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 2012

Johan Droz

Base de temps pour réseaux distribués

Professeur	Joseph Moerschell
Expert	Peter Zweifel

Sion, le 9 juillet 2012

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/17**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Johan Droz** Professeur / *Dozent* **Joseph Moerschell**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* *(données complètes)* **Peter Zweifel** ETH Zürich | Sonneggstrasse 5 | 8092 Zürich	

Titre / *Titel*

Base de temps pour réseaux distribués

Description et Objectifs / *Beschreibung und Ziele*

Les réseaux de capteurs distribués – p.ex. sur de grandes machines électriques – nécessitent une synchronisation très exacte afin de réaliser des bandes passantes de mesures élevées. Le projet consiste à interfacer une nouvelle horloge atomique miniature avec une carte FPGA et à réaliser une méthode de synchronisation par Ethernet entre deux nœuds équipés de telles horloges. Il s'agit d'implémenter un protocole PTP en VHDL, en utilisant un circuit PHY standard.

Les points à traiter sont :

— La définition d'une méthode de mesure du délai de transmission et du décalage d'horloge entre deux nœuds reliés par un lien Ethernet point à point

— L'implémentation de cette méthode et le test pratique à l'aide de deux cartes FPGA

— L'étude – par calcul et par simulation – des grandeurs d'influence sur la précision et la résolution de la synchronisation entre les deux nœuds. On s'intéresse en particulier, comment le bruit d'une horloge locale influence la synchronisation avec une horloge centrale 'maître'

— La mesure de la précision de synchronisation obtenue avec la réalisation du protocole PTP en VHDL

— L'intégration d'une horloge atomique miniature comme horloge d'un des deux nœuds de transmission et son effet sur la précision de la synchronisation.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
14.05.2012

Exposition publique / *Ausstellung Diplomarbeiten:*
31.08.2012

Remise du rapport / *Abgabe des Schlussberichts:*
09.07.2012 | 12h00

Défense orale / *Mündliche Verteidigung:*
Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

[1] Etudiant/*Student*: Johan Droz

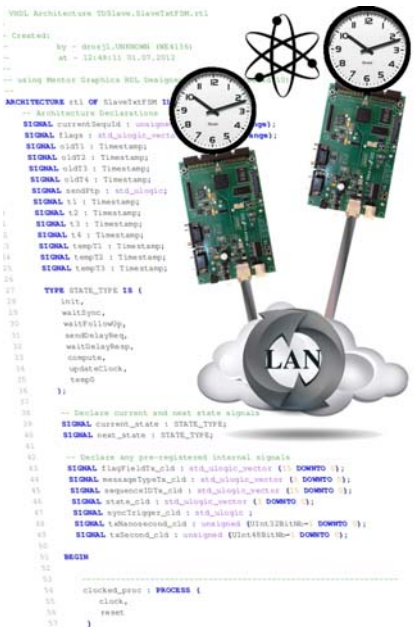
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

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

Base de temps pour réseaux distribués

Diplômant Johan Droz



Travail de diplôme
| édition 2012 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
M. Joseph Moerschell
Joseph.moerschell@hevs.ch

Objectif du projet

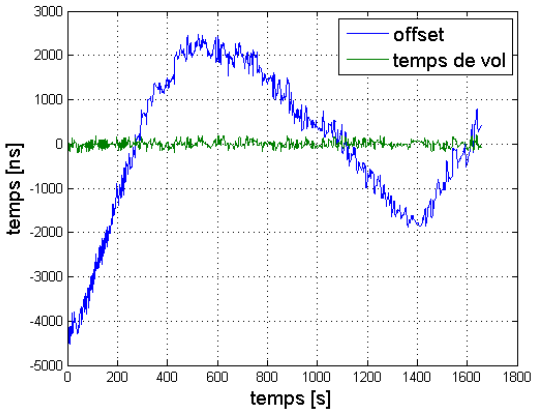
Le projet consiste à interfacer une nouvelle horloge atomique miniature avec une carte FPGA et réaliser une méthode de synchronisation entre deux nœuds Ethernet point à point équipés de telles horloges.

Méthodes | Expériences | Résultats

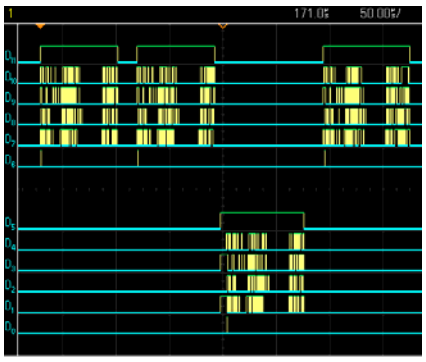
Les applications de mesure et de contrôle emploient de plus en plus des technologies de systèmes distribués comme les réseaux de communication. Lorsque le temps de transmission d'un message n'est pas négligeable par rapport à la période d'échantillonnage, il est nécessaire de synchroniser entre elles des horloges locales dans chaque nœud comportant des capteurs ou actionneurs.

Le protocole PTP est spécialement optimisé pour ces catégories d'applications.

