MAC (6 Bytes)
- l'adresse IP (4 Bytes)
- le numéro du port UDP pour communiquer avec les modules (2 Bytes)
- le nombre de port RS232 dont la carte dispose (1 Bytes)
- les numéros des ports RS232 qui ont un module actif (1 Byte par numéro)
- Symbole de fin (1 Byte)

Ces informations sont enregistrées sur le serveur dans un fichier XML.

4.3.2 Communication avec les modules

Il ne peut y avoir qu'un serveur qui envoie les trames de communication avec les modules. Une seule commande peut-être envoyée à la fois, donc le serveur doit attendre d'avoir reçu une réponse ou un temps de timeout avant d'en envoyer une autre.
Le protocole UDP est utilisé pour communiquer avec les modules. Le port de communication dépend de la carte FPGA et est donnée lors de l'auto-détection (fichier XML). Les données de ces paquets sont les suivantes :

- le numéro de port RS232 (1 Byte)
- les données à envoyer sur le port RS232 (variable)

Les données doivent correspondre au protocole de communication (§ 4.2.2).

5 PROGRAMMATION DE LA FPGA

5.1 STRUCTURE SUR LA CARTE FPGA

Quand un message transite sur le réseau Ethernet, la partie « Interface Ethernet » l'enregistre dans la mémoire de réception. Lorsqu'un message est présent dans la mémoire d'envoi, cette partie l'expédie sur le réseau Ethernet.
« Memory Manager » gère les données à transmettre et chaque bloc reçoit un pointeur sur le début des données qu'il traite. Par exemple l'adresse 0 dans le bloc « IPv4 » correspond à l'adresse de la première données du paquet IP.
Les autres blocs se trouvant dans la partie « Manageur de protocoles » se rapporte au protocole qu'ils décortiquent. Ceux de la partie réception stockent les informations utiles de l'en-tête du protocole et activent le bloc du protocole suivant. Les données nécessaires au traitement sont ensuite envoyées au bloc de la partie « Contrôleurs » qui les manipule. Les blocs de la partie d'envoi rajoutent les informations des en-têtes aux données reçues et, une fois que toutes les données du protocole sont rajoutées, active le bloc du prochain protocole. Lorsque tous les protocoles sont rajoutés, le paquet est stocké dans la mémoire d'envoi.
La partie « Contrôleurs » manipule les données reçues et transmet à la partie envoi de « Manageur de protocoles » les nouvelle données.
La partie « Contrôleur RS232 » s'occupe de sérialiser les données lors de l'envoi et de paralléliser lors de la réception des données sur les ports RS232.
La Figure 11 illustre les différentes parties de cette structure.

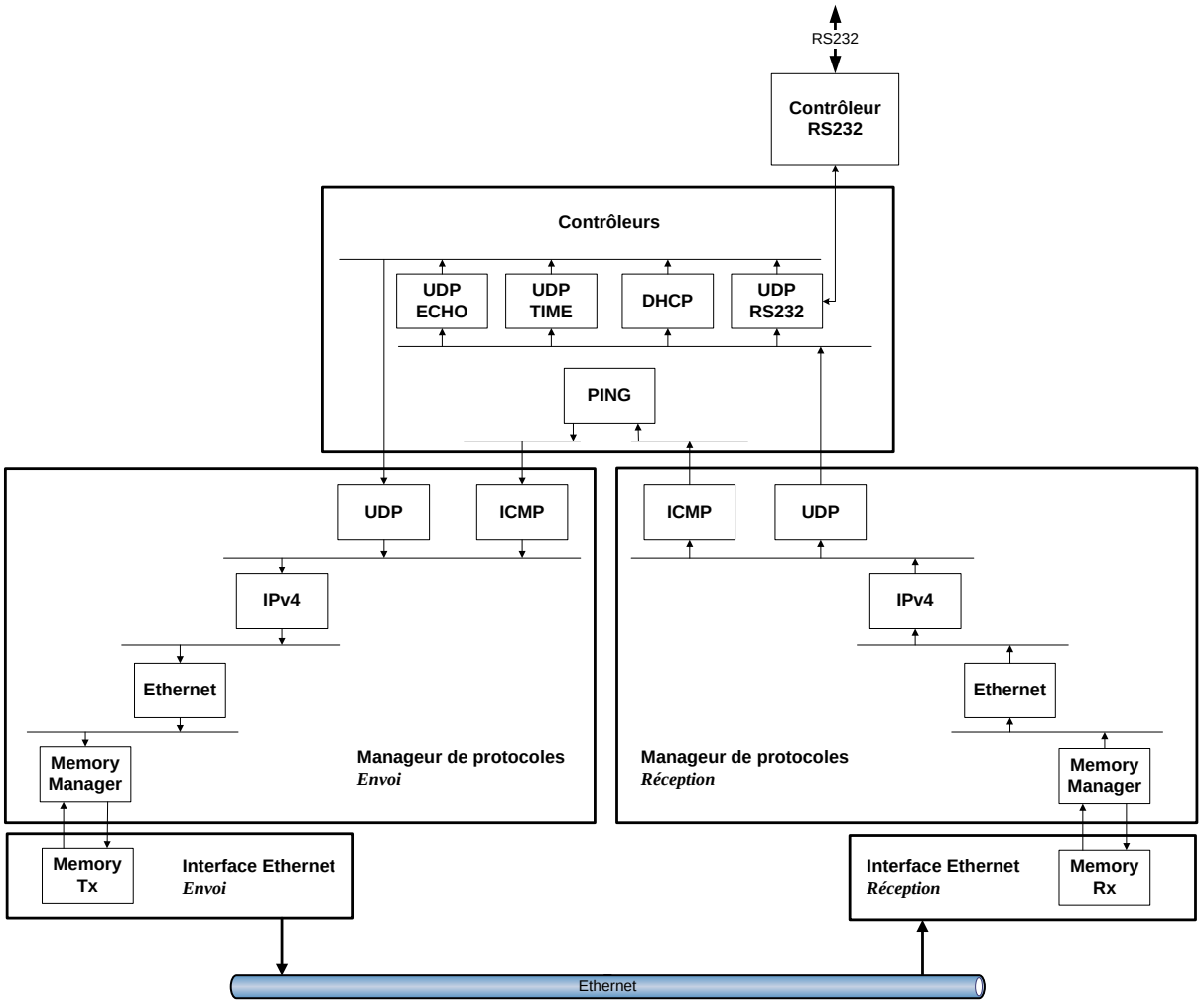


Figure 11 : Schématique de programmation

Seulement les parties importantes à la compréhension, implémentées ou modifiées lors du TD sont traitées dans ce rapport. Le design utilisé lors de ce projet est « fpga_ebs » dans la librairie « Board ».

5.2 HIÉRARCHISATION DES FICHIERS DE CONFIGURATION

- Config : ce fichier défini les différentes constantes de base.
- «Protocole»_definition : ces fichiers contiennent les constantes et les types de signaux des protocoles. Chaque fichier spécifie un protocole dont le nom est à place de «Protocole».
- Intercommunication_definition : les types de signaux des bus et leurs constantes se trouvent dans ce fichier.
- Controller_definition : dans ce fichier se trouve les types de signaux arrivant et partant de « controller_of_tests_top ». Il y a aussi les différentes constantes relatives à ses types.

5.3 CONTENU DES BUS

5.3.1 filter_info_s

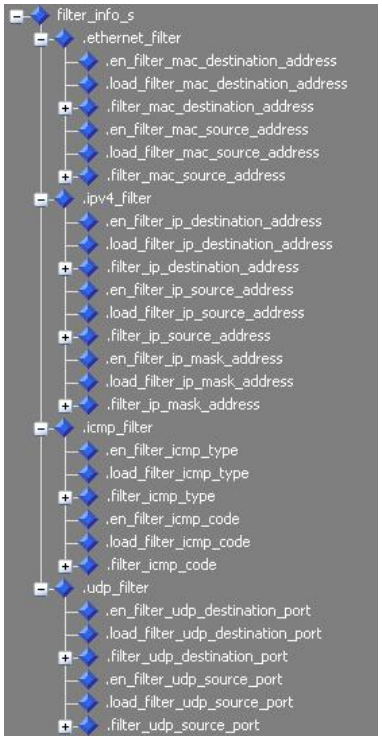


Figure 12 : Contenu de filter_info_s

Ce bus permet de configurer les filtres des protocoles. Les fitres ne sont utilisés que lors de la réception des paquets.
Chaque filtre a un signal d'activation (« en_... »), un signal de chargement de la valeur (« load_... ») et 8 bits de données («filter… »).
Les différents filtres sont :

Ethernet :	Adresse Ethernet destination
	Adresse Ethernet source
IPv4 :	Adresse IP destination
	Adresse IP source
	Masque de sous-réseau
ICMP :	Type
	Code
UDP :	Port destination
	Port source

Si le filtre du masque de sous-réseau d'IPv4 est désactivé, l'adresse de broadcast sera seulement 255.255.255.255, dans le cas contraire, le broadcast de sous réseau est aussi utilisé (adresse la plus haute du sous-réseau).
La Figure 12 montre la hiérarchisation de ce bus.

5.3.2 rx_info_s/tx_infos_s

Ces bus contiennent les informations des en-têtes des protocoles. « rx_info_s » donne les informations de réception et « tx_infos_s » celles d'émission. Les Figure 13 et Figure 14 montrent la hiérarchisation des informations dans ces bus.

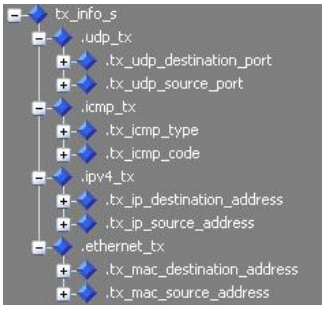


Figure 13 : Contenu de tx_info_s

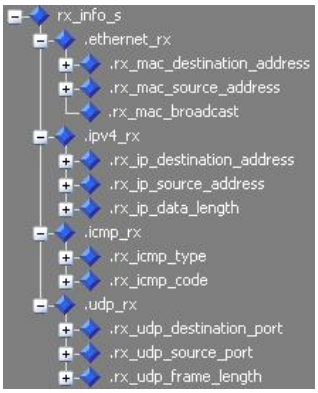


Figure 14 : Contenu de rx_info_s

Les informations contenues dans « rx_info_s » sont :

Ethernet :	Adresse Ethernet destination
	Adresse Ethernet source
	Broadcast Ethernet (si oui signal à 1)
IPv4 :	Adresse IP destination
	Adresse IP source
	Longueur des données contenues dans le paquet IPv4
ICMP :	Type
	Code
UDP :	Port destination
	Port source
	Longueur du paquet UDP

Les signaux « rx_mac_broadcast » et « rx_ip_data_length » ont été rajoutés.

5.3.3 array_of_bus_of_rx_inputs_s

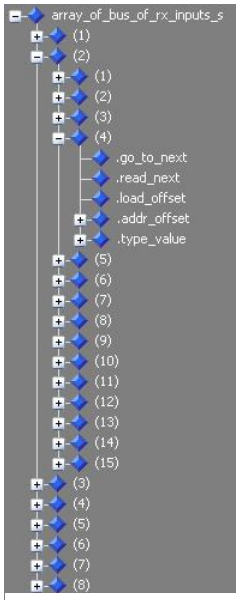


Figure 15 : Contenu de arry_of_bus_of_rx_inputs_s

La Figure 15 montre la hiérarchisation des informations dans ce tableau de bus. Chaque chiffre correspond à un protocole ou à un bloc de traitement. Leur signification se trouve dans le fichier « config ».
Le premier étage correspond à un protocole et le deuxième étage, au protocole suivant.
Le dernier étage se compose des signaux suivants :

- « go_to_next » : passe au paquet suivant
- « read_next » : demande le prochain byte
- « load_offset » : change l'offset
- « addr_offset » : offset pour arriver au byte de données (10 bits)
- « type_value » : valeur du protocole (32 bits)

5.3.4 array_of_bus_of_rx_outputs_s

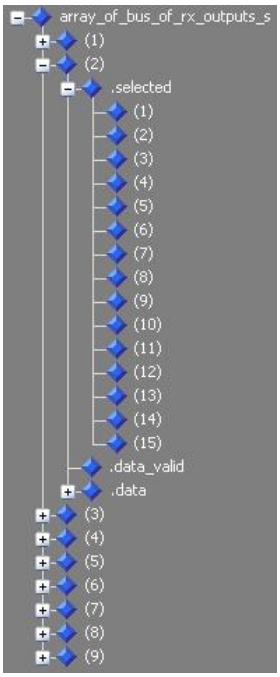


Figure 16 : Contenu de array_of_bus_of_rx_outputs_s

Chaque chiffre correspond à un protocole ou à un bloc de traitement. Leur signification se trouve dans le fichier « config ».

Le premier étage de ce bus correspond au bloc du protocole et contient les signaux suivant :

- « selected » : sélection du prochain protocole (active le bloc)
- « data_valide » : confirmation de la validité des données
- « data » : données (8 bits)

5.3.5 array_of_bus_of_tx_inputs_s

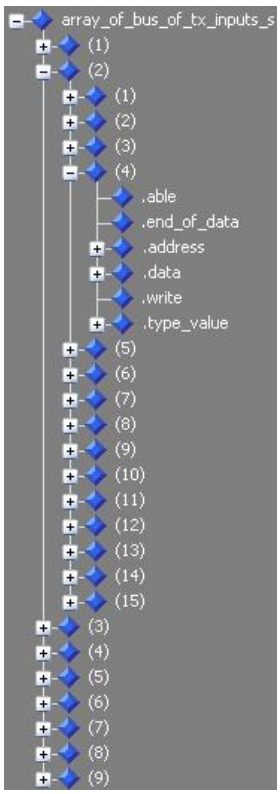


Figure 17 : Contenu de array_of_bus_of_tx_inputs_s

Chaque chiffre correspond à un protocole ou à un bloc de traitement. Leur signification se trouve dans le fichier « config ».
Tout comme pour «array_of_bus_of_rx_inputs_s », le premier étage correspond à un protocole et le deuxième étage, au protocole suivant.
Le dernier étage se compose des signaux suivants :

- « able » : activation du bloc
- « end_of_data »: fin de l'envoi des données
- « address » : offset pour arriver au byte de données (10 bits)
- « data » : données à transmettre (8 bits)
- « write » : confirmation d'écriture des données
- « type_value » : valeur du protocole (32 bits)

La Figure 17 montre la hiérarchisation des informations dans ce tableau de bus.

5.3.6 array_of_bus_of_tx_outputs_s

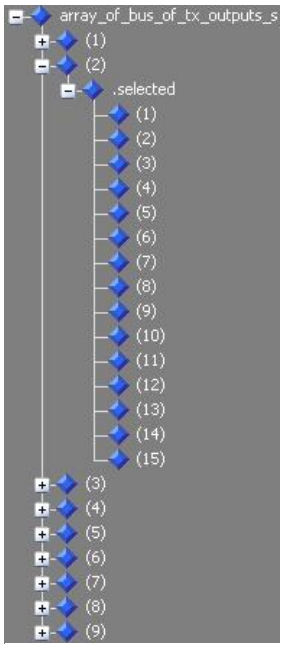


Figure 18 : Contenu de array_of_bus_of_tx_outputs_s

Chaque chiffre correspond à un protocole ou à un bloc de traitement. Leur signification se trouve dans le fichier « config ».

Le premier étage de ce bus correspond au bloc du protocole et contient les signaux suivant :

- « selected » : sélection du protocole suivant

5.3.7 bus_of_eth_to_serial_s

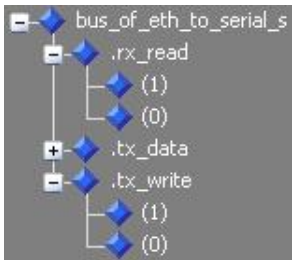


Figure 19 : Contenu de bus_of_eth_to_serial_s

Ce bus permet d'envoyer des informations aux ports série.
Ses signaux sont :

- « rx_read » : lecture de la donnée suivante
- « tx_data » : données à envoyer sur le port RS232 (8 bits)
- « tx_write » : écriture des données à envoyer

Les numéros des bits correspondent au numéro du port RS232.

5.3.8 bus_of_serial_to_eth_s

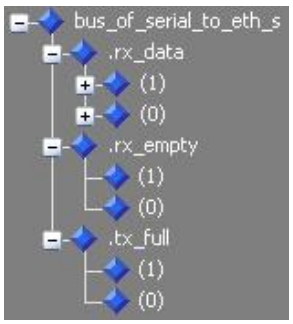


Figure 20 : Contenu de bus_of_serial_to_eth_s

Ce bus permet de recevoir des informations des ports série.
Les signaux sont :

- « rx_data » : données reçues
- « rx_empty » : mémoire de réception vide
- « tx_full » : mémoire d'envoi pleine

Les numéros des bits correspondent au numéro du port RS232.

5.4 BLOCS DE BASE

La structure de l'annexe 10.46 représente les blocs « interface », « manager_of_protocols_stack », « controller_of_tests_top » et « controller_rs232 » qui correspondent aux parties Interface Ethernet, Manageur de protocoles, Contrôleurs et Contrôleur RS232 du chapitre 5.1. Cette structure est contenue dans le bloc « toplevel » de la librairie « Trypanocontroller ».
Les blocs qui ont été rajoutés ou modifiés sont « manager_of_protocols_stack », « controller_of_tests_top » et « controller_rs232 ».

5.4.1 « manager_of_protocols_stack »

Le but et l'architecture de cette partie sont les mêmes que ceux de Manageur de protocoles expliqués au chapitre 5.1. Chaque protocole à un bloc de réception et un bloc d'envoi. Plusieurs blocs de protocoles ont été faits, mais tous ne sont pas implémentés.
Pour obtenir le byte de donnée que le bloc désire, celui-ci envoie l'offset pour arriver aux données. Le calcul de l'offset se fait depuis le commencement du protocole, car les blocs précédents rajoutent automatiquement l'offset correspondant à la taille de leur en-tête. Une fois les informations du protocole traitées, le bloc suivant est activé.

5.4.1.1 « icmp_rx »

Ce bloc s'occupe des paquets ICMP reçus. Il est composé des blocs suivants :

- « ipv4_checksum » : calcule le checksum (même que pour IPv4)
- « icmp_rx_registers » : traite les filtres et enregistre les informations d'ICMP (type et code)
- « type_registers » : sélectionne le bus du prochain protocole
- « pack_unpack_rx » : gère les données à envoyer/reçues au bloc de protocole suivant
- « address_registers » : calcule l'offset des données
- « icmp_rx_heart » : gère les autres blocs et envoie/reçoit les données du protocole précédant.

Lorsque ce bloc est activé, les informations de l'en-tête sont stockées, puis le checksum est contrôlé. Si la trame est incorrecte ou que le bloc de traitement n'existe pas, le signal « go_to_next » est mis à 1 pour passer au prochain paquet.
Dans le cas contraire, le bloc suivant est activé. Les données et signaux arrivant sur « icmp_rx » sont transmis sans modification aux autres blocs, à l'exception du bus « addr_offset » à qui l'adresse d'offset d'ICMP est rajoutée. Lorsque le traitement est fini, le bloc est désactivé et tous ses signaux sortant sont remis à leur état d'origine.

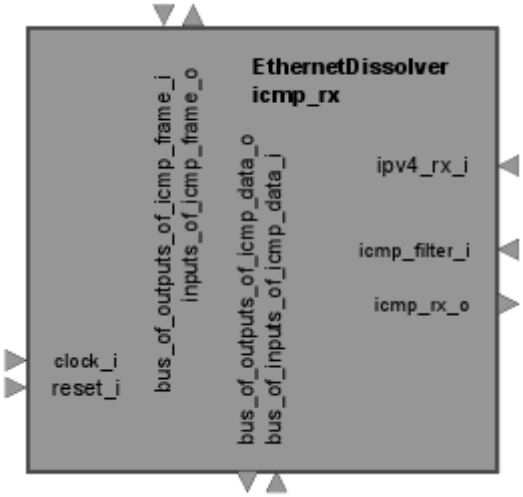


Figure 21 : Bloc icmp_rx

5.4.1.2 « icmp_tx »

Ce bloc rajoute les informations du protocole ICMP lors de l'envoi des paquets. Il est composé des blocs suivants :

- « ipv4_checksum » : calcule le checksum (même que pour IPv4)
- « pack_unpack_tx » : gère les données à envoyer/reçues au bloc de protocole précédent
- « address_registers » : calcule l'offset des données
- « icmp_rx_heart » : gère les autres blocs, envoie/reçoit les données du protocole suivant et rajoute les informations à l'en-tête du paquet à envoyer.

Lorsque ce bloc est activé, les données sont passées à la partie IPv4 et, quand il n'y a plus de données, l'en-tête ICMP est rajouté.

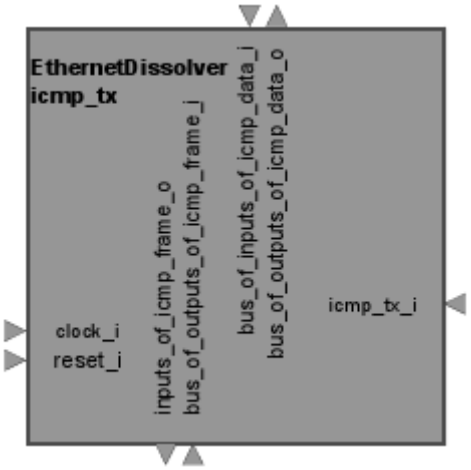


Figure 22 : Bloc icmp_tx

5.4.1.3 Blocs modifiés

Les blocs de réception Ethernet et IPv4 ont été modifiés pour pouvoir recevoir des broadcast. Désormais, ces blocs filtrent les messages en tenant compte des broadcast (de sous-réseau et universel). Le signal « rx_mac_broadcast » contenu dans « rx_info_s » (§ 5.3.2) est mis à 1 lorsque le message est un broadcast Ethernet.
Le bloc d'IPv4 envoie la longueur des données du paquet, ce qu'il ne faisait pas auparavant.