Le système réalisé permet de synchroniser deux horloges à travers un réseau Ethernet point à point, en mesurant le temps de vol des messages, le décalage et la dérive des horloges locales. L'implémentation du protocole PTP se fait au moyen d'un cœur IP programmé en VHDL, et d'un circuit PHY standard. Le but est de descendre à une précision en dessous de la microseconde.



Représentation de la variation de l'offset et du temps de vol calculés selon la norme PTP.



Signaux entre la FPGA et le PHY Ethernet pour un cycle de synchronisation

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

Table des matières

CHAPITRE 1

Introduction

Beaucoup de systèmes électroniques ont un sens du temps. Le temps du système est explicitement disponible quand il est représenté par une horloge. Ceci est souvent nécessaire, spécialement dans les systèmes complexes. Mais toutes les horloges ne sont pas exactes. En effet, la mesure du temps dans les dispositifs électroniques est faite localement à l'aide d'horloges pilotées par des quartz. Les fréquences sont en général très précises, mais on y observe des fluctuations dues à de nombreux facteurs (p. ex. aux variations de température, aux variations d'altitude...) et des différences de phase. Les fluctuations entraînent des dérives plus ou moins importantes (jusqu'à plusieurs secondes en fin de journée) et sont aléatoires. Le temps étant mesuré en comptant les impulsions d'horloge, les petites fluctuations sur la fréquence d'horloge peuvent conduire à des erreurs de temps importantes, dès lors que celles-ci sont accumulées sur un grand nombre d'impulsions. C'est pourquoi il est nécessaire de vérifier si l'écart est tolérable ou si l'horloge doit être corrigée. La communication entre les horloges individuelles est nécessaire pour cela. Deux effets sont mis en évidence lors de la création ou la synchronisation des horloges :

1. Des horloges indépendantes fonctionnent avec un décalage. Pour les synchroniser une correction d'offset est réalisée, c.-à-d. les horloges les moins exactes sont réglées sur l'horloge la plus exacte.
2. Deux horloges différentes ne fonctionnent pas exactement à la même vitesse. Par conséquent, une correction de la dérive est appliquée à l'horloge la moins précise. Sa vitesse est régulée en permanence.

Il existe plusieurs solutions pour synchroniser les horloges distribuées à travers un réseau. Les plus utilisées sont :

– Le *Network Time Protocol* (NTP) et le *Simple Network Time Protocol* (SNTP). Ces protocoles sont largement utilisés dans les réseaux locaux (LAN) et sur Internet. Ils permettent une précision de l'ordre de la milliseconde.

– Des impulsions haute précision : l'horloge mère synchronise les horloges secondaires en leurs envoyant des impulsions de temps haute précision sur des lignes séparées. Cette solution implique un énorme effort de câblage.

– L'utilisation des informations captées dans les signaux radio des satellites GPS. Cette solution requiert pour chaque horloge des récepteurs GPS avec leurs antennes appropriées.

– Le protocole PTP.

La solution choisie est le *Precision Time Protocol* (PTP) décrit dans IEEE 1588. Le but de ce projet est d'implémenter ce protocole en VHDL dans le but de synchroniser deux cartes FPGA à travers un réseau Ethernet.

1.1 Définitions, acronymes, abréviations

1.1.1 Définitions

Communication multicast [1] **:** Modèle de communication dans lequel chaque message PTP envoyé depuis n'importe quel port PTP est capable d'être reçu et traiter par tous les ports PTP concernés.

Dérive : La dérive d'un oscillateur est le changement systématique de sa fréquence au cours du temps. Elle est due au vieillissement, au changement de l'environnement et à d'autres facteurs extérieurs à l'oscillateur tels que des fluctuations d'alimentation, de température, d'accélération...

Échelle de temps [1] **:** Mesure linéaire du temps depuis une époque.

Encapsulation : Procédé qui consiste à inclure les données d'un protocole dans un autre protocole.

Époque [1] **:** Origine d'une échelle de temps.

Horloge maître (Master) : Horloge qui est la source du temps et sur laquelle les autres horloges se synchronisent.

MII : Interface entre la couche MAC et le PHY Ethernet.

Précision [1] **:** Moyenne, sur un ensemble de mesures, de l'erreur du temps ou de la fréquence entre l'horloge testée et une horloge de référence parfaite.

Stabilité [1] **:** Mesure de la façon dans la moyenne varie par rapport à des variables telles que le temps, la température...

Timeout : Mécanisme qui permet d'interrompre une activité qui ne se termine pas dans le délai imparti.

Timestamping[7] **:** Mécanisme qui consiste à associer une date et une heure à un événement, une information ou une donnée informatique.

Vieillissement : Changements de fréquences au cours du temps dus aux changements internes de l'oscillateur.

La figure 1.1 illustre les notions de précision et stabilité.