5.4.2 « controller_of_tests_top »

L'objectif et la structure de cette partie sont les mêmes que ceux de Contrôleurs expliqués au chapitre 5.1.
Le bloc « controller_echo » renvoie les données reçues sur le port 1234. Le bloc « controller_work_time » envoie, périodiquement, une trame UDP contenant le temps d'activation de la carte au PC ayant transmis un paquet UDP sur le port 4567. Ces deux parties ne sont pas expliquées plus en détail, car elles n'ont subit aucune modification lors de ce projet.
Les autres parties sont expliquées dans les pages suivantes.

5.4.2.1 « controller_ping »

Les paquets ICMP de demande d'écho sont traités ici. Cette partie est composée des blocs suivants :

- « controller_ping_heart » : gère les autres blocs et envoie/reçoit les données
- « controller_ping_tx_info_manager » : envoie/reçoit les informations des protocoles
- « address_registers » : compte le nombre de bytes de données envoyées

Le bourrage et les données sont transférés de la partie réception à la partie envoi. Ce bloc modifie les informations Ethernet, IPv4 et ICMP.

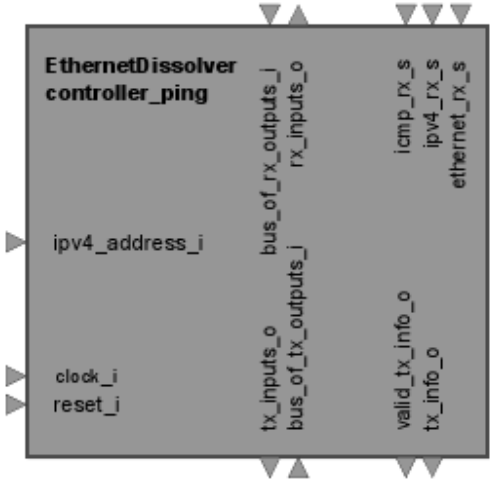


Figure 23 : Bloc controller_ping

5.4.2.2 « controller_dhcp »

Tout le protocole DHCP est géré par cette partie qui est composée des blocs suivants :

- « controller_dhcp_heart » : gère les autres blocs et envoie/reçoit des données d'UDP
- « controller_dhcp_write » : transmet les donnée des messages DHCP à la partie UDP d'envoie
- « controller_dhcp_timer » : compte les secondes
- « controller_dhcp_timer_lease » : minuteur qui, à la fin du bail, envoie un signal
- « controller_dhcp_filter » : met à jour les filtres et les active/désactive
- « controller_dhcp_tx_info_manager » : transmet les informations Ethernet, IPv4 et UDP
- « controller_dhcp_message_type » : contrôle le message et retourne le type DHCP
- « counter_random_number » : génère des nombres aléatoires
- « controller_dhcp_register » : stocke les informations DHCP
- « controller_dhcp_read » : lit les paquets DHCP
- « address_registers » : compte le nombre de bytes de données lues
- « counter » : compte la longueur des options

« controller_dhcp » s'occupe de faire les demandes d'adresse IP, ainsi que le renouvellement de leur bail. Il gère aussi les filtres IPv4 et transmet l'adresse IP aux autres parties.

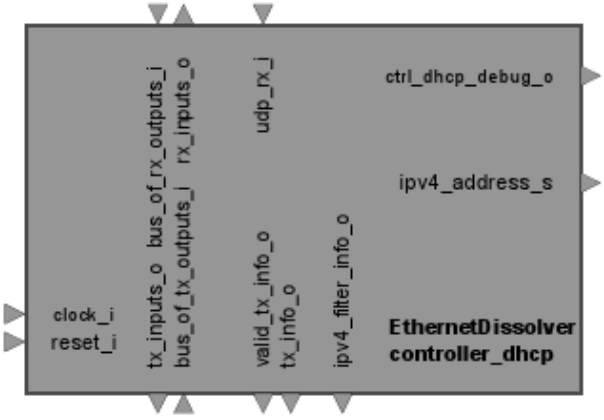


Figure 24 : Bloc controller_dhcp

5.4.2.3 « controller_udp_rs232 »

L'auto-détection et le transfert UDP-RS232 sont réalisés dans ce bloc composé des parties suivantes :

- « controller_udp_rs232_sender » : envoie les paquets vers le RS232
- « counter » : compte la longueur des paquets
- « controller_udp_rs232_register » : stocke et transmet les informations Ethernet, IP, UDP et le numéro de port RS232
- « controller_udp_rs232_receiver » : gère et stocke les données reçues par le RS232
- « controller_udp_rs232_write » : écrit les données stockées vers l'Ethernet
- « address_registers » : compte le nombre de bytes de données lues
- « controller_udp_rs232_autosensing » : manage l'envoi de paquet d'auto-détection

La partie « controller_udp_rs232_receiver » est composée des blocs suivants :

- « controller_udp_rs232_receiver_heart » : sélectionne le port RS232 et gère les autres blocs
- « controller_udp_rs232_receiver_read » : transmet les données de la partie RS232 à la partie de stockage
- « FIFO » : stocke les données
- « counter » : compte la longueur des paquets
- « counter » : compte le nombre de port RS232 qui ont répondu au message d'auto-détection

Le bloc « controller_udp_rs232_autosensing » est aussi divisé en sous-blocs qui sont :

- « controller_udp_rs232_autosensing_ip_heart » : manage les auto-détections
- « controller_udp_rs232_autosensing_ip_manager » : analyse les changements d'adresse IP
- « controller_timer_milisecond » : compte les millisecondes
- « controller_timer_second » : compte les secondes

Un message d'auto-détection est envoyé lorsque l'adresse IP de la carte change pour une adresse valide (tous sauf 0.0.0.0) ou lorsque le temps entre deux messages d'auto-détection est dépassé. Cette procédure commence seulement si aucun message n'est transféré et qu'aucune réponse n'est attendue, car aucune message ne peut être transféré durant la procédure d'auto-détection. Pour commencer, un byte (0X11) est transmit à tous les ports RS232 et les informations de la carte (adresse MAC, IP,…) sont enregistrées. Les réponses reçues (message NACK) sont lues, mais seulement le numéro du port est enregistré. Une fois que tous les ports ont répondus ou que le temps alloué à l'auto-détection est fini, les données enregistrées sont envoyées.
Lors d'un transfert, les données sont analysées et les informations Ethernet, IP et UDP sont enregistrées. Les données sont ensuite envoyées vers le port RS232 à qui elles sont destinées. Les mémoires de réception des ports série sont contrôlées alternativement, sauf lors de la réception de données. Quand cela arrive, le numéro du port et les données sont enregistrés. À la fin du transfert, les données sont envoyées avec les informations Ethernet, IP et UDP vers la partie d'envoi Ethernet et le contrôle des mémoires de réception des ports série reprend.

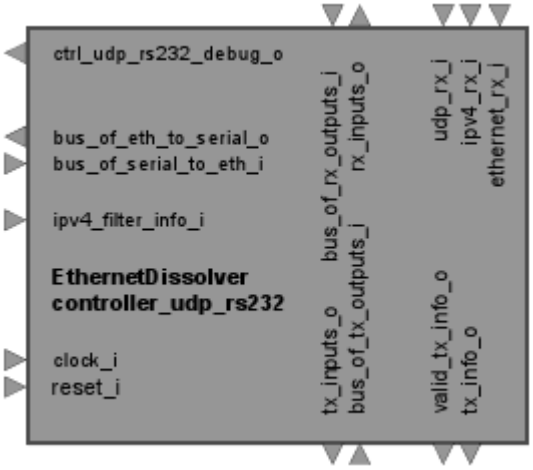


Figure 25 : Bloc controller_udp_rs232

5.4.2.4 *Blocs modifiés*

« controller_config_of_protocols_stack » a été modifié pour que les données des filtre transmises par DHCP soient prises en compte. La partie DHCP configure les filtres IPv4, les autres filtres sont configurés avec les données du fichier « controller_definition ».

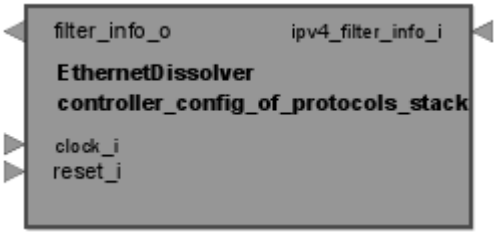


Figure 26 : Bloc controller_config_of_protocols_stack

5.4.3 « controller_rs232 »

La partie « controller_rs232 » permet d'envoyer et de recevoir des données sur le port RS232. Elle est composée d'un bloc « serialPortFIFO » par port RS232. Pour chaque port, il y a une mémoire d'envoi et une de réception.
Les entrées/sorties principales des blocs « serialPortFIFO » sont les suivantes :

- RxD : entrée série (réception)
- TxD : sortie série (envoi)
- rxData : données reçues depuis le port RS232 (8 bits)
- rxEmpty : mémoire de réception vide
- rxRd : passe à la donnée reçue suivante
- txData : données à envoyer vers le port RS232 (8 bits)
- txFull : mémoire d'envoi pleine
- txWr : enregistre la donnée à envoyer

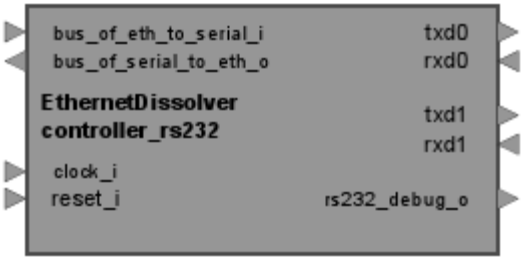


Figure 27 : Bloc controller_rs23

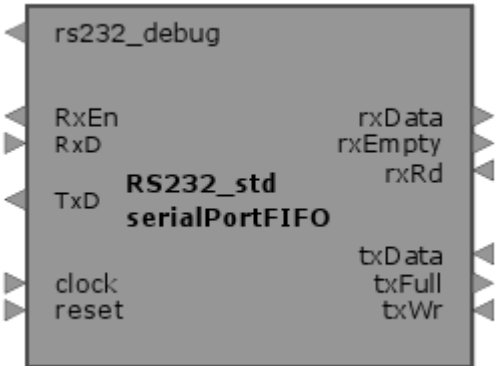


Figure 28 : Bloc serialPortFIFO

6 DÉVELOPPEMENT SOFTWARE SUR PC

6.1 PROGRAMME D'AUTO-DÉTECTION

Le code du fichier autosensing.pl effectue les actions suivantes :

- Initialise le socket
- Ouvre le fichier XML (DataBase.xml)
- Attend un message UDP sur le port 4321
- Ajoute les informations de la carte si elle n'est pas présente
- Modifie les informations de la carte si elle est présente
- Supprime les cartes si elles ne se sont plus authentifiées depuis un certain temps
- Met à jour la table ARP du serveur
- Enregistre les modifications du fichier XML

Le langage de programmation utilisé est le Perl.
Les paquets « libxml-simple-perl » et « libdate-manip-perl » ont été rajoutés à Debian, car ils sont utilisés dans le scripte Perl et ne sont pas présent par défaut.

6.2 STRUCTURE XML

La structure du fichier XML où sont stockées les informations de l'auto-détection est :

```
card
        addressMac      : Adresse Mac de la carte
        AddressIP       : Adresse IP de la carte
        udpPort         : Port UDP de communication
        numberRS232     : Nombre de port RS232 dont dispose la carte
        lastAccess      : Date et heure de la dernière mise à jour
    activePort
        number          : Numéro des ports actifs
```

Un exemple de fichier XML est présenté en annexe 10.2 et un fichier de description de structure (XSD) en annexe 10.3.

6.3 PROGRAMME DE DISCUSSION AVEC LES MODULES

Le code du fichier getInfos.pl a été fait pour tester le fonctionnement de la carte et du module. L'adresse IP, ainsi que les numéros de port UDP et RS232 ne peuvent être changés qu'en modifiant le code Perl.
Un message de lecteur de la mémoire (READ_MEM) est envoyé pour aller chercher la valeur de la fréquence d'oscillation et la réponse est affichée à l'écran.
L'utilisateur peut entrer une nouvelle valeur et un message d'écriture de la mémoire (WRITE_MEM) est envoyé. Après avoir reçu la confirmation d'écriture ou si, après un certain temps, aucune nouvelle valeur n'est entrée, le programme recommence.

7 TEST

7.1 SCHÉMATIQUE DU RÉSEAU DE TEST

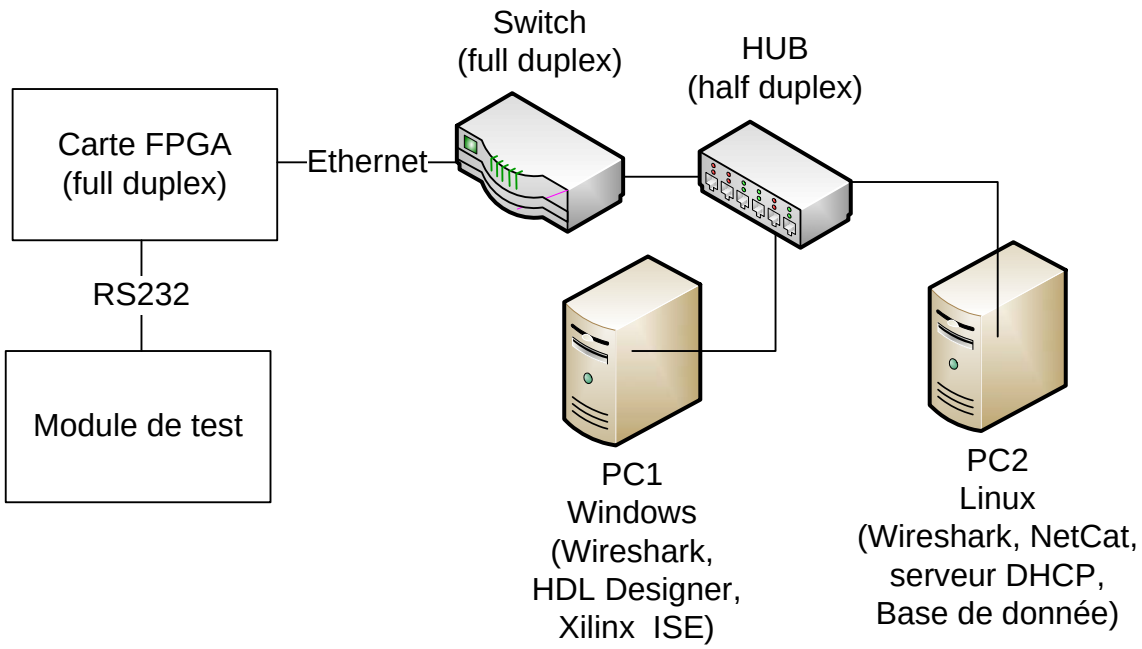


Figure 29 : Schématique du réseau de test

Le PC1 permet de programmer la FPGA et de voir les paquets circulants sur le réseau de test. Ce PC n'est raccordé au réseau de test que lors de quelques essais, car il envoie beaucoup de paquets non désirés.
Le PC2 permet d'envoyer les trames ICMP et UDP, de plus il fait office de serveur (DHCP et auto-détection). Les trames peuvent aussi être analysées sur ce PC, mais, pour Wireshark, certaines d'entre elles pourraient avoir des erreurs. En annexe 10.4 se trouve une aide pour configurer Debian.
Le hub transmet les données sur tous ses ports, ce qui permet une analyse sur les deux PC. La carte étant en full duplex, un switch doit être ajouté.

7.2 MODULE DE TEST

Ce module utilise le même protocole de communication que les contrôleurs d'éclairage, mais sa sortie (DAC3) est une tension sinusoïdale dont la fréquence peut être changée par des registres qui sont sur 32 bits.
Les signaux série de réception et d'émission peuvent être observés en branchant un oscilloscope sur les sorties 28 et 29.

7.3 TEST RÉALISÉ

Des tests ont été effectués sur toutes les parties réalisées et ont été documentés. Ces documents sont les annexes 10.5 à 10.42 et contiennent les informations suivantes :

- En-tête : Précise la partie testée
- Condition(s) : Conditions dans lesquelles ont été faits les tests
- Test(s) : Explications de ce qui est testé et comment cela est fait
- Résultat(s) : Résultats des tests

Les tests permettant de s'assurer du bon fonctionnement des parties de réception et d'envoi ICMP sont expliqués en annexes 10.5 et 10.8.
L'annexe 10.11 explique les tests exécutés sur la partie PING.
Les tests réalisés dans les annexes 10.15 et 10.22 permettent de s'assurer du bon fonctionnement des différentes parties du DHCP. Par la suite la carte FPGA à été branchée au réseau de l'école et aucun problème n'est survenu.
Les tests vérifiant les différentes parties du transfert UDP-RS232 et d'auto-détection sont décrits en annexe 10.26, 10.29, 10.34 et 10.36.
Le code Perl du programme d'auto-détection est vérifié lors des tests de l'annexe 10.43.

7.4 PROBLÈMES PERSISTANTS

Le premier problème est que, de temps à autre, le module met plus de 300ms à répondre (Annexe 10.44). Cela a pour effet d'envoyer un paquet au dernier ordinateur qui a communiqué avec la carte et certains messages d'auto-détection ne contiennent plus ce module.
Le deuxième problème est qu'il faut que la carte soit allumée avant les modules. Quand la carte FPGA s'allume ou s'éteint, le signal sur le port RS232 change et le module pense que c'est un message, ce qui peut bloquer le module.
Le programme en Perl ne crée pas le fichier XML lorsque celui-ci est inexistant. De plus, le fichier XML n'est mis à jour que lors de la réception de message d'auto-détection.

8 CONCLUSION

8.1 ETAT DES LIEUX

L'équipement nécessaire à la mise en place de l'environnement de test a été trouvé et branché. Le système d'exploitation Debian, ainsi que les programmes NetCat et WireShark ont été installés et configurés sur le PC faisant office de serveur. Les différents paquets nécessaires à la programmation en Perl ont été rajoutés et le service DHCP a été mis en fonction.
La programmation des blocs du protocole ICMP et PING a été faite. Les tests ont prouvés que leur fonctionnement correspond à ce que fait un périphérique Ethernet.
La carte FPGA peut désormais recevoir ses configurations réseau dynamiquement grâce à la partie DHCP. Les tests effectués montrent que son fonctionnement est équivalant à ce qui est utilisé habituellement.
Les différentes parties de l'application de démonstration se trouvant dans la FPGA ont été programmées. Les tests effectués sur celles-ci montrent que leur fonctionnement correspond aux demandes du cahier des charges.
L'application qui enregistre les données des messages d'auto-détection a été implémentée et testée. Malgré quelques inconvénients (§ 7.4), elle peut être utilisée facilement.
Une application permettant de discuter avec les modules a été faite, mais, étant conçue pour les tests, il faut l'améliorer, voir en créer une nouvelle, pour une utilisation plus concrète.

8.2 AMÉLIORATIONS

L'implémentation du protocole ARP permettrait de diminuer la taille des paquets d'auto-détection. De plus la table ARP serait mise à jour automatiquement, ce qui réduirait le travail effectué par les applications.
La partie gérant les transferts UDP-RS232 peut être améliorée pour permettre d'envoyer plusieurs messages sur des ports RS232 différents. Cela permettrait de ne plus forcément avoir besoin d'attendre une réponse pour envoyer la commande suivante.
L'application permettant de discuter avec les modules doit être améliorée. L'utilisateur devrait pouvoir choisir la carte, le port et les paramètres qu'il souhaite examiner ou modifier. Une page Web faciliterait l'utilisation.
Une authentification des modules pourrait être mise en place pour permettre à l'utilisateur de les identifier plus facilement.

Sion, le 12 juillet 2010

Praplan Bastien

9 BIBLIOGRAPHIE

Aide Linux. (s.d.). Récupéré sur http://doc.ubuntu-fr.org/

Configuration DHCP sous Linux. (s.d.). Récupéré sur http://www.freebsd.org/doc/fr/books/handbook/network-dhcp.html

Débuter en Perl. (s.d.). Récupéré sur http://perso.univ-rennes1.fr/francois.dagorn/perl/perl.html

Documentation Perl. (s.d.). Récupéré sur http://perldoc.perl.org/

Documentation sur Date::Manip. (s.d.). Récupéré sur http://search.cpan.org/~sbeck/Date-Manip-6.11/lib/Date/Manip.pod

Documentation sur XML::Simple. (s.d.). Récupéré sur http://search.cpan.org/~grantm/XML-Simple-2.18/lib/XML/Simple.pm

Paquets Debian. (s.d.). Récupéré sur http://www.debian.org/distrib/packages

Ping. (s.d.). Récupéré sur http://en.wikipedia.org/wiki/Ping

Protocole DHCP. (s.d.). Récupéré sur http://membres.multimania.fr/dlassauge/mcse/dhcp/dhcp.htm

Protocole DHCP. (s.d.). Récupéré sur http://www.frameip.com/dhcp/

Protocole DHCP. (s.d.). Récupéré sur Wikipedia en: http://en.wikipedia.org/wiki/DHCP

Protocole Ethernet. (s.d.). Récupéré sur http://fr.wikipedia.org/wiki/Ethernet

Protocole ICMP. (s.d.). Récupéré sur http://www.frameip.com/entete-icmp/#3.2_-_Checksum

Protocole ICMP. (s.d.). Récupéré sur http://fr.wikipedia.org/wiki/Internet_Control_Message_Protocol

Protocole IPv4. (s.d.). Récupéré sur http://fr.wikipedia.org/wiki/IPv4

Protocole UDP. (s.d.). Récupéré sur http://fr.wikipedia.org/wiki/User_Datagram_Protocol

(2006). *SINE 12 Protocole de communication Master<->DSP V2.2.*

10 ANNEXES

10.1 ICMP : TYPE ET CODE

Type 0 (réponse echo)
Type : 0
Code : 0
Message : réponse d'ECHO (echo-reply)

Type 3 (destinataire inaccessible)
Type : 3
Code : 0 à 15
Message : destinataire inaccessible. Le code dépend de la cause du problème.

Type 4 (extinction de la source)
Type : 4
Code : 0
Message : extinction de la source (*source quench*)

Type 5 (redirection)
Type : 5
Code : 0 à 3
Message : redirection pour

- un hôte
- un hôte et un service
- un réseau
- un réseau et un service

Type 8 (echo)
Type : 8
Code : 0
Message : demande d'ECHO (echo-request)

Type 11 (temps dépassé)
Type : 11
Code : 0 ou 1
Message : temps dépassé ou temps de ré-assemblage des fragments d'un datagramme dépassé

Type 12 (en-tête erroné)
Type : 12
Code : 0
Message : en-tête erroné

Type 13 (demande heure)
Type : 13
Code : 0
Message : timestamp request

Type 14 (réponse heure)
Type : 14
Code : 0
Message : timestamp reply

Type 15 (demande adresse IP)
Type : 15
Code : 0
Message : demande d'adresse réseau

Type 16 (réponse adresse IP)
Type : 16
Code : 0
Message : réponse d'adresse réseau

Type 17 (demande masque sous-réseau)
Type : 17
Code : 0
Message : demande de masque de sous-réseau

Type 18 (réponse masque sous-réseau)
Type : 18
Code : 0
Message : réponse de masque de sous-réseau

10.2 EXEMPLE DE FICHIER XML

```
<opt>
      <card   addressIP="996D0518"
              addressMAC="000017FA0101"
              lastAccess="Thu Jun 24 12:50:53 2010"
              numberRS232="02"
              udpPort="1DE6" >
          <activePort number="1"/>
      </card>
      <card   addressIP="996D0519"
              addressMAC="000017FA0102"
              lastAccess="Thu Jun 24 12:50:36 2010"
              numberRS232="02"
              udpPort="1DE6"/>
      <card   addressIP="996D0520"
              addressMAC="000017FA0103"
              lastAccess="Thu Jun 24 12:51:10 2010"
              numberRS232="02"
              udpPort="1DE6" >
          <activePort number="0"/>
          <activePort number="1"/>
      </card>
</opt>
```

10.3 FICHIER XSD

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

      <xsd:element name="opt">
          <xsd:complexType>
              <xsd:sequence>
                  <xsd:element maxOccurs="unbounded" ref="card"/>
              </xsd:sequence>
          </xsd:complexType>
      </xsd:element>

      <xsd:element name="card">
          <xsd:complexType>
              <xsd:sequence>
                  <xsd:element minOccurs="0" maxOccurs="unbounded" ref="activePort"/>
              </xsd:sequence>
              <xsd:attribute name="addressMAC" use="required"/>
              <xsd:attribute name="addressIP" use="required"/>
              <xsd:attribute name="udpPort" use="required"/>
              <xsd:attribute name="lastAccess" use="required"/>
              <xsd:attribute name="numberRS232" use="required"/>
          </xsd:complexType>
      </xsd:element>

      <xsd:element name="activePort">
          <xsd:complexType>
              <xsd:attribute name="number" use="required"/>
          </xsd:complexType>
      </xsd:element>
</xsd:schema>
```

10.4 DEBIAN

10.4.1 Modification de l'adresse IP

System -> Administration -> Network
Onglet Connections -> sélectionné la connexion -> properties
Mettre les informations requises
Dans un terminal en mode root, entrer la commande « /etc/init.d/networking restart » pour être sûr d'avoir la nouvelle configuration réseau.

10.4.2 Serveur DHCP

Installer le paquet dhcp3-client (commande apt-get install) en root.
Modifier le fichier « /etc/dhcp3/dhcpd.conf » avec la configuration désirée (exemple au chapitre 10.4.3)
Démarrer le service avec la commande « /etc/init.d/dhcp3-server start »

10.4.3 Fichier dhcpd.conf

```
#
# Sample configuration file for ISC dhcpd for Debian
#
# $Id: dhcpd.conf,v 1.1.1.1 2002/05/21 00:07:44 peloy Exp $
#

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
option domain-name "ethfpga.org";
default-lease-time 180;
max-lease-time 600;

# This is a very basic subnet declaration.

subnet 153.109.5.0 netmask 255.255.255.0 {
  range 153.109.5.20 153.109.5.100;
  option subnet-mask 255.255.255.0;
  option routers 153.109.5.10;
}
```

10.4.4 Commandes utiles

Les commandes suivantes sont à exécuter en mode texte (terminal) en root.

./[nom du fichier]	: exécute le fichier (si exécutable)
/etc/init.d/dhcp3-server start	: démarre le serveur DHCP
/etc/init.d/dhcp3-server stop	: stoppe le serveur DHCP
/etc/init.d/networking restart	: redémarre la configuration réseau
apt-get install [nom du paquet]	: installe le paquet et ses dépendances
arp –a	: affiche toute la table de routage
arp –d [adresse IP]	: supprime l'adresse de la table de routage
arp –s [adresse IP] [adresse MAC]	: nouvelle entrée dans la table de routage
chmod 775 [nom du fichier]	: rend un fichier exécutable pour l'utilisateur et son groupe
ifconfig	: affiche les informations réseau
nano [fichier avec chemin]	: éditeur de fichier en mode commande
nc –u [adresse IP] [port]	: connexion UDP avec netcat
ping [adresse IP]	: fait une demande ping à l'adresse IP

10.5 TEST DE RÉCEPTION DE PAQUETS ICMP

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	icmp_rx

1) Condition(s)

Ces tests sont effectués sur ModelSim. La structure de ce test est « controller_of_tests_top_tb » et le fichier Wireshark utilisé est test_PING_with_UPD.cap.
La partie de traitement du PING ne fait, pour l'instant, que mettre à 1 ce signal.

2) Test(s)

a) Ce premier test permet de vérifier que la longueur de données envoyées par le bloc IPv4 correspond à la réalité. La taille d'un paquet ICMP est comparée à la valeur envoyée par la partie IPv4 (bus « rx_ip_data_length »).

b) Contrôle que les bus contenant les informations sur le type (« rx_icmp_type ») et le code (« rx_icmp_code ») des messages ICMP changent.

c) Examine que le bloc « icmp_rx » traite seulement le paquet ICMP. Les données doivent changer sur le bus ICMP (valeur 5 de « array_of_bus_of_rx_inputs_s ») seulement quand des paquets ICMP sont traités.

d) Inspecte que les données sont envoyées au bloc suivant. Les signaux « go to next » du bus PING (valeur 13 de « array_of_bus_of_rx_inputs_s ») et ICMP doivent passer à 1 lors de la fin de la lecture.

3) Résultat(s)

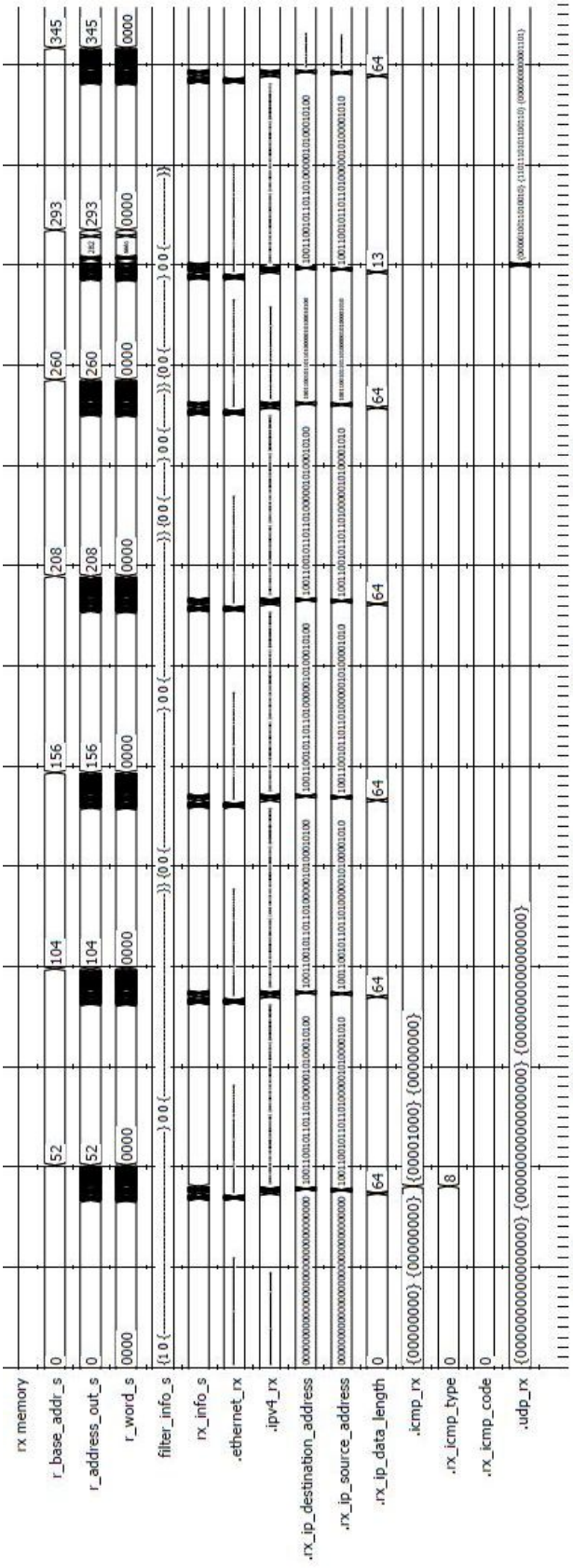
a) Les paquets ICMP sont composés de 64 bytes et le premier paquet UDP de 13 bytes, ce qui correspond aux valeurs se trouvant sur le bus. (Graphique en annexe 10.6)

b) Les valeurs étant sur les bus sont 8 pour le type et 0 pour le code et cela correspond bien avec les informations d'une commande PING. (Graphique en annexe 10.6)

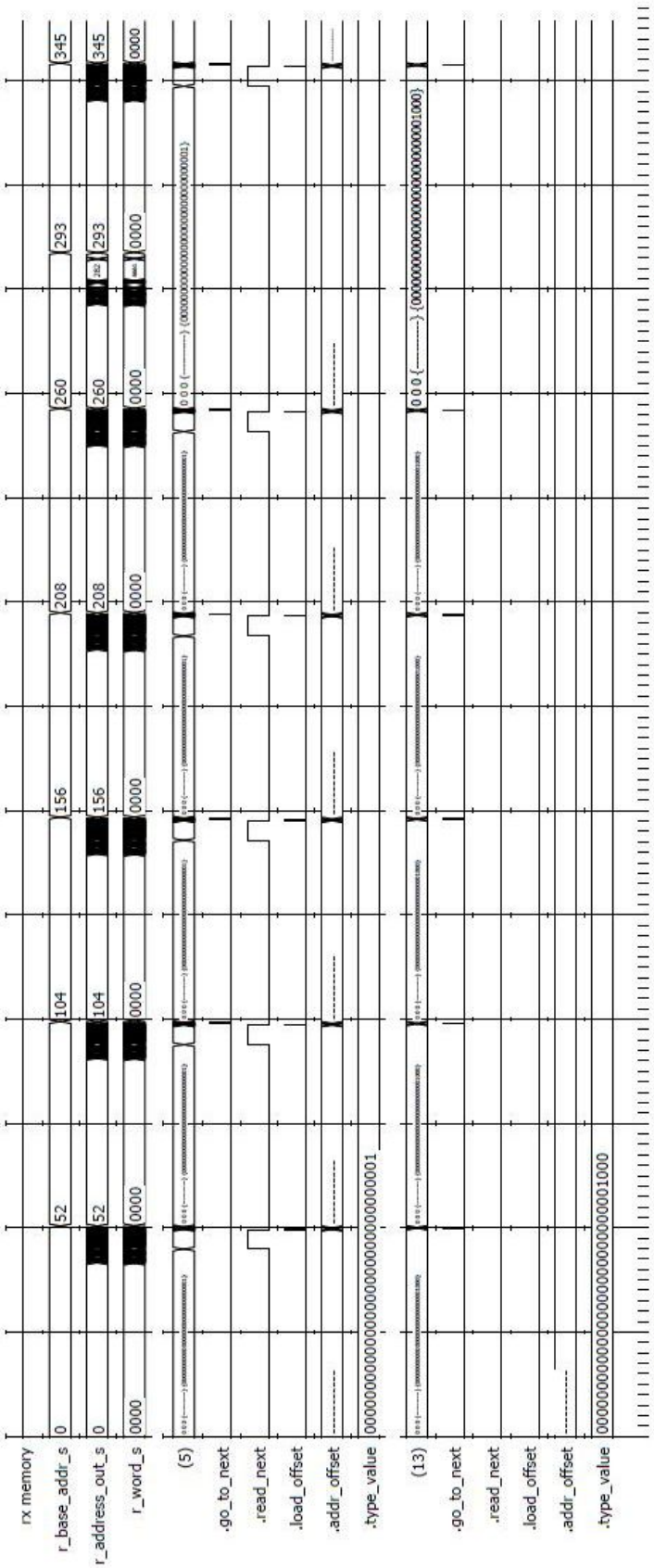
c) Les signaux du bus ICMP ne changent que quand les paquets ICMP sont traités. (Graphique en annexe 10.7)

d) Le bloc de traitement du PING est activé et les données sont répétées. (Graphique en annexe 10.7)

10.6 RÉCEPTION ICMP (EN-TÊTE)



10.7 RÉCEPTION ICMP (DONNÉES)



10.8 TEST D'ENVOI DE PAQUETS ICMP

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	icmp_tx

1) Condition(s)

Ces tests sont effectués sur ModelSim. La structure de ce test est « controller_of_tests_top_tb » et le fichier Wireshark utilisé est test_PING_with_UPD.cap.

2) Test(s)

a) Contrôle que tous les états de la machine d'états sont effectués.

b) Vérifie que les données sont transmises à la partie IPv4.

c) S'assure que l'en-tête ICMP est rajouté aux données.

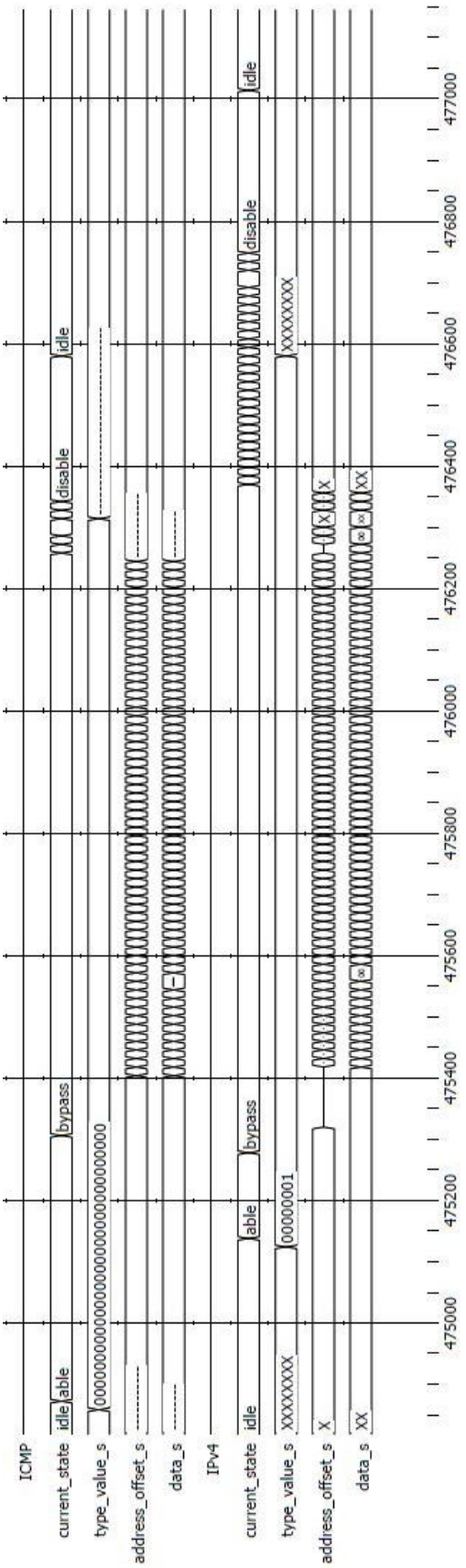
3) Résultat(s)

a) La machine d'états effectue tous ses états et retourne à l'état IDLE pendant que le bloc IPv4 traite ses données. (Graphique en annexe 10.9)

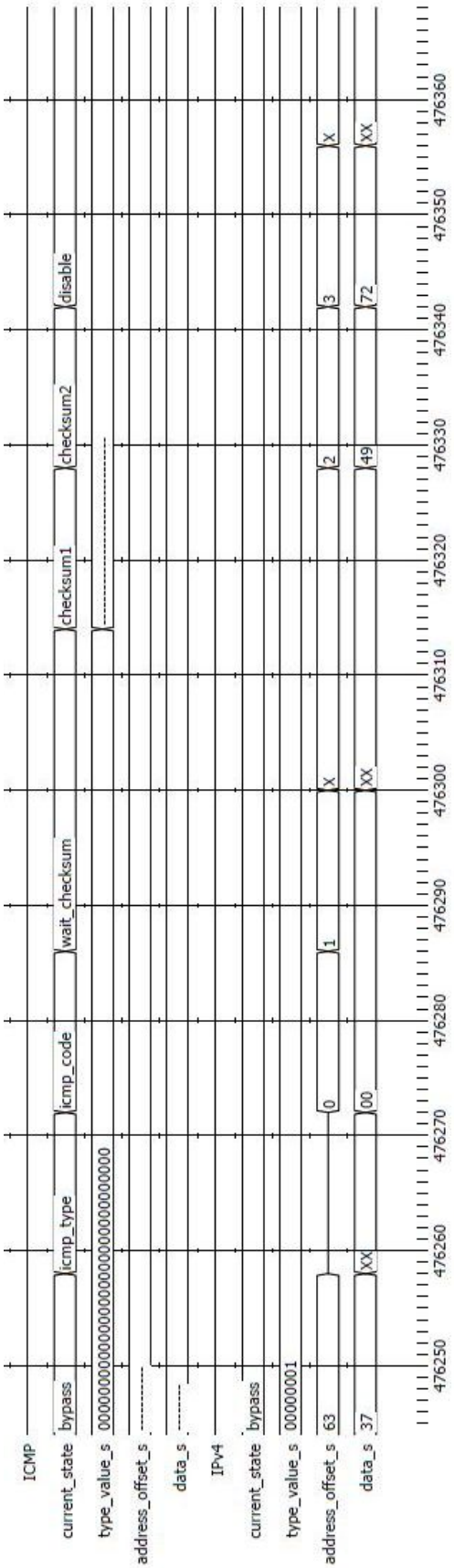
b) Les données sont bien transmises plus loin. (Graphique en annexe 10.9)

c) Les informations de l'en-tête ICMP sont transmises correctement. (Graphique en annexe 10.10)

10.9 ENVOI ICMP (ETATS ET DONNÉES)



10.10 ENVOI ICMP (EN-TÊTE)



10.11 TEST PING

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_ping, icmp_rx, icmp_tx

1) Condition(s)

Les premiers tests sont effectués sur ModelSim. La structure de ce test est « controller_of_tests_top_tb » et le fichier Wireshark utilisé est test_PING_with_UPD.cap.
Pour le dernier test, le programme est chargé dans la FPGA (mémoire flash) pour permettre de tester en réel.

2) Test(s)

a) S'assure que la machine d'état de PING retourne à l'état IDLE après que le bloc « icmp_tx » ai fini de traiter les données.

b) Contrôle que les données (bus « data_s » pour ICMP et IPv4) sont les mêmes que lors de la réception et qu'elles sont envoyées avec le bon offset (bus « address_offset_s » pour ICMP et IPv4). Les données sont comparées avec celles du fichier Wireshark.

c) Une commande PING est envoyée depuis le PC vers la carte. Wireshark analyse les paquets voyageant sur le réseau Ethernet.

3) Résultat(s)

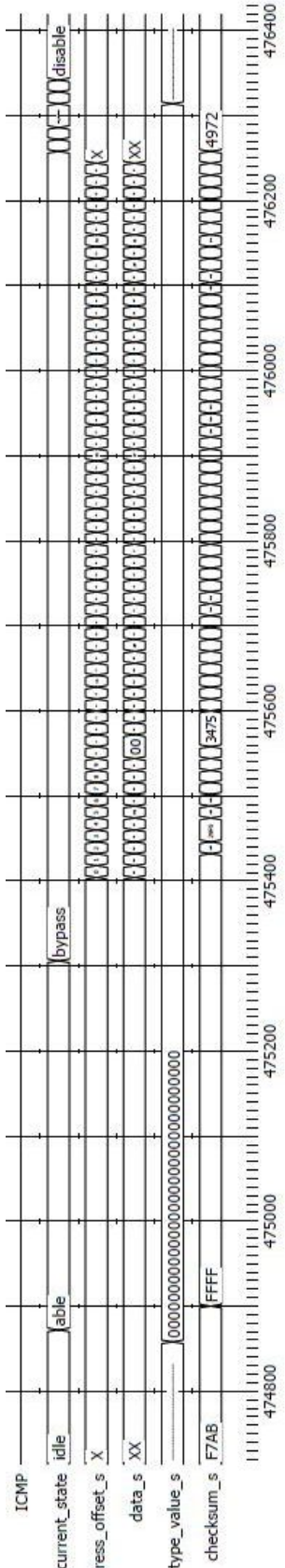
a) Le bloc PING retourne bien à son état initial (IDLE) quand il faut. (Graphique en annexe 10.12 a)

b) Lors du premier test, les données avaient un offset incorrect (+4). Après avoir changé le code de la machine d'état, les données sont placées au bon endroit dans la mémoire. (Graphique en annexe 10.12 b)

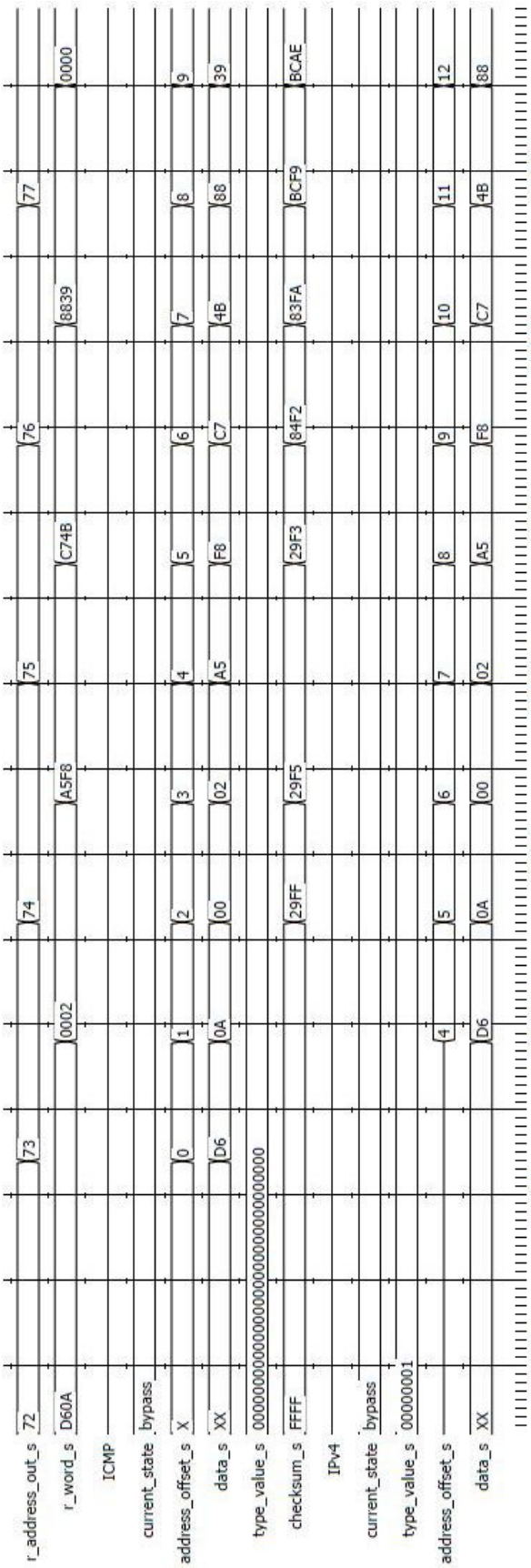
c) Les trames retournées contiennent bien les mêmes données que lors de l'envoi. L'en-tête ICMP est mis à jour correctement (Annexe 10.13 a et b). Le PC, quand-à-lui, traite les réponses correctement (Annexe 10.14).

10.12 PING (MODELSIM)

a)



b)



10.13 PING (WIRESHARK)

a)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	153.109.5.10	153.109.5.20	ICMP	Echo (ping) request
2	0.000120	153.109.5.20	153.109.5.10	ICMP	Echo (ping) reply
3	0.998905	153.109.5.10	153.109.5.20	ICMP	Echo (ping) request
4	0.999017	153.109.5.20	153.109.5.10	ICMP	Echo (ping) reply

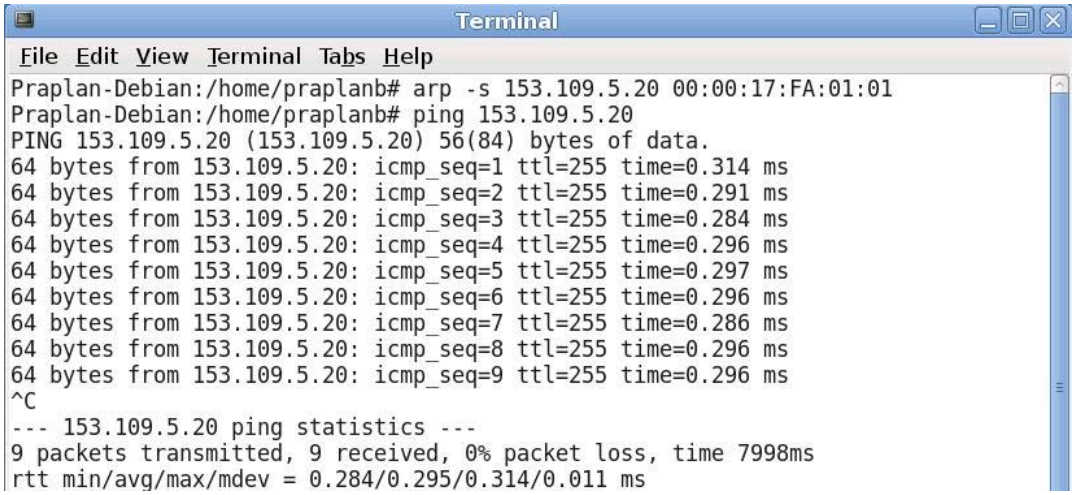
```
⊞ Frame 1 (98 bytes on wire, 98 bytes captured)
⊞ Ethernet II, Src: 3Com_21:6d:a8 (00:04:76:21:6d:a8), Dst: Tekelec_fa:01:01 (00:00:17:fa:01:01)
⊞ Internet Protocol, Src: 153.109.5.10 (153.109.5.10), Dst: 153.109.5.20 (153.109.5.20)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0 ()
    Checksum: 0xc753 [correct]
    Identifier: 0xd10a
    Sequence number: 1 (0x0001)
  ⊟ Data (56 bytes)
      Data: 3C37D14B5E1A090008090A0B0C0D0E0F1011121314151617...
      [Length: 56]
```

b)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	153.109.5.10	153.109.5.20	ICMP	Echo (ping) request
2	0.000120	153.109.5.20	153.109.5.10	ICMP	Echo (ping) reply
3	0.998905	153.109.5.10	153.109.5.20	ICMP	Echo (ping) request
4	0.999017	153.109.5.20	153.109.5.10	ICMP	Echo (ping) reply