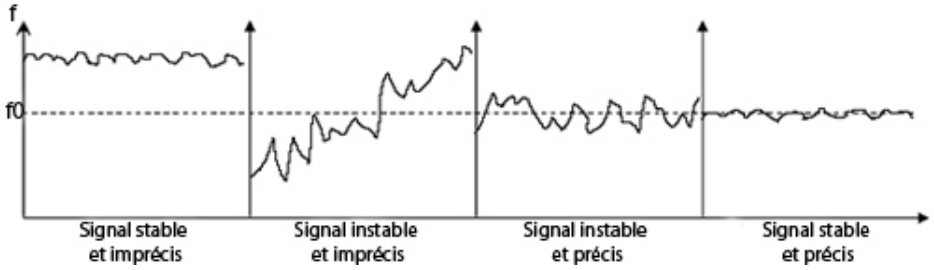


FIGURE 1.1 – Précision vs. Stabilité [8]

1.1.2 Acronymes et abréviations

FIFO	First In First Out
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GPS	Global Positioning System
ID	Identification
IPv4, IPv6	Internet Protocol version 4 / 6
MAC	Media Access Control
MII	Media Independent Interface
NTP	Network Time Protocol
PHY	Physical Layer
PPS	Pulse Per Second
PTP	Precision Time Protocol
SNTP	Simple Network Time Protocol
UDP	User Datagram Protocol
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits

CHAPITRE 2

Precision Time Protocol

Le standard PTP[1] définit un protocole permettant une synchronisation précise des horloges dans les systèmes de mesure et contrôle implémentés avec des technologies telles que la communication réseau, l'informatique locale et les objets distribués. Le protocole est applicable aux systèmes communicants par réseaux locaux supportant les messages *multicast*. Il permet à des systèmes hétérogènes qui incluent des horloges de différentes précisions, résolutions et stabilités de se synchroniser à une horloge maîtresse. Le protocole prend en charge une synchronisation à l'échelle du système dans la gamme submicroseconde avec une utilisation minimale des ressources et de la bande passante du réseau. Le comportement par défaut du protocole permet à des systèmes simples d'êtres installés et de fonctionner sans demander d'effort d'administration. La norme inclut des mappages à UDP/IP, DeviceNet, et une implémentation Ethernet 2 couches. Elle inclut des mécanismes formels pour les extensions de messages, les taux d'échantillonnages élevés, la correction de l'asymétrie, un type d'horloge pour réduire l'accumulation d'erreurs dans les grandes topologies et les spécifications sur la façon d'intégrer des données supplémentaires dans le protocole de synchronisation. Le standard permet une précision de synchronisation meilleure que 1 nanoseconde. Il est prévu de supporter aussi bien les messages *unicast* que *multicast*.

2.1 But

Les applications de mesure et de contrôle emploient de plus en plus des technologies de systèmes distribués comme les réseaux de communications, l'informatique locale et les objets distribués. Beaucoup de ces applications seraient améliorées en ayant un sens du temps précis à l'échelle du système réalisé en ayant des horloges locales dans chaque capteur, actionneur ou autres dispositifs du système. Sans un protocole normalisé pour la synchronisation de ces horloges, il est improbable que les avantages soient réalisés avec les nombreux fournisseurs de composants sur le marché. Les protocoles existants pour la synchronisation d'horloges ne sont pas optimisés pour ces applications. Par exemple, le *Network Time Protocol* (NTP) cible les grands systèmes de calculs distribués avec des exigences de synchronisation dans la gamme de la milliseconde. Le protocole dans le standard IEEE 1588 s'adresse spécifiquement aux besoins des systèmes de mesure, contrôle et opération dans les domaines de test et mesure, automatisation industrielle, systèmes militaires, systèmes de fabrications, systèmes de distributions d'énergie et certains systèmes de télécommunications. Ces applications ont les caractéristiques suivantes :

- Systèmes localisés dans l'espace avec options pour des systèmes plus grands
- Précision dans la gamme microseconde à submicroseconde
- Faible effort d'administration
- Applicables pour appareils haut de gamme et *low-cost*[1]
- Utilisé via un réseau Ethernet, mais aussi avec d'autres réseaux

1. Exigences minimales des performances du processeur et de la bande passante du réseau

– Spécifié comme norme internationale.

2.2 Application

Pour le travail de diplôme, seule la partie de la norme PTP permettant une synchronisation de deux kits FPGA, équipés chacun d'une horloge, et reliés par un lien Ethernet point à point est à implémenter. Le protocole permet de déterminer le décalage entre les horloges, le délai de transmission entre les deux kits ainsi que la dérive des horloges. Un des kits est désigné comme maître, et le second, l'esclave, le prend comme référence pour la synchronisation.

La méthode de mesure du délai de transmission et du décalage d'horloge entre deux nœuds est illustrée en figure 2.1.