```
⊞ Frame 2 (98 bytes on wire, 98 bytes captured)
⊞ Ethernet II, Src: Tekelec_fa:01:01 (00:00:17:fa:01:01), Dst: 3Com_21:6d:a8 (00:04:76:21:6d:a8)
⊞ Internet Protocol, Src: 153.109.5.20 (153.109.5.20), Dst: 153.109.5.10 (153.109.5.10)
⊟ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0 ()
    Checksum: 0xcf53 [correct]
    Identifier: 0xd10a
    Sequence number: 1 (0x0001)
  ⊟ Data (56 bytes)
      Data: 3C37D14B5E1A090008090A0B0C0D0E0F1011121314151617...
      [Length: 56]
```

10.14 PING (TERMINAL)



10.15 TEST 1ÈRE CONNEXION DHCP

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_dhcp

1) Condition(s)

Les premiers tests (a à e) sur DHCP sont effectués sur ModelSim. La structure de ce test est « controller_of_tests_top_tb » et le fichier Wireshark utilisé est test_dhcp.cap. Les compteurs pour le temps (« controller_dhcp_timer » et « controller_dhcp_timer_lease ») ont été divisés par 1000000 pour diminuer le temps de la simulation. Le nombre aléatoire est fixe (0x07de2905).
Les tests f à h sont faits en analysant les paquets avec Wireshark. Les temps sont, au démarrage, d'une seconde et, entre chaque nouvelle tentative de connexion, d'une minute.

2) Test(s)

a) Examine qu'après le reset, les filtres d'IPv4 soient mis à leur valeur par défaut (0.0.0.0 pour l'adresse et 255.255.255.255 pour le masque) et activés.

b) Vérifie qu'un paquet DISCOVER soit envoyé au démarrage après que « controller_dhcp_timer » est fini son comptage.

c) Contrôle que seuls les paquets ayant le bon xid soient traités et que les registres soient mis à jour après la réception d'un bon OFFER. Les quatre premières trames du fichier test_dhcp.cap n'ont pas le bon xid, tandis que celui des quatre suivants correspond à la valeur fixée. Cela permet aussi de s'assurer que les filtres broadcast fonctionnent.

d) Vérifie qu'après une offre correcte ai été reçue, un paquet REQUEST soit envoyé.

e) S'assure qu'après la réception d'une ACK, les registres et les filtres IPv4 soient à jour.

f) Teste si les messages DISCOVER sont réémis et chronomètre le temps entre chaque envoi de messages. Ce temps étant bien sûr qu'indicatif. Pour faire cela, le serveur DHCP n'est allumé que lorsque le premier message DHCP a été émis.

g) Analyse si les messages sont reçus et envoyés correctement. Le serveur est allumé et la carte FPGA branchée au réseau.

h) Contrôle que le tri des paquets se fasse correctement. Pour ce faire, l'adresse MAC de la carte FPGA est enregistrée deux fois dans la table de routage avec des adresses IP différentes. Des commandes PING sont, ensuite, faites avec les deux adresses IP.

3) Résultat(s)

a) Les filtres sont bien initialisés aux bonnes valeurs et sont activés. L'annexe 10.16 permet aussi de voir que le timer est initialisé et enclenché.

b) Les bus d'adresse et de données de « tx_inputs_o » changent lorsque la machine d'état est sur WRITE_DISCOVER, ce qui prouve que des données sont envoyées (Annexe 10.17 a). L'annexe 10.17 b est un zoom sur les données envoyées, celui-ci permet de voir que le type du message est bien un DISCOVER (adresse 240 à 243). Les données du magic cooki (adresse 236 à 239) sont aussi correctes.

c) Le graphique de l'annexe 10.18 montre que les messages OFFER et ACK ne sont traités que la 2ème fois (après 620us) et qu'après leur réception les registres sont mis à jour. A chaque fois que le timer est fini (signal « time_end_s » à 1), un message DISCOVER est envoyé.

d) Lorsqu'un message OFFER correct est reçu, un message REQUEST est envoyé (Annexe 10.18 et 10.19 a). Les options du message sont le type de message (adresse 240 à 242), l'adresse demandée (adresse 243 à 248), l'adresse du serveur (adresse 249 à 254) et l'option de fin (adresse 255). (Annexe 10.19 b)

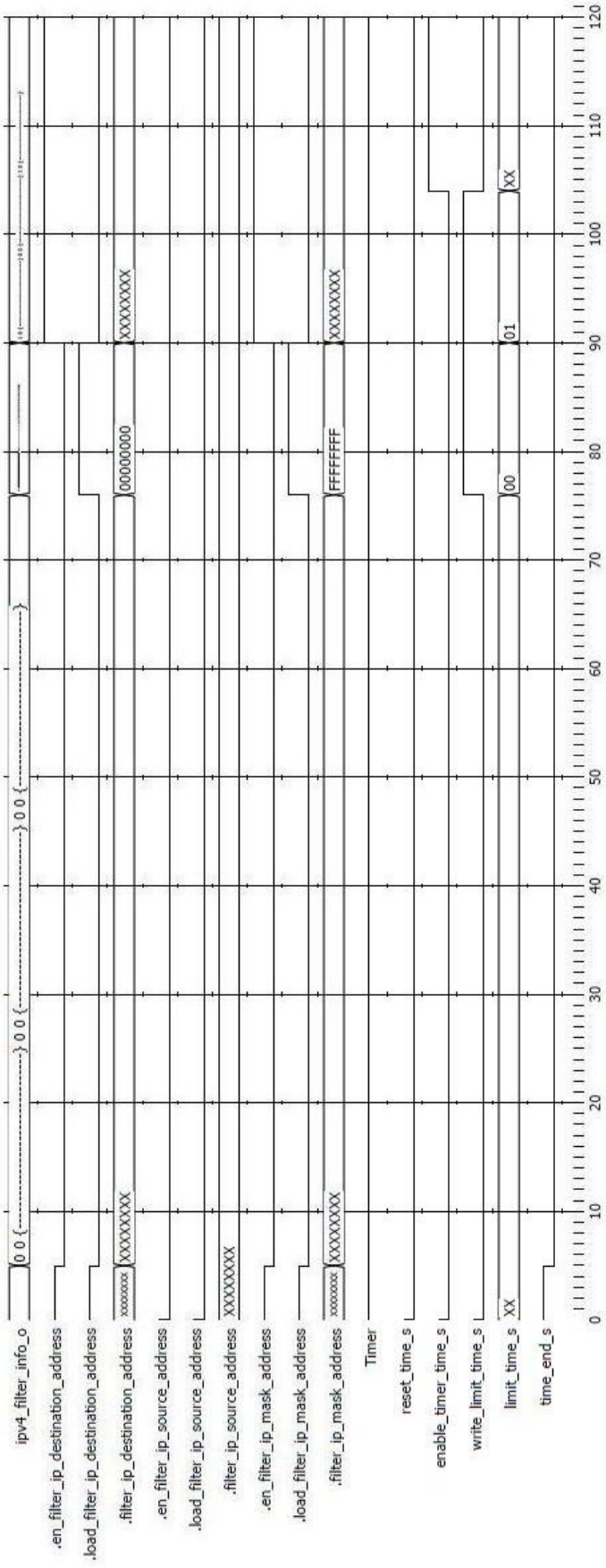
e) Les registres et les filtres IPv4 sont mis correctement à jour après la réception d'un message ACK (Annexe 10.18 et 10.20).

f) Lors des premiers tests, le message DHCP commençait par un byte 0 et tous les bytes étaient décalés. Le code de « controller_dhcp_write » a été modifié pour que les données soient synchronisées sur le compteur et plus sur l'horloge. Maintenant, les messages DISCOVER contiennent les données désirées. Le temps entre deux messages est d'une minute (ce temps n'est qu'une indication). (Annexe 10.21)

g) Lors de la réception du paquet OFFER, le message REQUEST est envoyé et contient les bonnes informations. (Annexe 10.21)

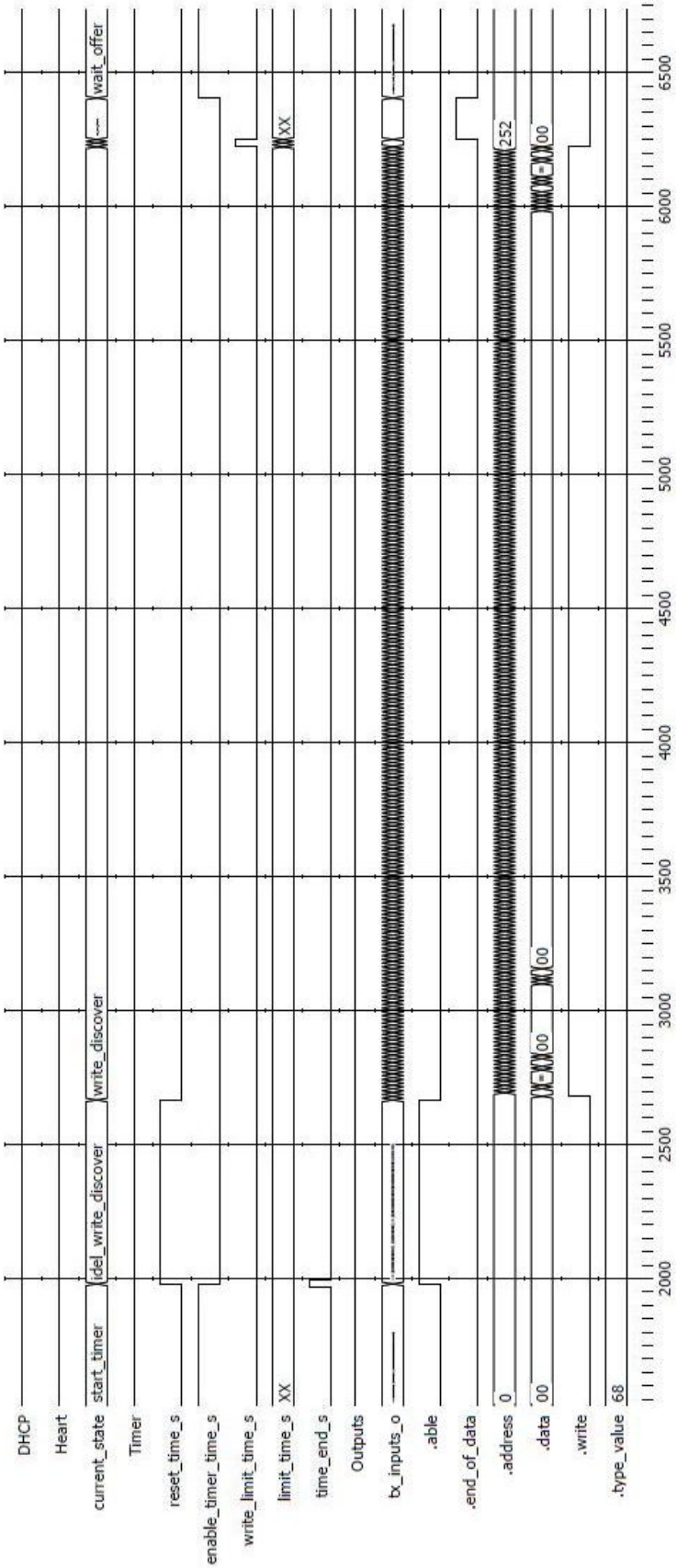
h) Les 3 premiers messages PING, après la réception du paquet ACK, n'ont pas de réponse. Par contre, les suivants, qui ont la bonne adresse IP, en ont une. Les filtres sont donc activés correctement. (Annexe 10.21)

10.16 DHCP INITIALISATION DES FILTRES IPv4

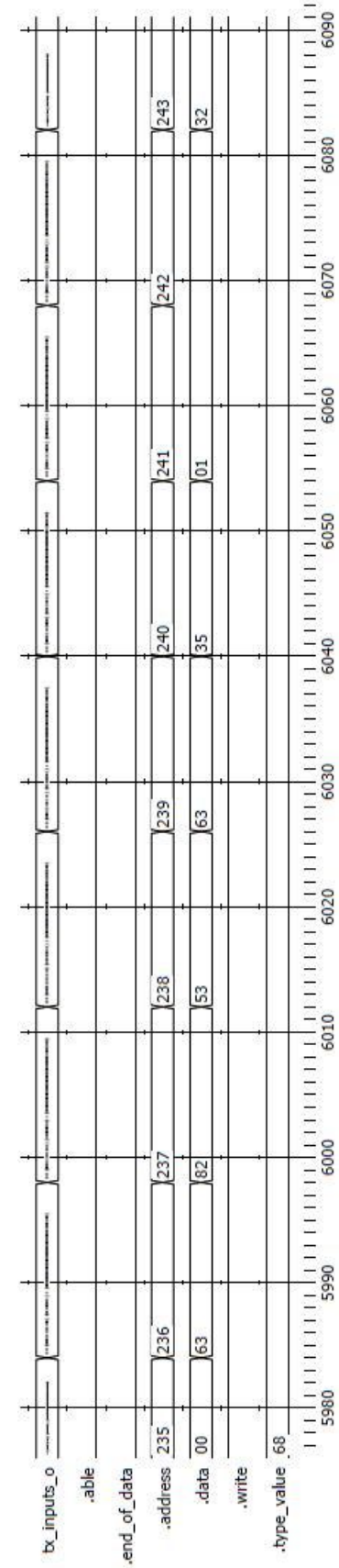


10.17 DHCP DISCOVER

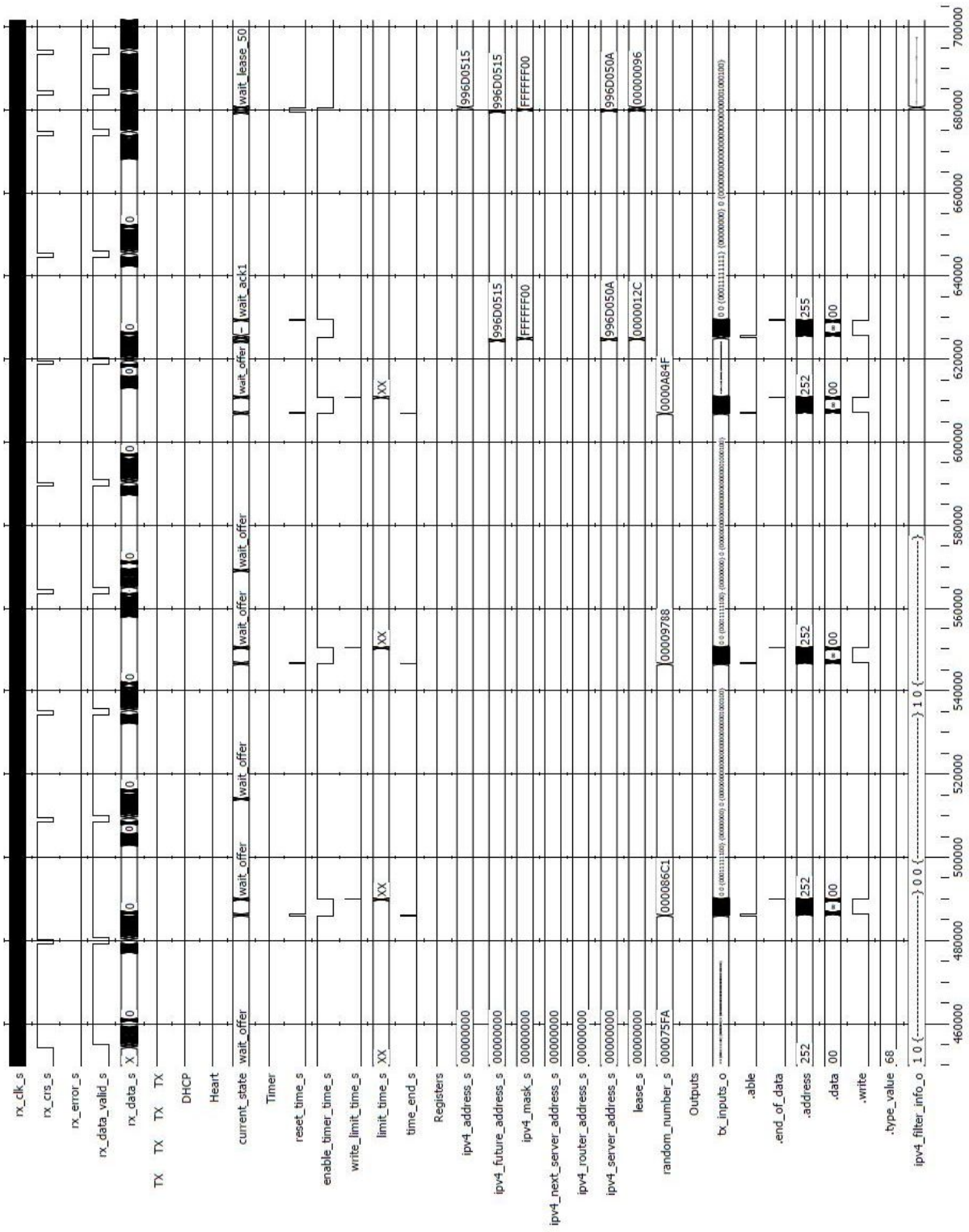
a)



b)

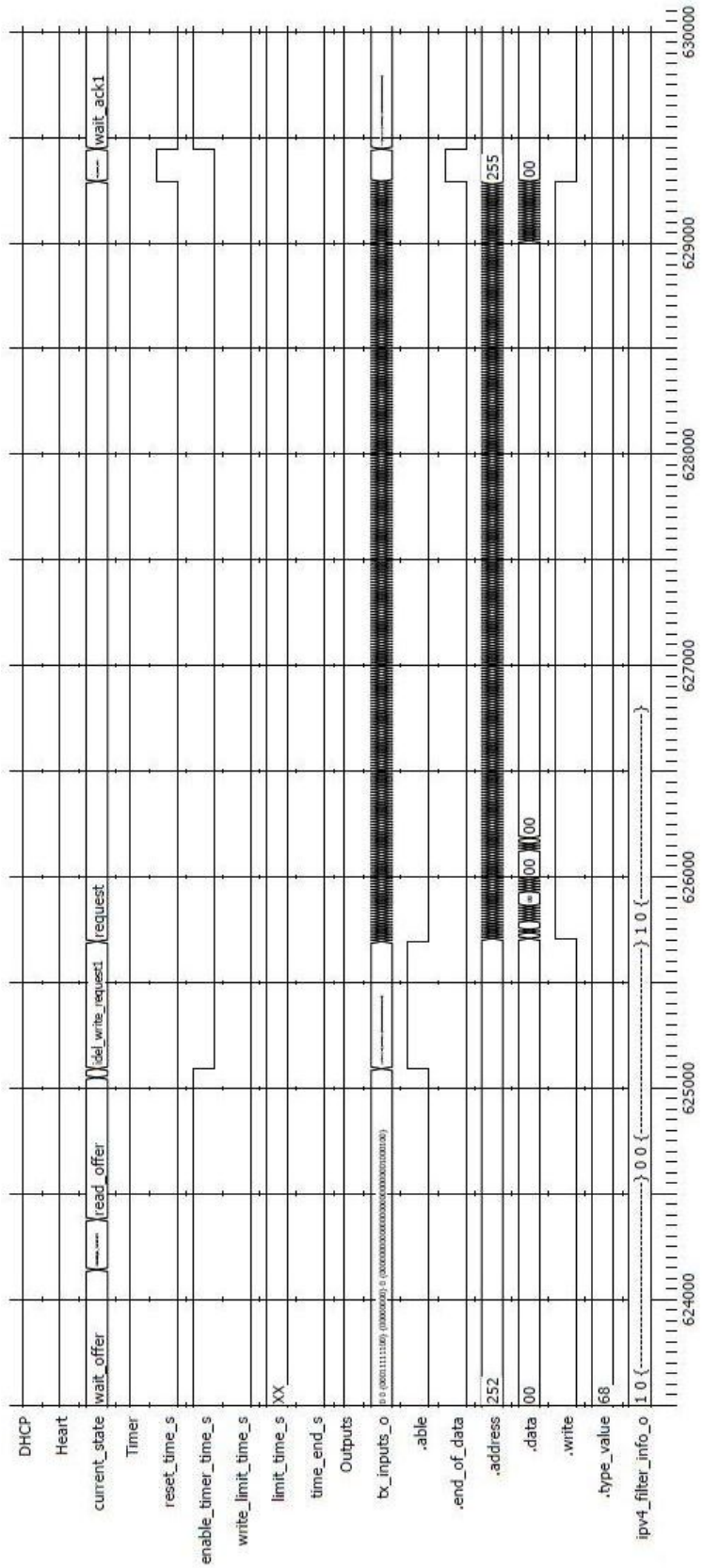


10.18 DHCP OFFER

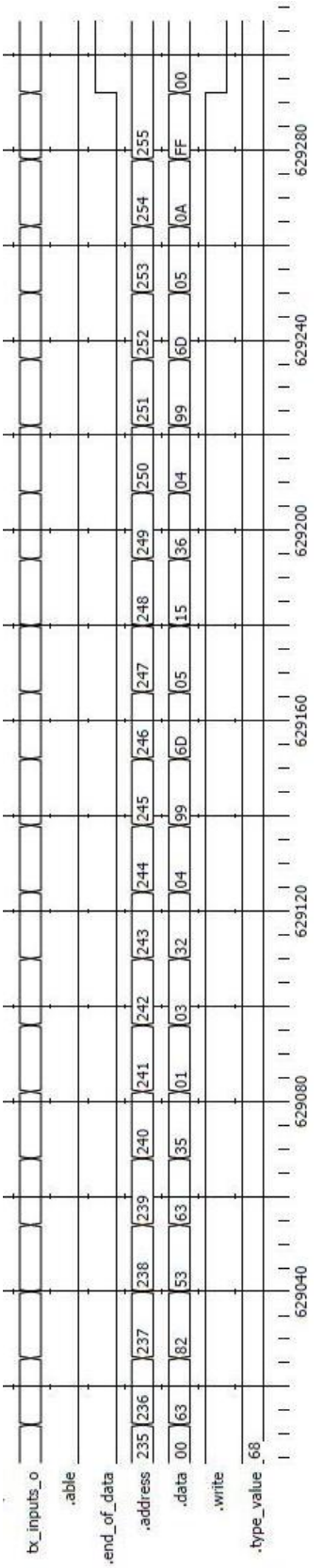


10.19 DHCP REQUEST

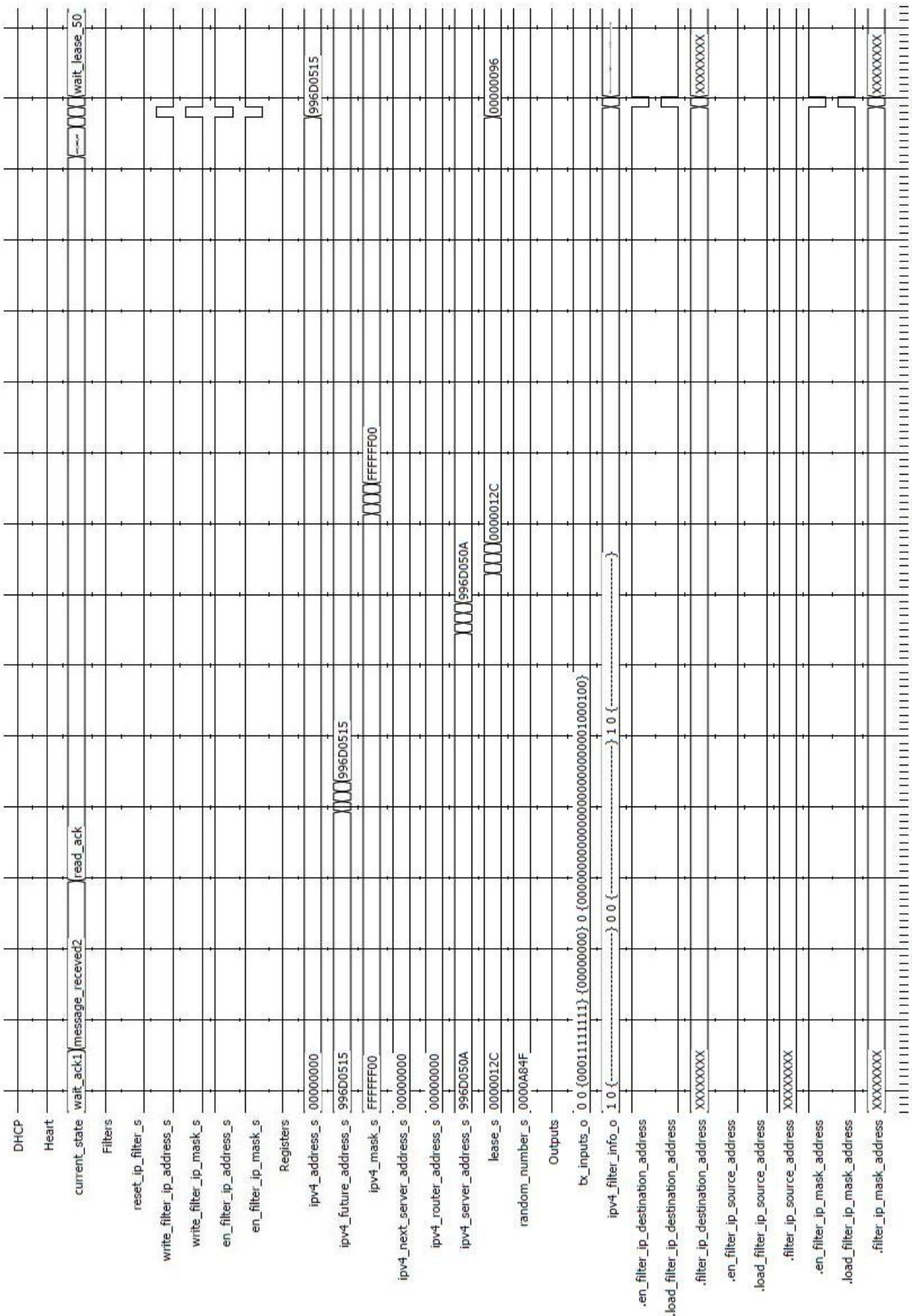
a)



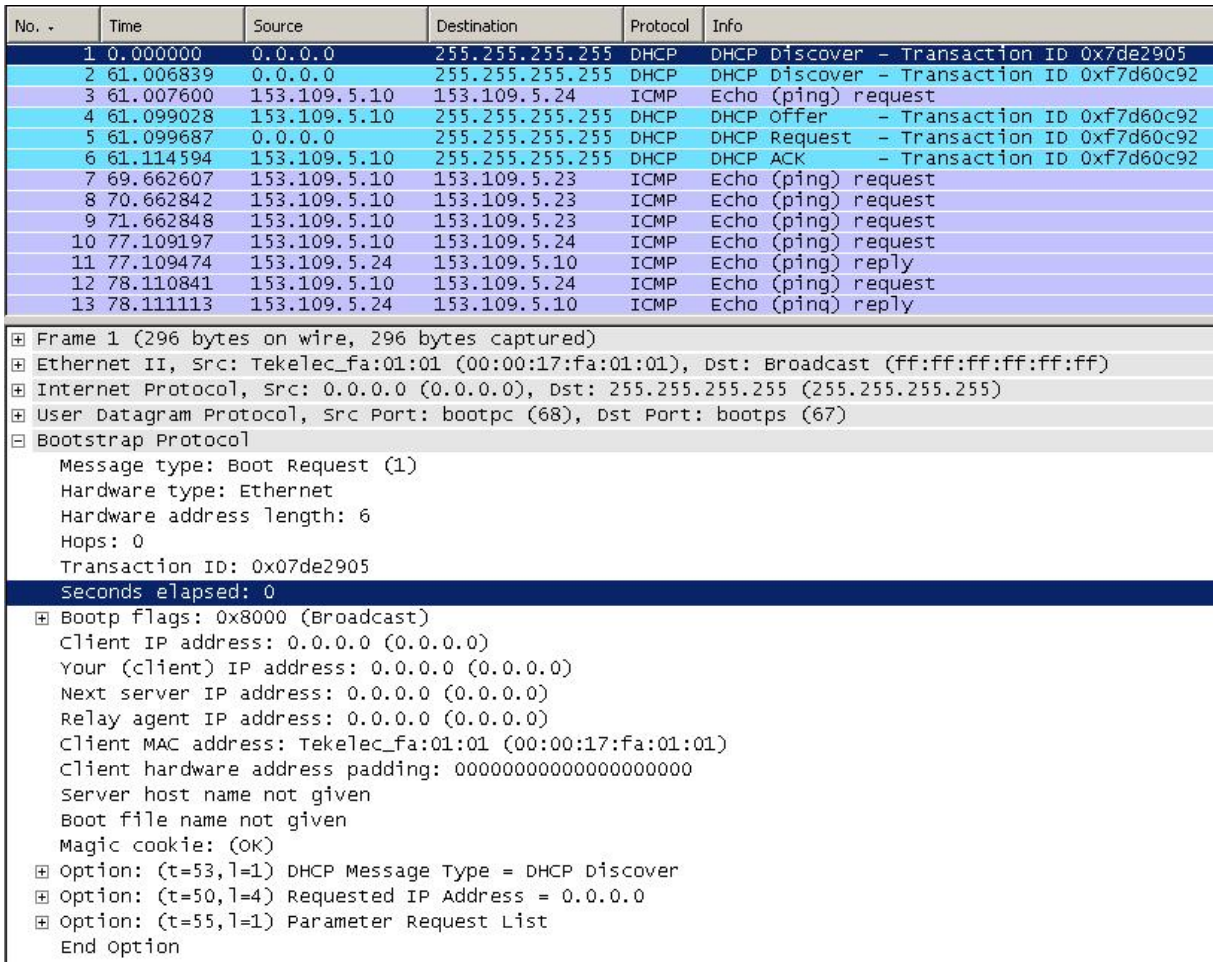
b)