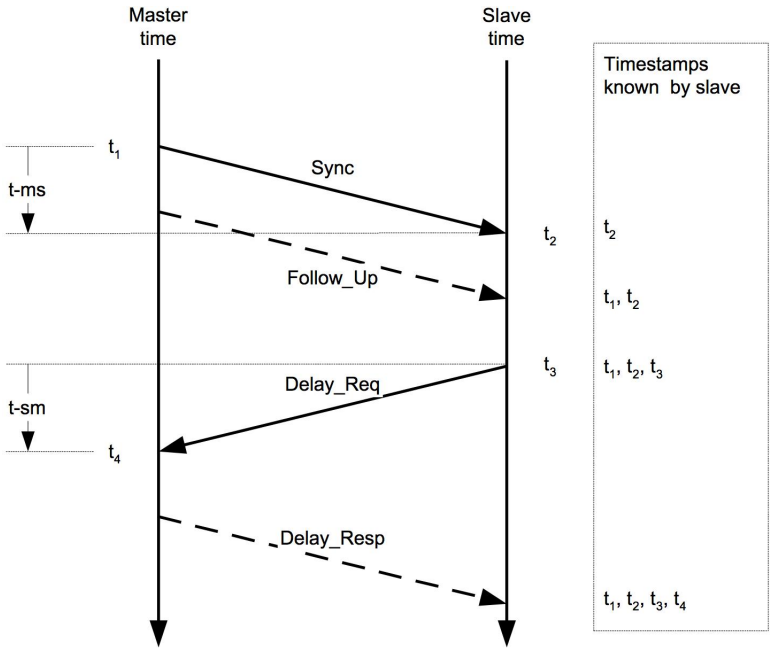


FIGURE 2.1 – Échange de messages pour une synchronisation

Le maître envoie cycliquement un message `Sync` de synchronisation avec une valeur estimée du temps aux esclaves connectés. Parallèlement, le temps exact auquel le message quitte l'émetteur (t_1 figure 2.1) est mesuré aussi précisément que possible par un support hardware dédié. Ensuite le maître envoie ce temps exact de transmission du message `Sync` aux esclaves dans un second message, le message `Follow_Up`[2]. Les esclaves mesurent le temps de réception du message `Sync` (t_2 figure 2.1) et peuvent corriger leur offset avec le maître. Si la ligne de transmission n'avait pas de délai, les deux horloges seraient synchronisées.

La seconde phase de synchronisation, la mesure du délai, détermine le temps de propagation entre le maître et l'esclave. Ce temps est déterminé à l'aide des messages appelés `Delay_Req`[3] et `Delay_Resp`[4]. L'esclave envoie un message `Delay_Req` et note le temps (t_3 figure 2.1) auquel le message quitte l'émetteur. Le maître mesure le temps (t_4 figure 2.1) de réception du message et le fait parvenir à l'esclave dans le message `Delay_Resp`. L'esclave est à ce moment en possession des temps t_1 à t_4 et peut calculer le délai moyen de transmission entre le maître et l'esclave avec l'équation 2.1.

2. Message de suivi.
3. Delay Request.
4. Delay Response.

Si d est le temps de vol du message, et o l'offset de l'horloge :
$d + o = t_2 - t_1$ et $d - o = t_4 - t_3$

$$d = Délai\ moyen = \frac{[(t_2 - t_1) + (t_4 - t_3)]}{2} = \frac{[(t_2 - t_3) + (t_4 - t_1)]}{2} \tag{2.1}$$

Remarque :
Le cas décrit en figure 2.2 présente un réseau avec des asymétries dans les délais de transitions, l'équation 2.1 donne comme résultat le délai moyen.