10.20 DHCP ACK



10.21 1ÈRE CONNEXION DHCP (WIRESHARK)



No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x7de2905
2	61.006839	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xf7d60c92
3	61.007600	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
4	61.099028	153.109.5.10	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0xf7d60c92
5	61.099687	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xf7d60c92
6	61.114594	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xf7d60c92
7	69.662607	153.109.5.10	153.109.5.23	ICMP	Echo (ping) request
8	70.662842	153.109.5.10	153.109.5.23	ICMP	Echo (ping) request
9	71.662848	153.109.5.10	153.109.5.23	ICMP	Echo (ping) request
10	77.109197	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
11	77.109474	153.109.5.24	153.109.5.10	ICMP	Echo (ping) reply
12	78.110841	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
13	78.111113	153.109.5.24	153.109.5.10	ICMP	Echo (ping) reply

10.22 TEST RENOUVELLEMENT DE BAIL DHCP

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_dhcp

1) Condition(s)

Les premiers tests (a à c) sur DHCP sont faits sur ModelSim. La structure de ce test est « controller_of_tests_top_tb » et le fichier Wireshark utilisé est test_dhcp.cap. Les compteurs pour le temps (controller_dhcp_timer et controller_dhcp_timer_lease) ont été divisés par 1'000'000 pour diminuer le temps de la simulation. Le nombre aléatoire est fixe (0x07de2905).
Les tests suivants (d à e) sont fait en analysant les paquets avec Wireshark. La table de routage du PC a été mise à jour de façon à connaître les adresses de la FPGA. Le temps d'un bail est de 5 minutes, donc le premier message vient après 2 minutes 30, le second à 4 minutes 22 et la carte renvoie un DISCOVER après 4 minutes 41.

2) Test(s)

a) Vérifie que la machine d'état attend dans les bons états et que celui-ci change quand le timer du bail fini.

b) Examine que les messages envoyés sont bien des REQUEST et qu'ils contiennent les bonnes données.

c) Contrôle que les filtres soient bien resetés avant d'envoyer un nouveau DISCOVER.

d) S'assure que le système de renouvellement de bail fonctionne correctement et que les messages REQUEST soient corrects. Après l'envoi du paquet ACK, le serveur DHCP est stoppé. A la fin du bail, des trames PING sont envoyées. Les temps entre les messages REQUEST sont chronométrés.

e) Vérifie qu'après chaque paquet ACK, la FPGA reprend son attente au début. Pour ce faire le serveur est stoppé après le deuxième ACK et redémarré après la réception d'un message REQUEST. Puis on stoppe à nouveau le serveur pour le redémarrer après avoir reçu deux fois un REQUEST.

3) Résultat(s)

a) Chaque fois que le timer pour le bail finit son décompte, l'état de la machine d'état change. Ces changements d'état réenclenchent le timer. (Annexe 10.23)

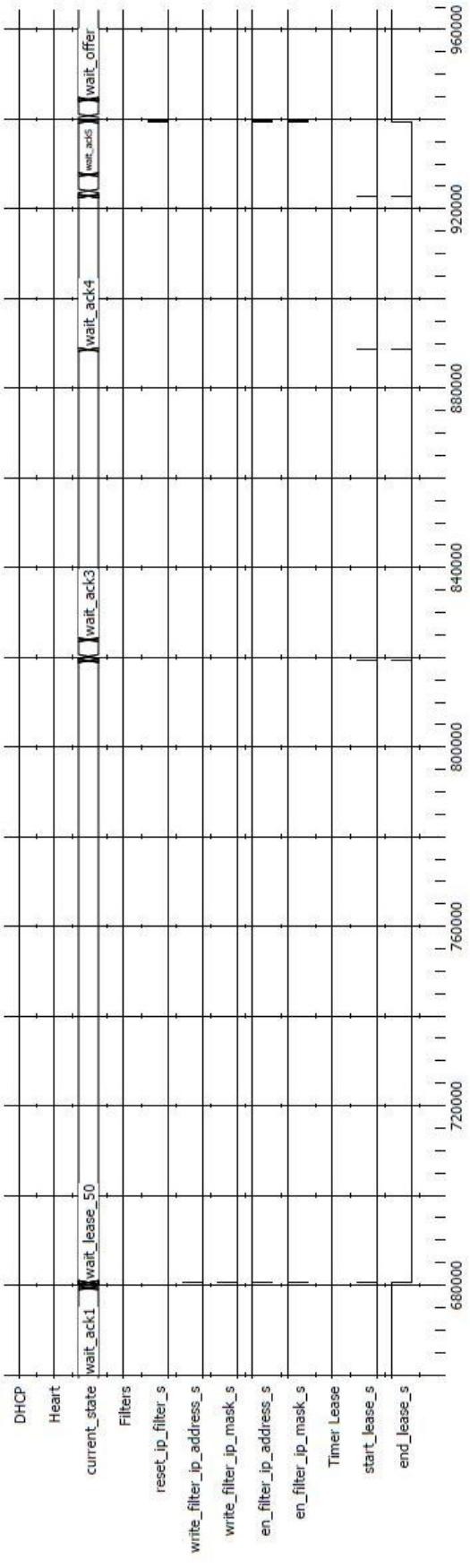
b) Le premier message de renouvellement de bail est bien un REQUEST (donnée 0x03 à l'adresse 242). Dans ses options, se trouve aussi l'adresse demandée (adresse 243 à 248), l'adresse du serveur (adresse 249 à 254) et l'option de fin (adresse 255) (Annexe 10.24 a). Le second message est aussi un REQUEST (l'adresse 242), l'adresse demandée est toujours là (adresse 243 à 248), et l'adresse du serveur ne l'est plus (Annexe 10.24 b). Les messages sont donc corrects.

c) Le signal « reset_ip_filter_s » est mis à 1 lors de la fin du bail. Ce signal remet les filtres IPv4 à leurs valeurs par défaut. (Annexe 10.22)

d) Lors des premiers tests le deuxième message de renouvellement de bail contenait des données complètement erronées. Ces fautes venaient du fait que, dans le bloc «controller_dhcp_write », il y avait des latch. Le code a été modifié pour les supprimer. Désormais, les messages contiennent les bonnes informations. Les temps sont de 2 minutes 30, 4 minutes 23 et 4 minutes 41 (Ces temps ont indicatifs). La commande PING ne reçoit aucune réponse. (Annexe 10.25 a)

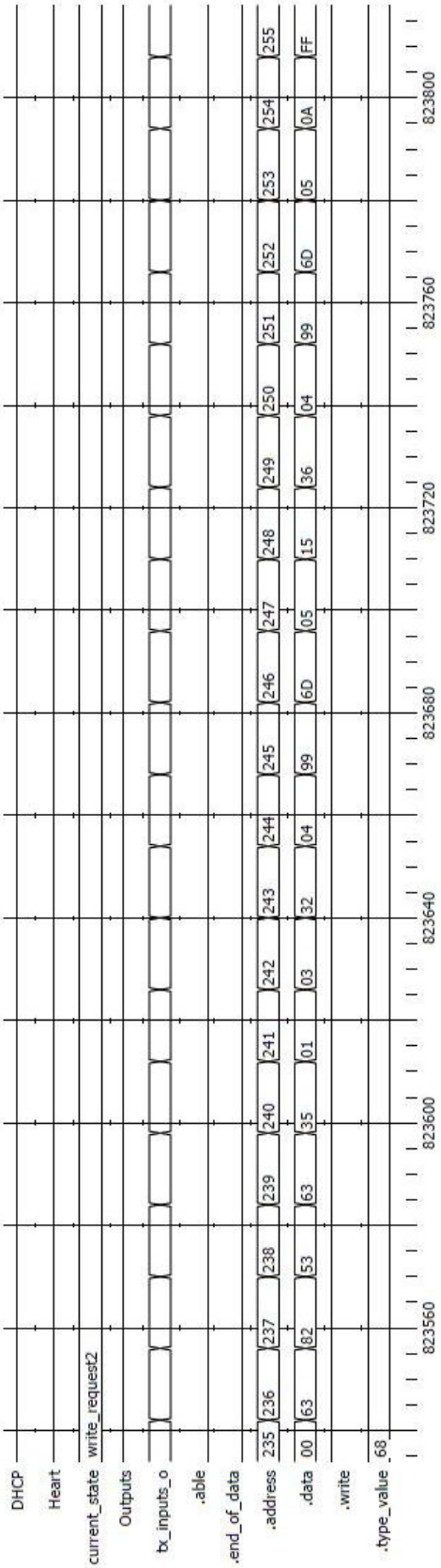
e) Après chaque réception d'un paquet ACK, la FPGA reprend l'attente du bail au début. (Annexe 10.25 b).

10.23 DHCP ATTEND RENOUVELLEMENT DE BAIL

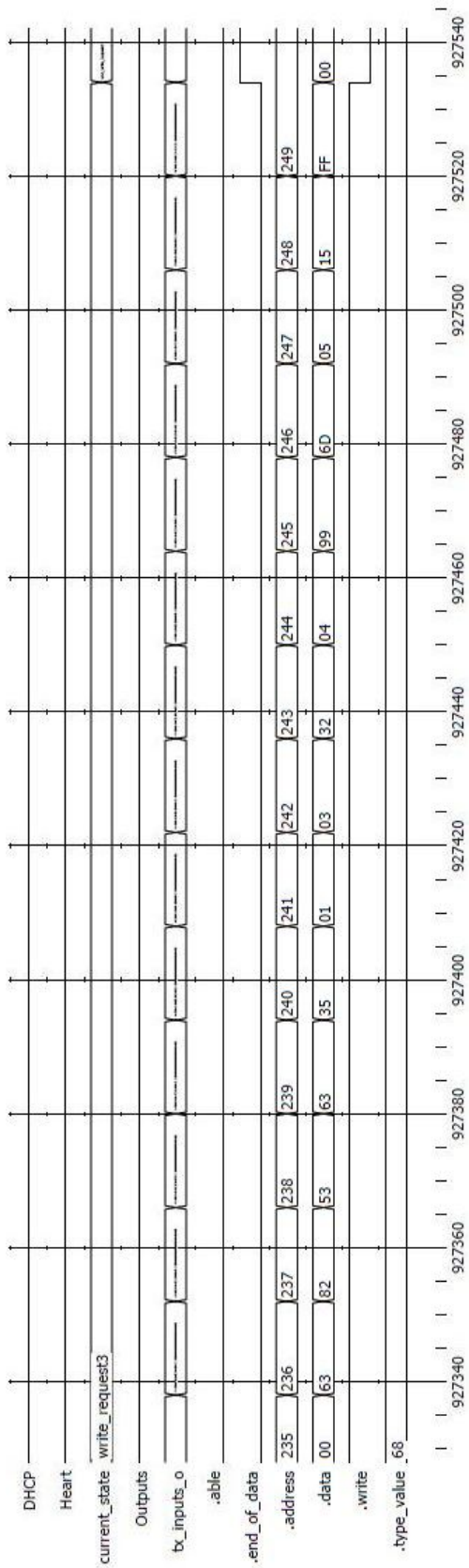


10.24 DHCP RENOUVELLEMENT DE BAIL (MODELSIM)

a)



b)



10.25 DHCP RENOUVELLEMENT DE BAIL (WIRESHARK)

a)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x7de2905
2	0.000163	153.109.5.10	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x7de2905
3	0.000053	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x7de2905
4	0.012738	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x7de2905
5	150.030524	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x5602d675
6	262.043294	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xe9bce50
7	280.045351	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x556b3f27
8	311.428741	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
9	312.443139	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
10	313.443176	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request
11	314.444061	153.109.5.10	153.109.5.24	ICMP	Echo (ping) request

```
⊞ Frame 6 (292 bytes on wire, 292 bytes captured)
⊞ Ethernet II, Src: Tekelec_fa:01:01 (00:00:17:fa:01:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
⊟ Bootstrap Protocol
    Message type: Boot Request (1)
    Hardware type: Ethernet
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x0e9bce50
    Seconds elapsed: 0
  ⊞ Bootp flags: 0x8000 (Broadcast)
    Client IP address: 153.109.5.24 (153.109.5.24)
    Your (client) IP address: 153.109.5.24 (153.109.5.24)
    Next server IP address: 0.0.0.0 (0.0.0.0)
    Relay agent IP address: 0.0.0.0 (0.0.0.0)
    Client MAC address: Tekelec_fa:01:01 (00:00:17:fa:01:01)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: (OK)
  ⊞ Option: (t=53,l=1) DHCP Message Type = DHCP Request
  ⊞ Option: (t=50,l=4) Requested IP Address = 153.109.5.24
    End Option
```

b)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x7de2905
3	0.320082	153.109.5.10	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x7de2905
4	0.320738	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x7de2905
5	0.350352	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x7de2905
6	150.368149	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x5756fcda
7	150.383546	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x5756fcda
8	300.401347	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xa57d2fd5
9	412.414123	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x5e1627c5
10	412.429810	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x5e1627c5
11	562.447607	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xac3ca2fa
12	674.460390	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x64d59ade
13	692.462446	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xaba50bbe
15	693.320182	153.109.5.10	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0xaba50bbe
16	693.320848	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xaba50bbe
17	693.337138	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xaba50bbe
18	843.354940	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xfd2c850f
19	843.371308	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xfd2c850f

10.26 TEST D'ENVOI DE TRAMES UDP SUR RS232

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_udp_rs232_sender, controller_udp_rs232_register

1) Condition(s)

Ces tests ont été effectués sur ModelSim. La structure de ce test est « controller_of_tests_udp_rs232_send_tb ». La taille des mémoires est de 8 Bytes et le Baudrate est le même pour les deux ports RS232 (460800). Les données à envoyer au port série commencent par 0x81 (seulement pour les tests avec ModelSim).

2) Test(s)

a) Teste que les données soient transmises sur le bon port série et que, quand la mémoire est pleine, les données attendent. Pour tester cela, deux trames sont envoyées à la partie envoi. La première, à 0us, est pour le port série 1 et contient 12 bytes, la seconde, à 100us, est pour le port série 0 et contient 7 bytes.

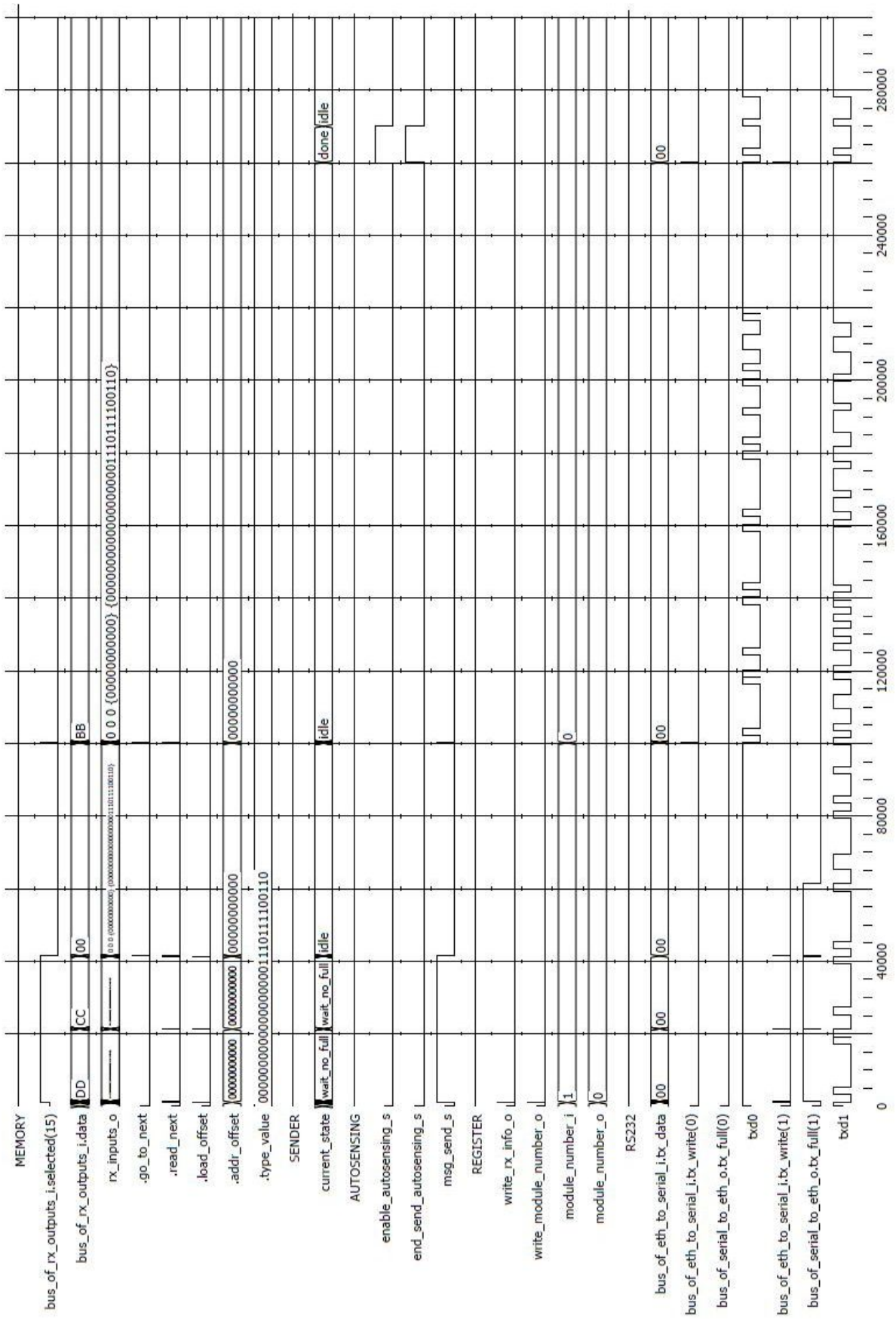
b) Contrôle que, lors d'une auto-détection, les données soient envoyées à tous les ports RS232 et que les signaux allant sur cette partie soient corrects. Pour ce faire, après 260 us, le signal « enable_autosensing_s » est mis à 1 et est remis à 0 10 us plus tard.

3) Résultat(s)

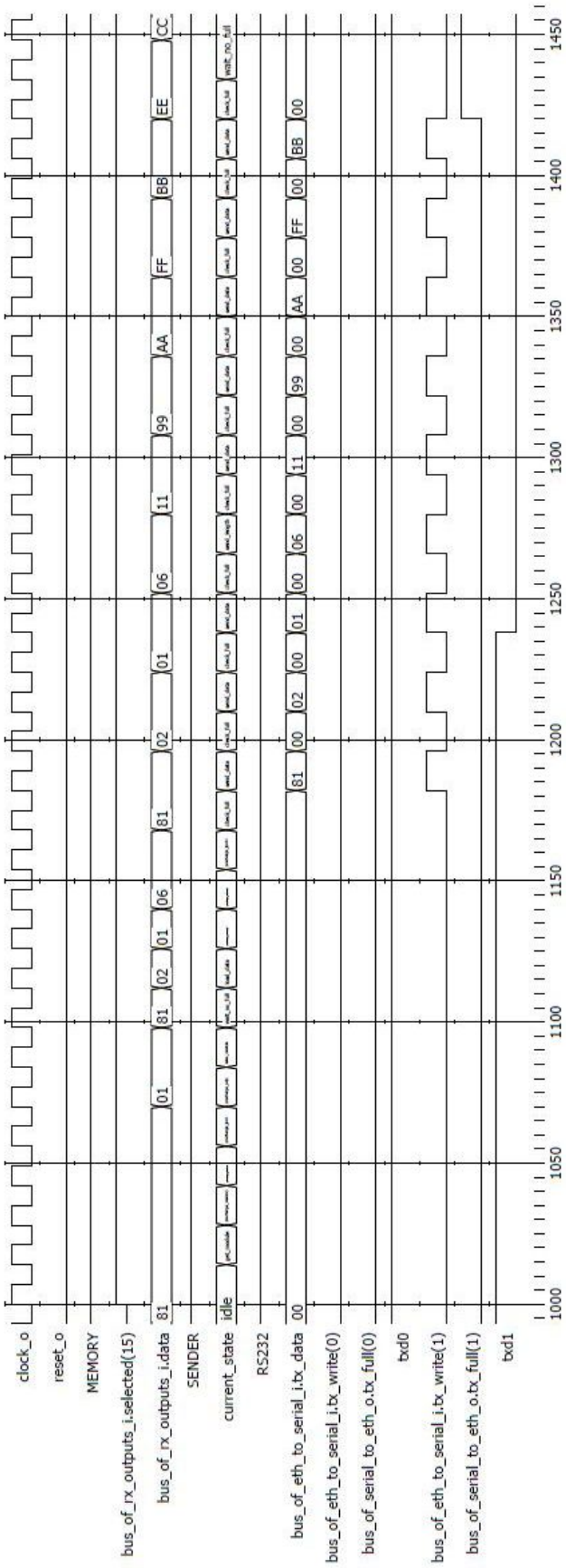
a) L'annexe 10.27 prouve que les données sont envoyées sur le bon port RS232 et que le signal « msg_send_s » est mis à l'état haut lors de la transmission. Ces données sont mises en attente lorsque la mémoire est pleine. L'annexe 10.28 est un zoom sur les données transmises pour vérifier qu'elles correspondent à celles reçues.

b) Lors d'une auto-détection, les données sont bien transmises sur les deux ports RS232 et lorsque la transmission est finie le signal « end_send_autosensing_s » est mis au niveau haut. (Annexe 10.27)

10.27 ENVOI DE TRAMES UDP SUR RS232



10.28 ENVOI DE TRAMES UDP SUR RS232 (ZOOM)



10.29 TEST DE RÉCEPTION SUR RS232 ET ENVOI SUR ETHERNET

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_udp_rs232_receiver, controller_udp_rs232_write

1) Condition(s)

Ces tests ont été effectués sur ModelSim. La structure de ce test est « controller_of_tests_udp_rs232_receive_tb ».

2) Test(s)

a) Contrôle que les données se trouvant sur les ports série soient transmises à la partie suivante et que le numéro du port soit rajouté. Pour cela, deux trames sont envoyées depuis le RS232, chacune sur un port différent.

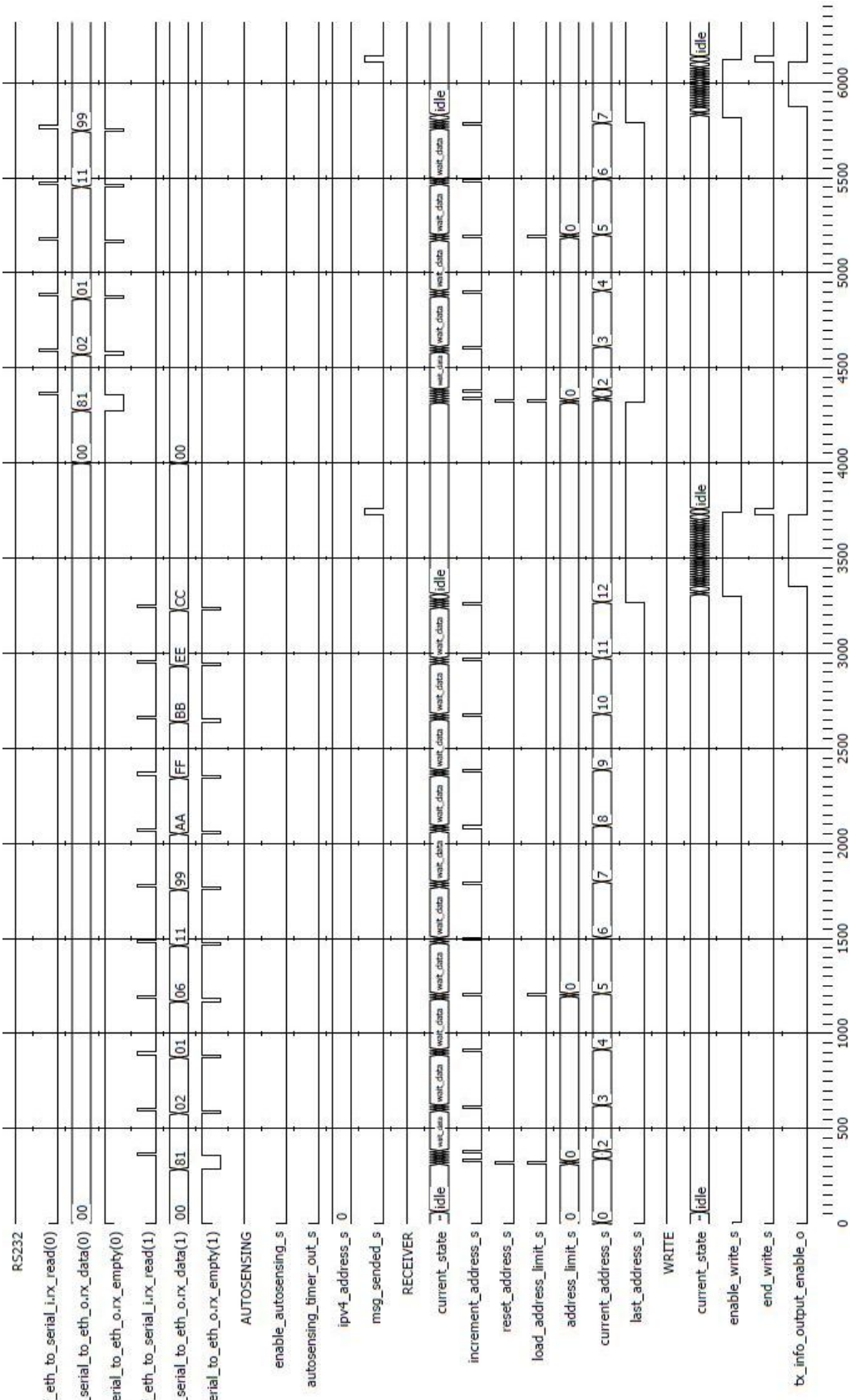
b) Vérifie que, lors de la phase d'auto-détection, les données reçues sur les ports RS232 ne soient pas envoyées plus loin et que les données d'auto-détection sont transmises. Après 7us, le signal « enable_autosensing_s » est mis à l'état haut. Seul le port 0 répond et le délai de réponse (« autosensing_timer_out_s ») est fini lors de la réception des données de ce port.

3) Résultat(s)

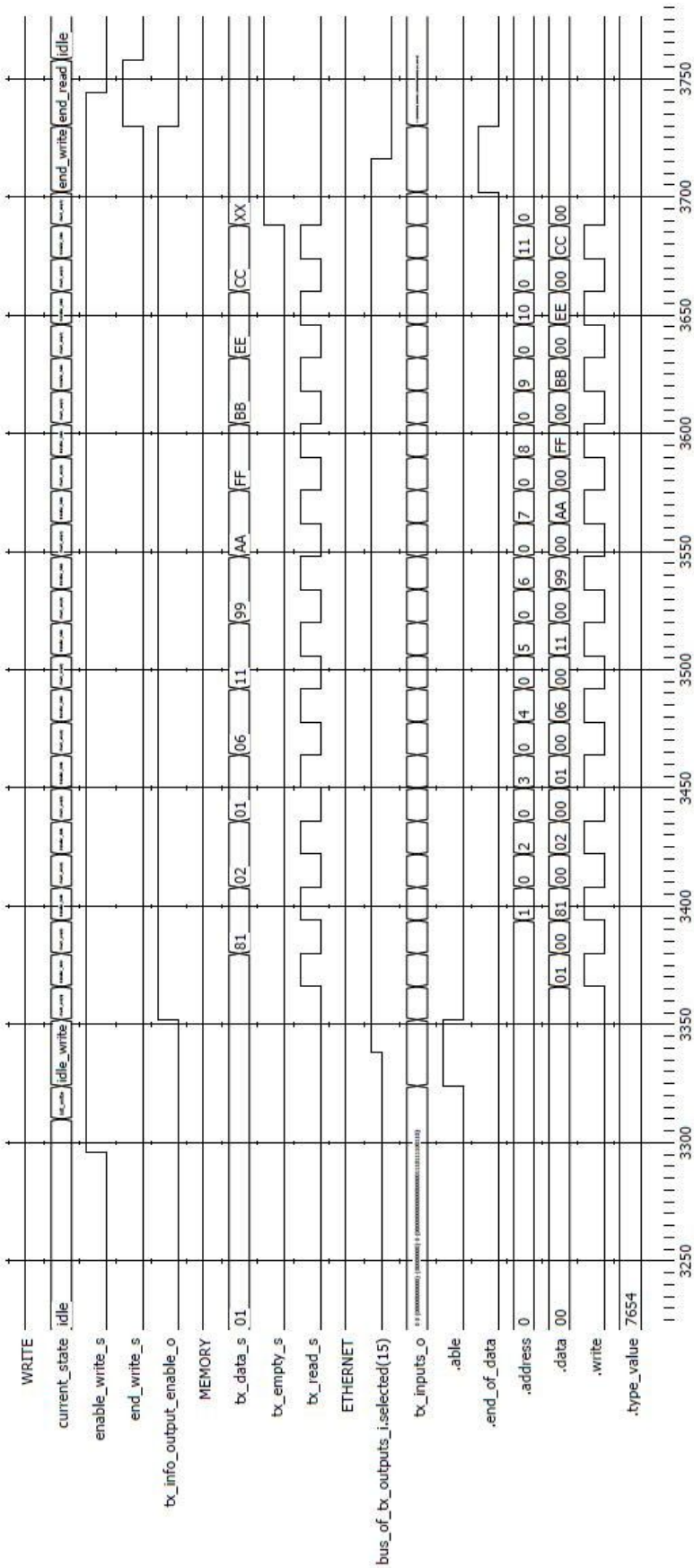
a) Les données sont bien réceptionnées avant d'être transmises à la partie « controller_udp_rs232_write ». Le byte indiquant le port est rajouté dans la trame et les données correspondent à celles reçues. (Annexe 10.30 et 10.31)

b) Le message d'auto-détection n'est envoyé qu'une fois toutes les données reçues. Ce message contient l'adresse MAC (00:00:17:FA:01:01) et IP (0.0.0.0), le port UDP (1DE6), le nombre de port RS232 (2), les ports RS232 actif (0) et le byte de fin (FF). Le paquet est donc correct. (Annexe 10.32 et 10.33)

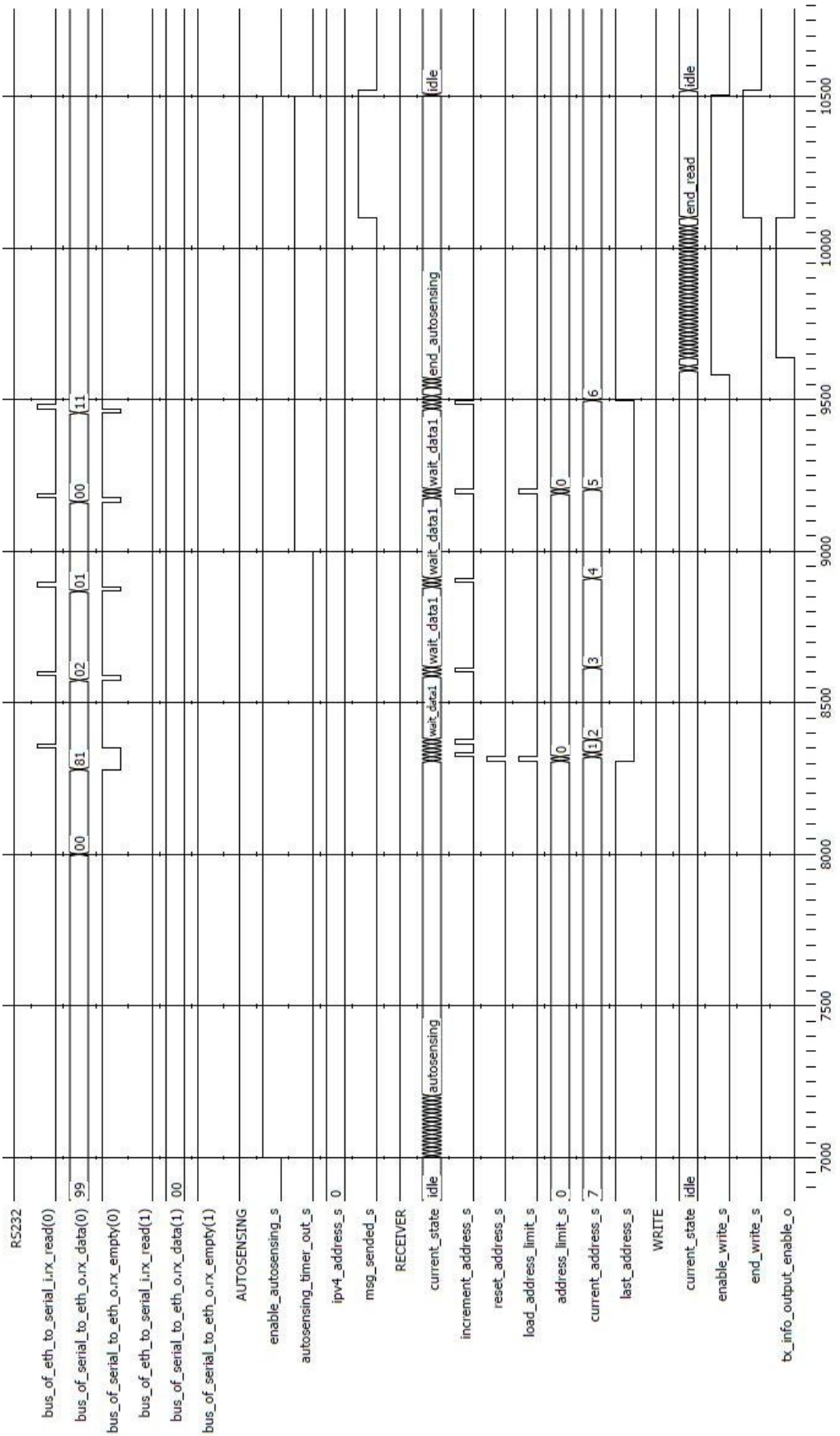
10.30 RÉCEPTION SUR RS232 ET ENVOI SUR ETHERNET



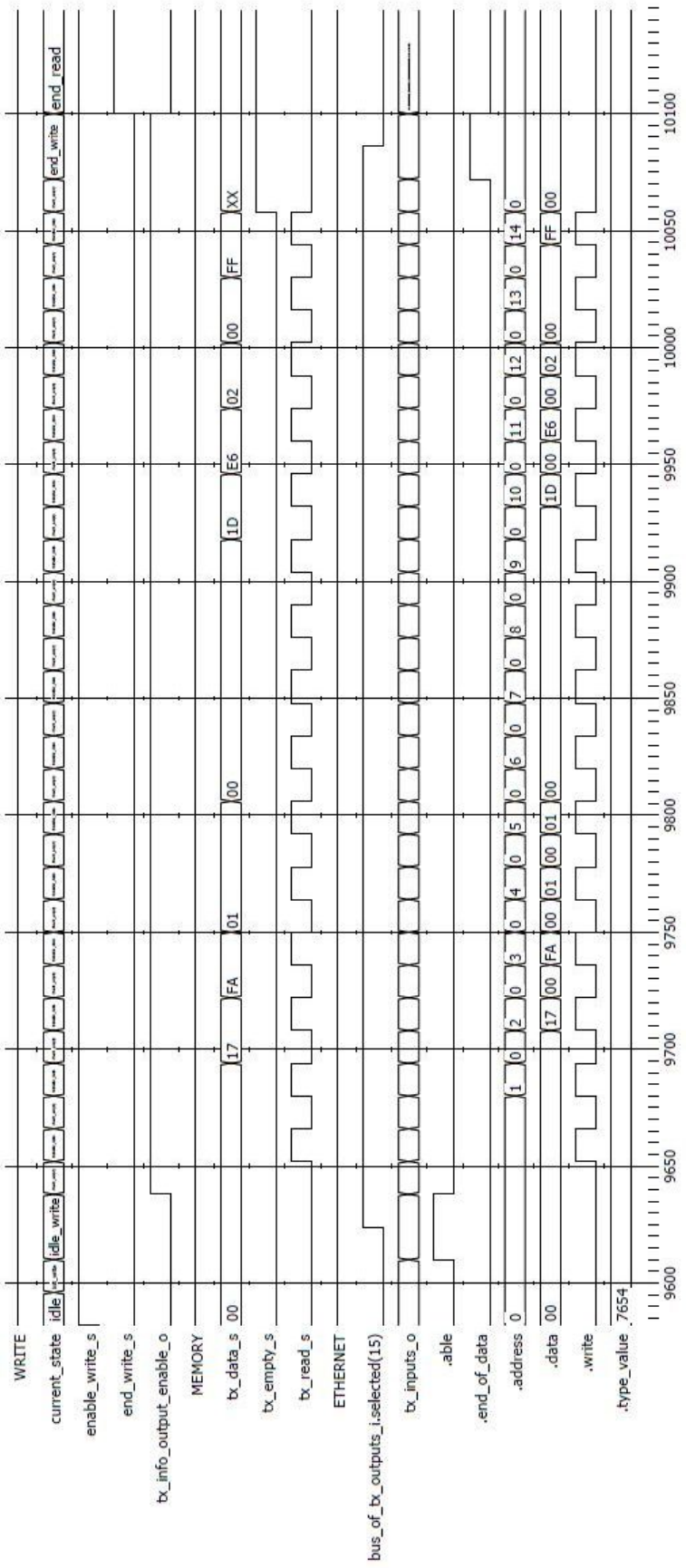
10.31 RÉCEPTION SUR RS232 ET ENVOI SUR ETHERNET (ZOOM)



10.32 RÉCEPTION AUTO-DÉTECTION ET ENVOI SUR ETHERNET



10.33 RÉCEPTION AUTO-DÉTECTION ET ENVOI SUR ETHERNET (ZOOM)



10.34 TEST AUTO-DÉTECTION

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	controller_udp_rs232_autosensing

1) Condition(s)

Ces tests ont été effectués sur ModelSim. La structure de ce test est « controller_of_tests_udp_rs232_autosensing_tb ».

2) Test(s)

a) Contrôle que, lorsque la FPGA n'a pas d'adresse IP, aucun message d'auto-détection n'est envoyé. Pour effectuer cette vérification, le filtre IP de destination est changé pour l'adresse 0.0.0.0 (adresse de reset) après 100ns.

b) Vérifie que, pendant qu'un message est en traitement, la procédure d'auto-détection ne commence pas. Pour cela, le signal « msg_send_i » est mis à l'état haut après 150 ns et cela pendant 2 flancs d'horloge. Par la suite, le filtre IP est modifié (200ns).

c) Examine que l'ordre des actions effectuées lors de l'auto-détection est exécuté correctement. Le signal « msg_sended_i » est mis à 1 deux fois, la première pour finir le traitement du message et la seconde pour signaler que le message d'auto-détection est terminé.

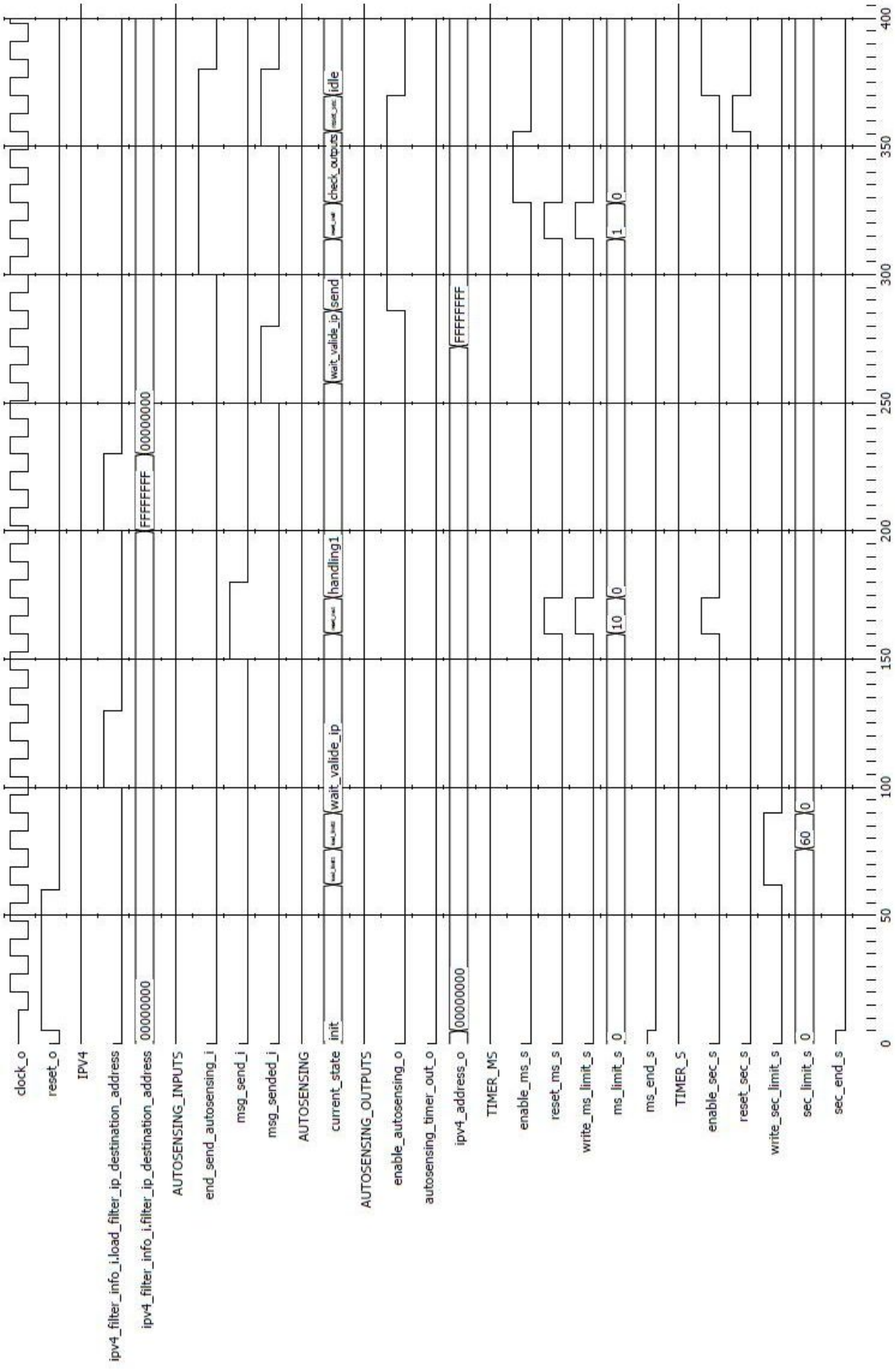
3) Résultat(s)

a) La machine d'état de l'auto-détection n'a pas changé après la modification du filtre IP, c'est ce qui était attendu. (Annexe 10.35)

b) Après que « msg_send_i » ai été modifié, la machine d'état n'a pas changé, avant que « msg_sended_i » passe à 1. (Annexe 10.35)

c) Lors de l'auto-détection le signal « enable_autosensing_o » est à 1. Après l'envoi des données (« end_send_autosensing » à 1), le timer pour les millisecondes est reseté et enclenché et, quand le message d'auto-détection est envoyé (« msg_sended_i » à 1), il est arrêté tandis que le timer pour les secondes est reseté et enclenché. Cette partie fonctionne donc comme désiré. (Annexe 10.35)

10.35 GESTION DE L'AUTO-DÉTECTION



10.36 TEST TRANSMISSION UDP RS232

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	udp_rs232

1) Condition(s)

A l'exception du d, tous les tests sont faits en réel. Le programme est chargé dans la mémoire flash de la carte FPGA. Les Baudrate sont de 57600 pour le port 1 et de 9600 pour le 0.
Pour les tests a et b, la taille des mémoires est de 8 bytes pour la partie RS232 et de 260 bytes pour celle de RECEIVER. Le temps entre chaque message d'auto-détection est d'une seconde.
Pour le test d, la structure utilisée est « controller_of_test_rs232_tb ».
Pour les autres, la taille de la mémoire est de 512 bytes et le temps entre chaque message est d'une minute. Les tests c et e utilisent le code en Perl se trouvant dans le fichier « test_FPGA.pl ».

2) Test(s)

a) S'assure que les données sont envoyées sur le port RS232. Un câble RS232 est branché entre la carte FPGA et l'ordinateur.

   i) Les bytes envoyés sont analysés avec le programme « SuPTerminal ».

   ii) La pine Tx du RS232 est analysée avec un oscilloscope pour vérifier s'il y a des changements sur celle-ci.

   iii) Les signaux « enable_autosensing_s », « tx_write(0) » et « txd0 » sont amenés sur la sortie debug et dirigés vers les LEDs. Ces signaux sont analysés avec l'oscilloscope pour déterminer s'ils sont modifiés comme désiré.

   iv) Les signaux « txBusy », « txSend », « txFifoEmpty » et « txWr » se trouvant dans « serialPortFifo » sont dirigés vers les LEDs et analysés avec l'oscilloscope.

   v) La schématique RTL (option « View RTL Schematic » dans ISE) est contrôlée pour voir s'il n'y a pas d'erreur dans celle-ci.

   vi) Le test i est refait.

b) Vérifie que les paquets d'auto-détection sont envoyés périodiquement et contiennent les bonnes informations. Wireshark analyse les trames après que la carte soit branchée au réseau de test et redémarrée.

c) Contrôle avec Wireshark que les paquets d'auto-détection et les trames envoyées depuis le PC soient transmis correctement. Un oscilloscope est branché sur la pine de réception et envoie RS232 du module de test pour s'assurer que les informations transmises soient correctes.

d) Teste que toutes les parties de transmission UDP-RS232 et d'auto-détection fonctionnent quand elles travaillent ensemble.

e) Le test du point c est refait après modification du code.

3) Résultat(s)

a)

   i) Aucune trame n'est reçue par le port RS232.

   ii) La pine Tx reste toujours à 1 (environs -5V), donc aucun byte n'ai envoyé sur ce port.

   iii) Les signaux changent normalement à l'exception du signal « txd0 » qui ne bouge pas.

   iv) Le signal « txWr » change d'état quand il le faut, mais pas « txFifoEmpty » ce qui bloque la sortie vers le RS232.

v) Un signal d'horloge différent est utilisé par le bloc « controller_rs232 ». Les signaux d'entrée du clock et du reset ont été inversés.

vi) Les données sont transmises correctement sur le port RS232 (Annexe 10.37).

b) Lors des premiers tests, certains messages d'auto-détection étaient incorrects. Les données étaient là mais pas à la bonne place et répétées plusieurs fois (Annexe 10.38 a). Le problème venait du fait que le nombre d'emplacement mémoire doit être une puissance de 2, ce qui n'était pas le cas. Maintenant, les paquets contiennent tous les adresses MAC et IP, le port UDP, le nombre de port RS232 et finissent par 0xFF. Ces messages sont envoyés périodiquement. (Annexe 10.38 b)

c) Les réponses aux messages envoyées depuis le PC sont fausses. L'oscilloscope montre que les trames arrivant et partant du module sont correctes (Annexe : 10.39), ce qui veut dire qu'il y a une erreur lors de la retransmission. Pour trouver l'erreur un test sur ModelSim doit être fait.

d) Lors des premiers tests, les données stockées n'étaient pas les bonnes. Le signal « rx_read » était mis à 1 avant de stocker la donnée, la donnée stockée était donc fausse. Après modification de la machine d'état, les données sont enregistrées correctement (Annexe 10.40 et 10.41)

e) Les messages reçus sont maintenant corrects. (Annexe : 10.42)

10.37 AUTO-DÉTECTION (SUPTERMINAL)