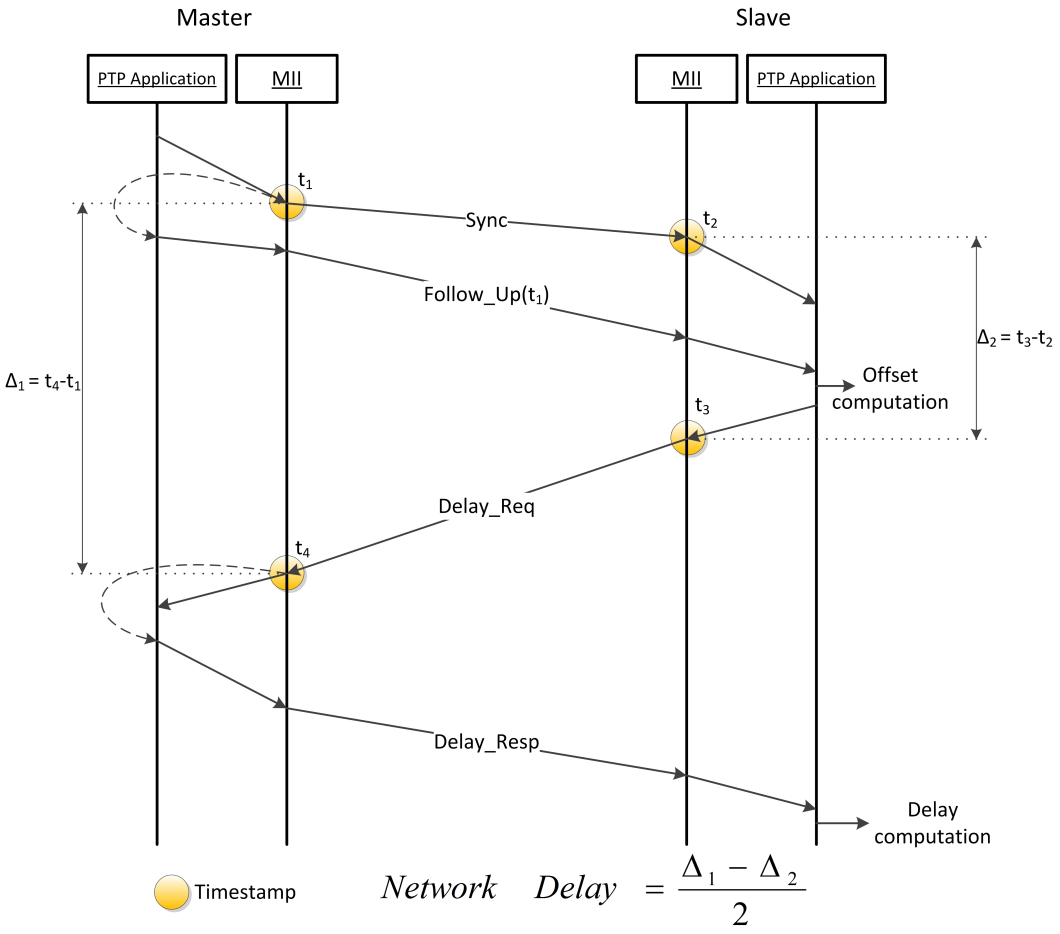


FIGURE 2.2 – Envoi des messages

La figure 2.2 présente le principe par lequel les messages sont envoyés. L'application définit le message PTP à envoyer et donne simplement l'ordre d'envoi (début d'une flèche en provenance de l'application). Un message est réellement envoyé lorsqu'il quitte le MII. Le temps entre la couche application et le MII comprend le temps que met le message pour être encapsulé [5] dans les protocoles UDP, IP et MAC et le temps de résidence du message dans la FIFO d'émission. Ce temps est aléatoire (cf. figure 2.2).

Des messages `Sync` consécutifs permettent de déterminer la dérive de l'horloge. La dérive est compensée en accélérant ou ralentissant l'oscillateur.
La dérive de l'horloge est calculée entre les cycles k et k+1 avec l'équation 2.2.

5. Voir section 3.4

$$Dérive = \frac{\Delta_2 - \Delta_1}{\Delta_1} \tag{2.2}$$

avec $\Delta_1 = t_1^{k+1} - t_1^k$ et $\Delta_2 = t_2^{k+1} - t_2^k$

2.3 Échelle de temps et époque dans PTP [1]

Les caractéristiques du temps sont déterminées par le dispositif maître. Le maître détermine la vitesse à laquelle le temps avance ainsi que l'époque, c.-à-d. l'origine de l'échelle de temps. Le temps instantané est représenté par une variable qui décompte le nombre de secondes depuis l'époque.

Par exemple, le délai de remise de ce rapport est 1 341 828 000 (époque Unix [5]). Cette valeur exprime le nombre de secondes écoulées depuis l'époque Unix (1er janvier 1970 00 :00 :00) ce qui correspond à la date du 9 juillet 2012 à 12 :00 :00.

2.3.1 UTC [6]

Le temps universel coordonné (UTC) est une échelle de temps adoptée comme base du temps civil international par la majorité des pays du globe. UTC est mis en œuvre par une série d'horloges atomiques et constitue la base de chronométrage pour d'autres échelles de temps d'usage commun. La représentation UTC est spécifiée sous la forme AAAA - MM - JJ pour la date et hh :mm :ss pour le temps de chaque jour.

2.3.2 Source de temps standard

Il existe deux sources de temps standard d'un intérêt particulier pour les systèmes PTP dont l'application requiert un temps UTC traçable.

La première est l'ensemble des systèmes mettant en œuvre le protocole NTP, largement utilisé dans la synchronisation de systèmes informatiques partout dans le monde. Un ensemble de serveurs NTP auxquels les clients NTP sont synchronisés est maintenu. La précision du temps UTC depuis des systèmes NTP est habituellement dans la gamme de la milliseconde. L'époque NTP est le 1er janvier 1900, à 00 :00 :00. NTP représente les secondes avec un entier 32 bits non signé.

La seconde source d'intérêt est le système GPS. La précision du temps UTC depuis le système GPS est habituellement comprise entre 10 ns et 100 ns. Les systèmes de transmission GPS [2] représentent le temps comme `{GPS Weeks, GPS SecondsInLastWeek}`, c.-à-d. le nombre de semaines depuis l'époque GPS et le nombre de secondes depuis le début de la semaine courante. Le temps UTC peut être calculé en utilisant les données incluses dans les transmissions GPS. L'époque GPS débute le 6 janvier 1980 à 00 :00 :00.