```
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
```

10.38 AUTO-DÉTECTION (WIRESHARK)

a)

No.	Time	Source	Destination	Protocol	Info
25	17.694417	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
26	18.696532	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
27	19.698652	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
28	20.700771	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
29	21.702894	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois

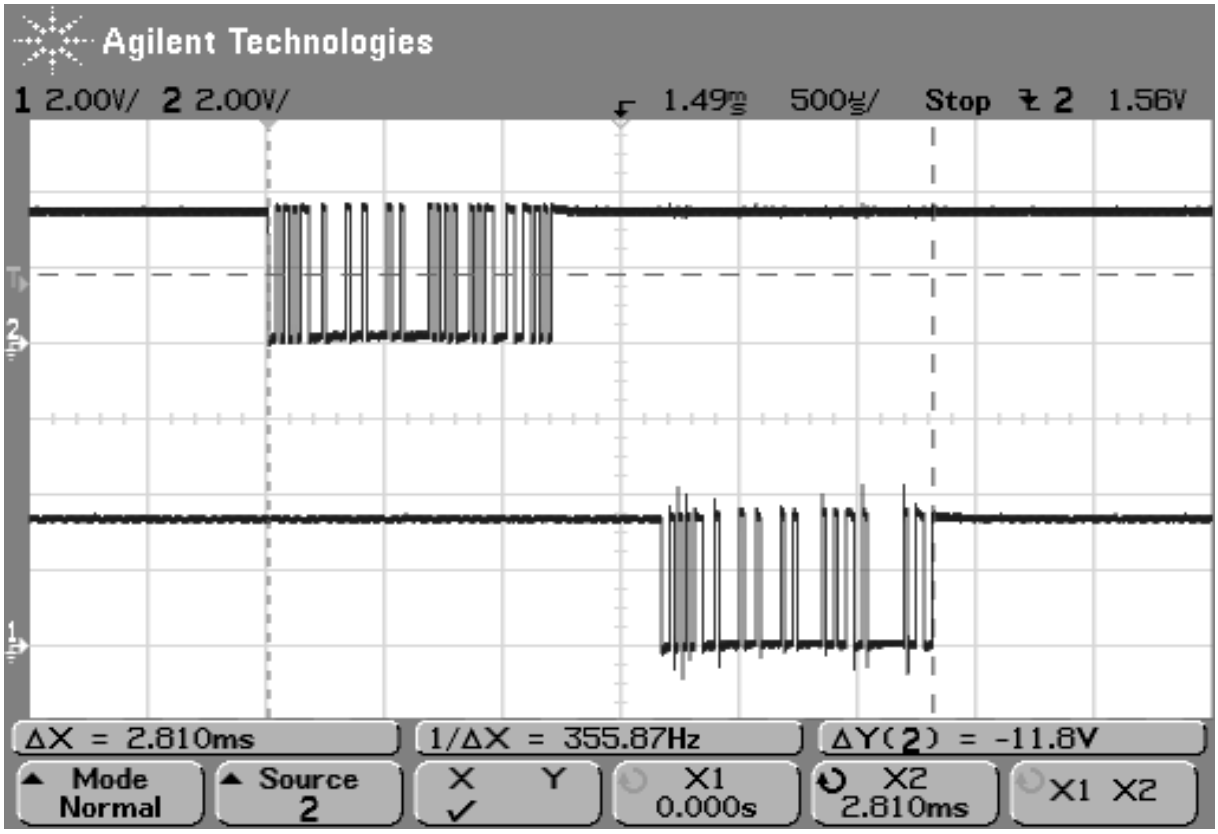
```
⊞ Frame 28 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Tekelec_fa:01:01 (00:00:17:fa:01:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 153.109.5.24 (153.109.5.24), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: 7654 (7654), Dst Port: rwhois (4321)
⊟ Data (14 bytes)
    Data: 17FA010117FA010117FA010117FA
    [Length: 14]
```

b)

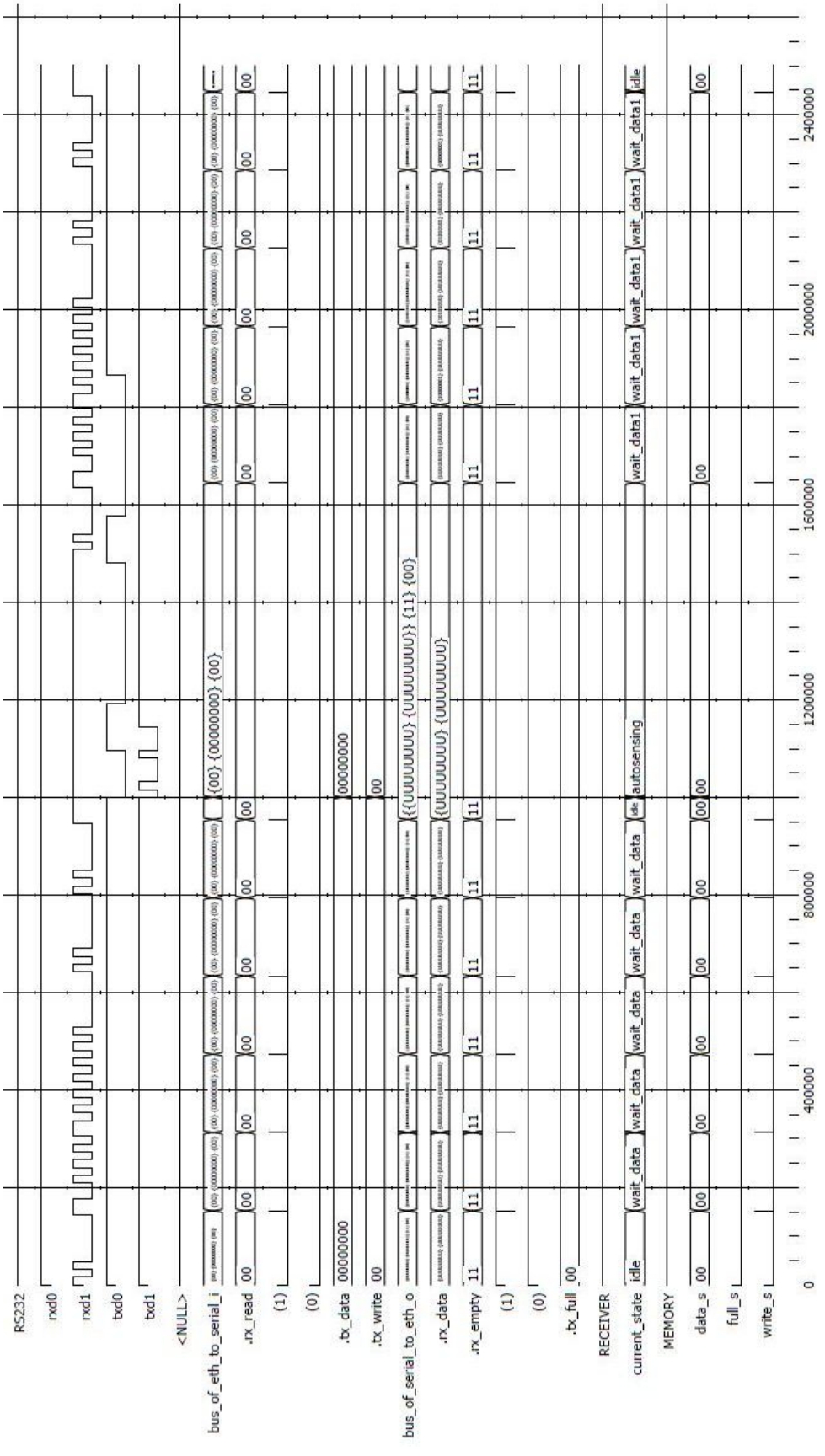
No.	Time	Source	Destination	Protocol	Info
23	15.670397	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
24	16.672516	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
25	17.674632	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
26	18.676749	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
27	19.678864	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
28	20.680984	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
29	21.683095	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois

```
⊞ Frame 28 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Tekelec_fa:01:01 (00:00:17:fa:01:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 153.109.5.24 (153.109.5.24), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: 7654 (7654), Dst Port: rwhois (4321)
⊟ Data (14 bytes)
    Data: 000017FA0101996D05181DE602FF
    [Length: 14]
```

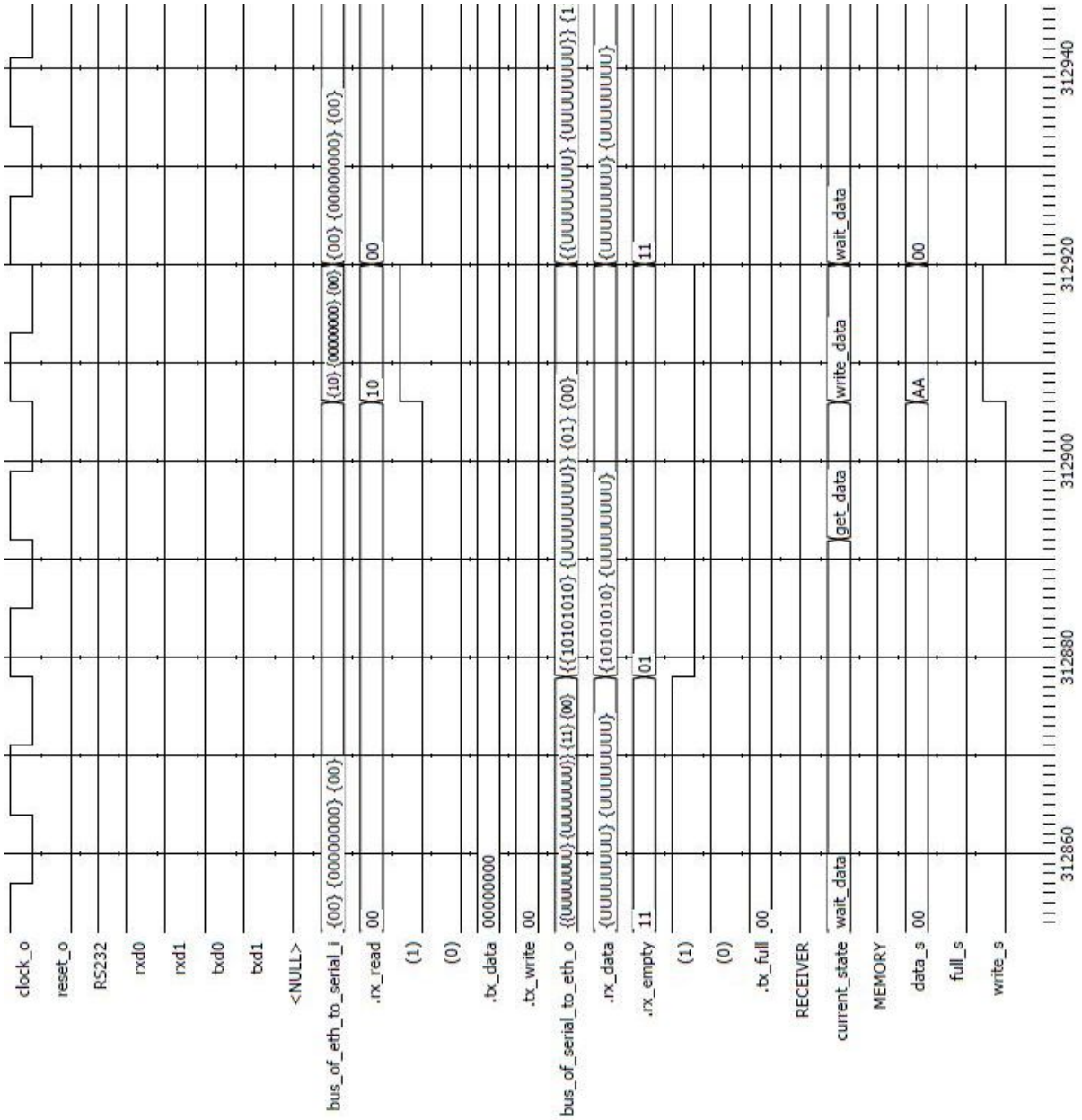
10.39 TRANSMISSION UDP RS232 (OSCILLOSCOPE)



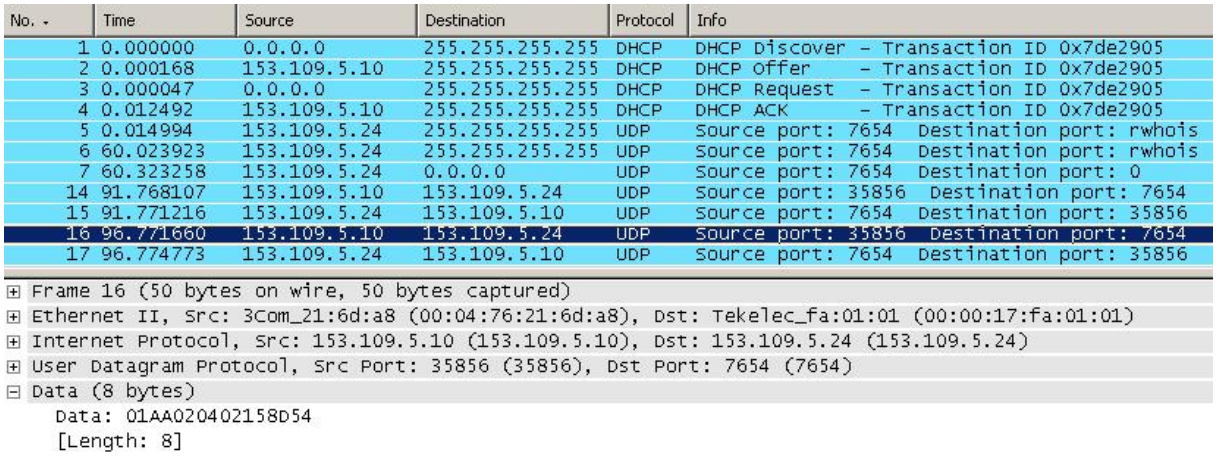
10.40 TRANSMISSION UDP RS232 (MODELSIM)



10.41 TRANSMISSION UDP RS232 (MODELSIM ZOOM)



10.42 TRANSMISSION UDP RS232 (WIRESHARK)



No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x7de2905
2	0.000168	153.109.5.10	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x7de2905
3	0.000047	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x7de2905
4	0.012492	153.109.5.10	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x7de2905
5	0.014994	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
6	60.023923	153.109.5.24	255.255.255.255	UDP	Source port: 7654 Destination port: rwhois
7	60.323258	153.109.5.24	0.0.0.0	UDP	Source port: 7654 Destination port: 0
14	91.768107	153.109.5.10	153.109.5.24	UDP	Source port: 35856 Destination port: 7654
15	91.771216	153.109.5.24	153.109.5.10	UDP	Source port: 7654 Destination port: 35856
16	96.771660	153.109.5.10	153.109.5.24	UDP	Source port: 35856 Destination port: 7654
17	96.774773	153.109.5.24	153.109.5.10	UDP	Source port: 7654 Destination port: 35856

10.43 TEST DU PROGRAMME D'AUTO-DÉTECTION

Project	Composant Ethernet en FPGA
Participants	Bastien Praplan
Place	A309
Partie testée	autosensing.pl

1) Condition(s)

Ces tests s'effectuent en réel. La carte FPGA est resetée et le programme autosensing.pl est démarré.

2) Test(s)

a) Contrôle que l'adresse d'un message d'auto-détection soit rajoutée. Le fichier XML est vide.

b) S'assure que les ports actifs sont modifiés correctement. Deux ports actifs sont rajoutés dans le fichier XML et le 1 est supprimé. Le module est allumé.

c) Examine que lors du démarrage du programme tout fonctionne correctement. La table ARP est vidée et le fichier XML contient des cartes.

d) Vérifie que les données dans le fichier XML et la table ARP sont modifiées correctement. Des cartes sont rajoutées dans le fichier XML, une avec un temps dépassé et l'autre avec un temps en avance.

e) Teste si le programme peut créer un fichier XML. Le fichier XML est renommé pour ce test.

3) Résultat(s)

a) La carte et ses informations sont rajoutées.

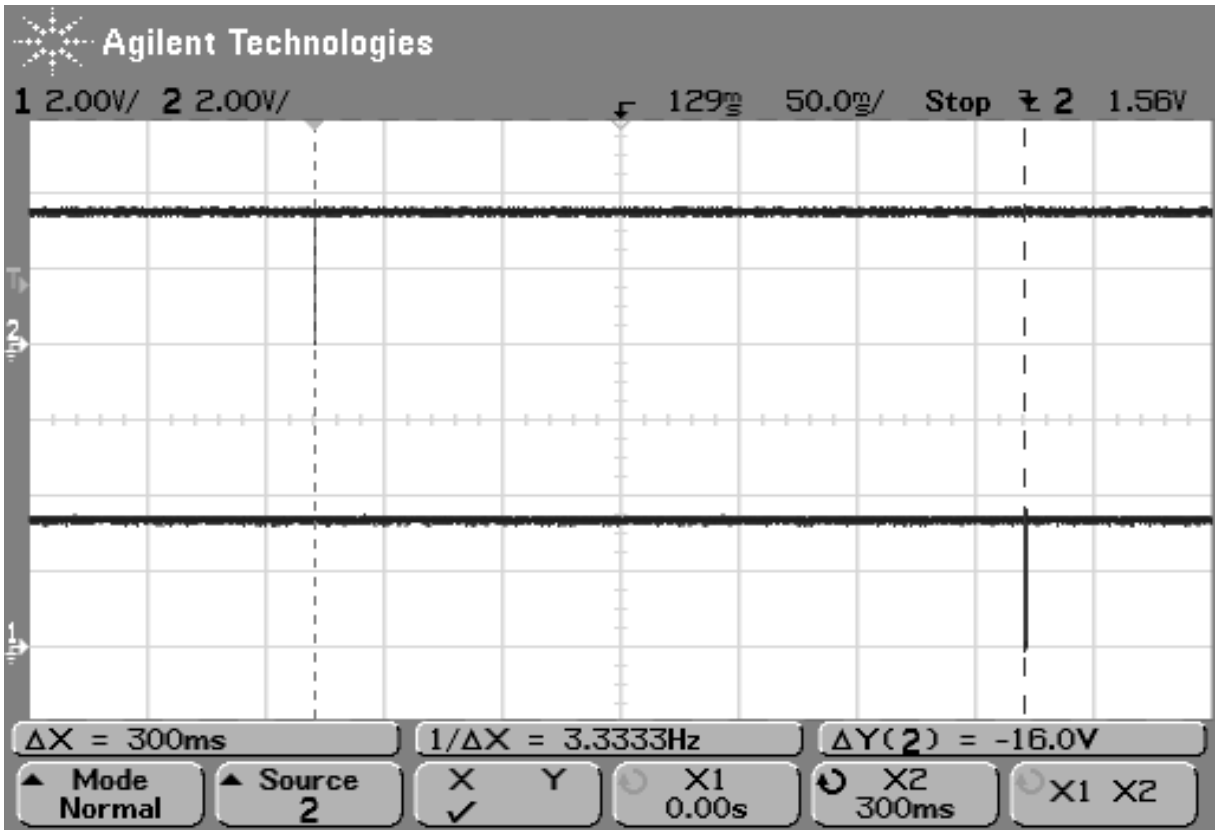
b) Les ports actifs sont rajoutés et supprimés. Cependant les tags « activePort » des éléments supprimés restent dans le fichier. L'option « SuppressEmpty » à été rajoutée à XMLin et XMLout pour supprimer ces tags, il reste le temps de recevoir un nouveau message.

c) Lors des premiers tests la table ARP ne se mettait pas à jour avec les cartes se trouvant dans le fichier XML. Le code à été modifié pour qu'elle le fasse.

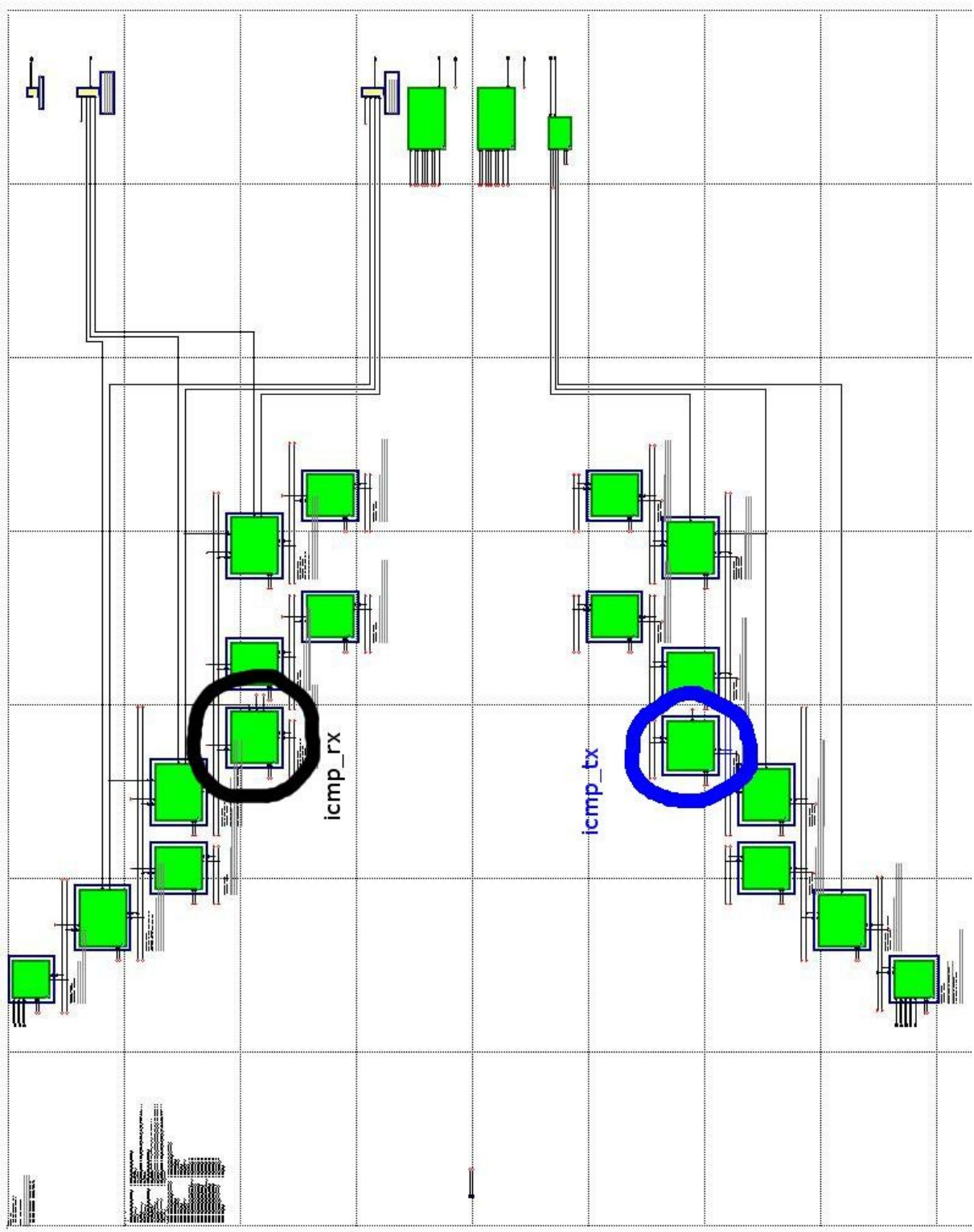
d) Les cartes sont supprimées correctement.

e) Le programme ne peut pas créer le fichier XML. Une erreur est affichée sur le terminal et le code Perl s'arrête.

10.44 ERREUR TEMPS DE RÉPONSE



10.45 STRUCTURE VHDL – MANAGER_OF_PROTOCOLS_STACK



10.46 STRUCTURE VHDL – BLOCS DE BASE