CHAPITRE 3

Conception

La configuration du système est représentée en figure 3.1. Les deux cartes FPGA sont connectées via un lien Ethernet point à point. L'esclave est capable de déterminer et de corriger l'offset et la dérive de son horloge locale par rapport à celle du maître en échangeant périodiquement une série de trames Ethernet avec celui-ci. À chaque période, l'offset et la dérive sont calculés et envoyés à un PC via une liaison RS-232. Sur le PC, une application réceptionne les données, les traite et les enregistre au format voulu. Ce principe permet de collecter et gérer facilement les données de tests de longue durée.

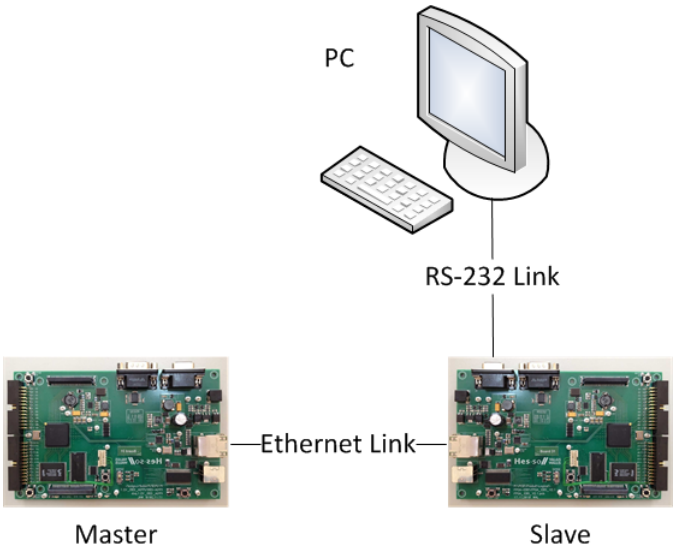


FIGURE 3.1 – Configuration matérielle

3.1 Outils de développements

3.1.1 Matériel

Le kit FPGA utilisé pour le développement du projet est le kit FPGA-EBS v2.1 [1] de la HES-SO. Les ressources utilisées du kit et leurs agencements sont décrits en annexe A page 33.

3.1.2 Logiciels

Développement FPGA

Le tableau 3.1 répertorie les programmes utilisés pour le développement FPGA.

1. http://wiki.hevs.ch/uit/index.php5/Hardware/FPGAEBS

Logo	Programme	Version	Utilisation
	HDL Designer	2009.2 (Build 10)	Développement FPGA
	ModelSim	SE 6.6a (Révision 2010.03)	Simulation
	Xilinx ISE Project Navigator	12.1 (nt)	Synthèse, configuration du périphérique cible avec le programmeur

TABLE 3.1 – Programmes utilisés pour le développement FPGA

Utilitaires PC

Le tableau 3.2 répertorie les programmes utilitaires utilisés durant le projet.

Logo	Programme	Version	Utilisation
	Colasoft Packet Builder	1.0	Composition et envoi de paquets dans un réseau Ethernet
	Matlab R2009a	7.8.0.347	Traitement et visualisation des données
	Wireshark	1.6.0	Capture et analyse des trames Ethernet échangées entre les deux cartes FPGA

TABLE 3.2 – Utilitaires

Développement PC

Le tableau 3.3 répertorie les programmes utilisés pour le développement sur PC.

Logo	Programme	Version	Utilisation
	QtCreator	2.2.0	Création d'une application qui réceptionne, traite et enregistre les données reçues par RS-232

TABLE 3.3 – Programmes utilisés pour le développement sur PC

3.2 Vue d'ensemble du projet FPGA

Le but du projet est d'implémenter en VHDL la partie du protocole PTP qui permet la synchronisation de deux nœuds reliés par un lien Ethernet. Il y a deux unités différentes à concevoir : le maître et l'esclave. Bien que différents, ils ont en commun une grande partie du code VHDL. La majeure différence entre le maître et l'esclave réside dans la partie application. La section 3.8 est décomposée en deux sous-sections, décrivant respectivement le maître et l'esclave.

Le code VHDL est réalisé avec le programme HDL Designer en mode schématique. Les différents composants du projet sont représentés sous forme de blocs imbriqués et reliés par des signaux et des bus. Les machines d'états sont également réalisées en mode graphique.

La figure 3.2 présente l'architecture du projet. Elle expose les différentes librairies utilisées ainsi que les différents niveaux d'imbrication des blocs. Les librairies `Board`, `Ethernet`, `RS232` et `TD_TimeBase4DiS` contiennent les composants communs au maître et à l'esclave.

Les librairies `Ethernet` et `RS232` ont été fournies au début du projet.

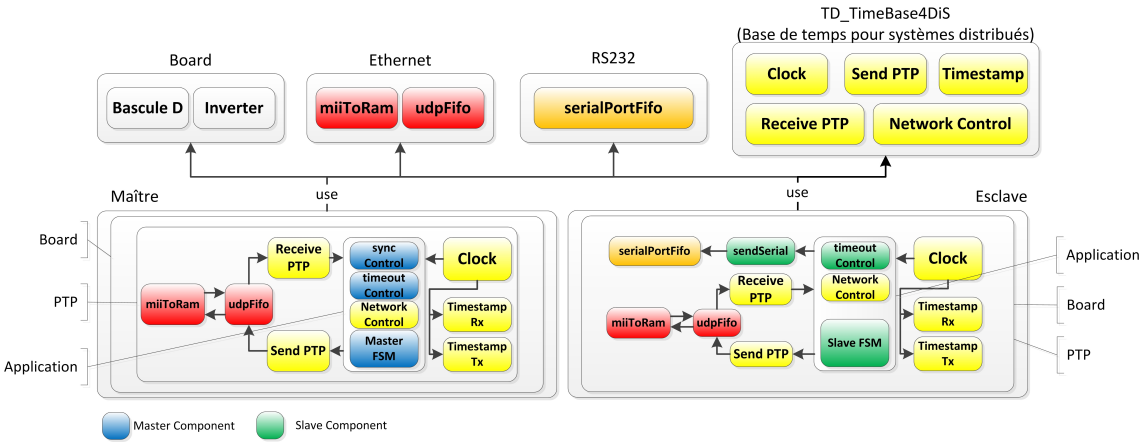


FIGURE 3.2 – Structure du projet

3.2.1 Maître

Le plus haut niveau de la vue graphique (Top-Level) de l'élément maître est donné en annexe B.1.1 page 35. Le bloc central (n° 5) contient la vue pages 37 et 38.

3.2.2 Esclave

Le « Top-Level » de l'élément esclave est donné en annexe C.1.1 page 49. Le bloc central (n° 5) contient la vue pages 51 à 53.

3.3 Communication RS-232

La communication RS-232 est utilisée pour transmettre des données depuis l'esclave à un ordinateur. La logique utilisée est visible en annexe C.1.2 page 51. Le bloc n° 2 envoie les données contenues dans sa FIFO interne sur le port série. Le bloc n° 3 s'occupe de la gestion des données envoyées. Le signal `syncTrigger` provient de l'application et indique que les données sont valides. Elles sont alors copiées dans la FIFO du bloc n° 2.
Les données transmises sont répertoriées dans le tableau 3.4. Elles sont envoyées à la fin de chaque cycle de synchronisation, autrement dit chaque 2 secondes, lorsque l'esclave connaît les temps t_1 à t_4.

Ordre	Donnée	Taille en bits	Format
1	Offset s	48	non signé
2	Offset ns	32	
3	Dérive	32	signé
4	t_1 s	48	non signé
5	t_1 ns	32	
6	t_2 s	48	
7	t_2 ns	32	
8	t_3 s	48	
9	t_3 ns	32	
10	t_4 s	48	
11	t_4 ns	32	
12	temps de vol s	52	signé
13	temps de vol ns	36	

TABLE 3.4 – Données transmises au PC

Annexe C.1.2 page 51 blocs n° 2 & n° 3 : Vue graphique de la liaison RS-232
Annexe D.5 page 72 : Code VHDL du bloc de gestion des données RS-232

3.4 Communication Ethernet/IP/UDP

La norme IEEE 1588 décrit le transport du protocole PTP par UDP/IPv4. La communication est réalisée en encapsulant ces protocoles comme le présente la figure 3.3.

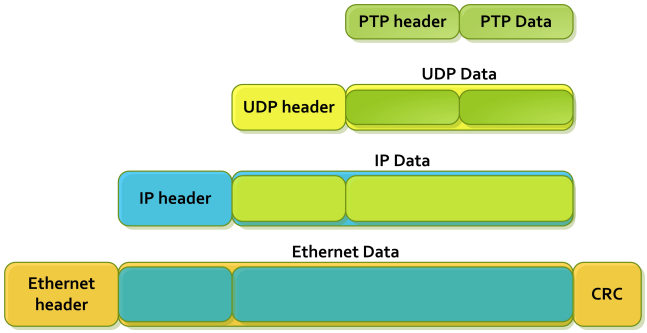


FIGURE 3.3 – Encapsulation des protocoles

La logique programmée fonctionne selon le principe décrit en figure 3.4. Un canal d'émission achemine les données de l'application au PHY. Chaque couche ajoute un en-tête aux données reçues avant de les propager à la couche suivante.

Pour la réception, le principe est inversé. Le canal de réception achemine les données en provenance du PHY jusqu'à l'application. À chaque couche l'en-tête du protocole est lu et les données sont transmises à la couche suivante uniquement si les données sont destinées à l'application. Dans le cas contraire, elles sont ignorées. Les adresses MAC, IP et ports UDP sont répertoriés dans le tableau 3.5.

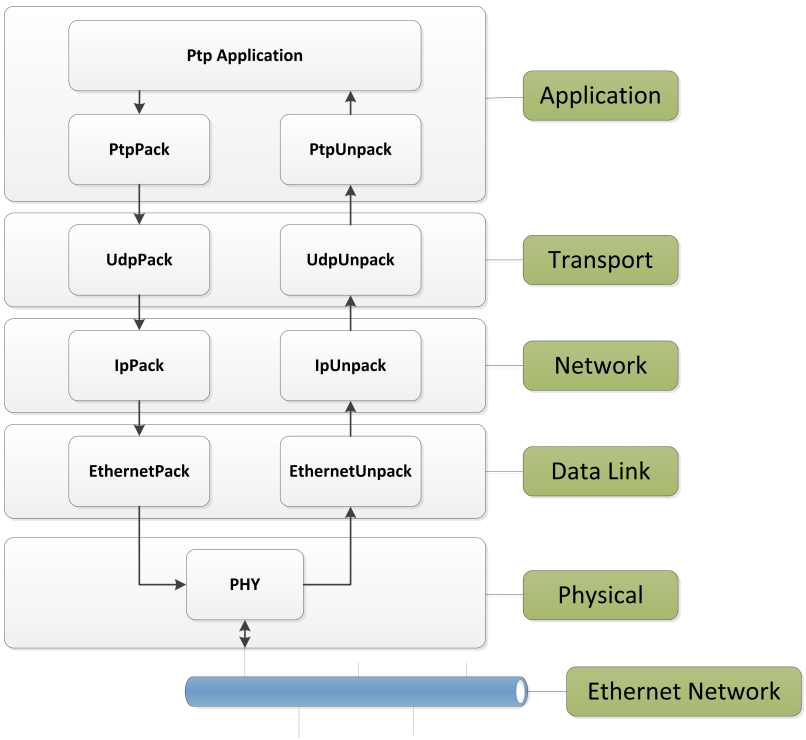


FIGURE 3.4 – Encapsulation / désencapsulation des protocoles

Dispositif	Adresse MAC	Adresse IP	Port UDP
Maître	E4 :AF :A1 :39 :10 :00	169.254.240.92	319
Esclave	E4 :AF :A1 :39 :10 :01	169.254.240.91	319

TABLE 3.5 – Paramètres de la connexion

Annexe B.1.2 page 37 blocs n° 2 & n° 3 : Vue graphique de la communication Ethernet (maître)
Annexe C.1.2 page 52 blocs n° 4 & n° 5 : Vue graphique de la communication Ethernet (esclave)

3.5 Couche PTP [1]

La couche PTP est encapsulée dans la couche UDP (cf. figure 3.3). Un message PTP est composé d'un en-tête (cf. tableau 3.6) qui est commun à tous les messages PTP, ainsi que de données qui varient en fonction du type des messages. La partie du protocole utilisée pour déterminer l'offset et le temps de vol utilise quatre différents types de messages représentés en figure 2.1 page 10.

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
transportSpecific				messageType				1	0
reserved				versionPTP				1	1
messageLength								2	2
domainNumber								1	4
reserved								1	5
flagField								2	6
correctionField								8	8
reserved								4	16
sourcePortIdentity								10	20
sequenceId								2	30
controlField								1	32
logMessageInterval								1	33

TABLE 3.6 – En-tête des messages PTP

Le format des messages `Sync` et `Delay_Req` est décrit dans le tableau 3.7. Ils sont composés de l'en-tête suivi du temps local estimé.

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
En-tête PTP (cf. tableau 3.6)								34	0
originTimestamp								10	34

TABLE 3.7 – Messages `Sync` et `Delay_Req`

La donnée qui suit l'en-tête du message `Follow_Up` (cf. tableau 3.8) est le temps exact auquel le message `Sync` a été envoyé.

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
En-tête PTP (cf. tableau 3.6)								34	0
preciseOriginTimestamp								10	34

TABLE 3.8 – Message `Follow_Up`

Le champ `receiveTimestamp` du tableau 3.9 est le temps exact auquel le message `Delay_Req` est reçu. Le champ `requestingPortIdentity` n'est pas utilisé pour ce projet, il est donc mis à une valeur par défaut de 0.

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
En-tête PTP (cf. tableau 3.6)								34	0
receiveTimestamp								10	34
requestingPortIdentity								10	44

TABLE 3.9 – Message `Delay_Resp`

3.5.1 Implémentation

Émission

Le bloc n° 4 de l'annexe B.1.2 page 37 analyse les messages PTP, récupère les données utiles et les transmets à la couche application où elles sont traitées.

Réception

Pour l'émission des données, le principe est inversé. L'application fournit les données ainsi que la valeur des différents champs de l'en-tête puis donne l'ordre d'envoyer les données. Le bloc n° 5 de l'annexe B.1.2 créé le message PTP en conséquence et le copie dans la FIFO d'émission.

Annexe B.1.2 page 37 blocs n° 4 & n° 5	: Vue graphique de la couche PTP (maître)
Annexe C.1.2 page 52 blocs n° 4 & n° 5	: Vue graphique de la couche PTP (esclave)
Annexe D.1 page 58	: Code VHDL du Package PTP
Annexe D.2 page 62	: Code VHDL du bloc de réception (n° 4)
Annexe D.3 page 65	: Code VHDL du bloc d'émission (n° 5)

3.6 Horodatage [7]

L'horodatage [2] est un mécanisme qui consiste à associer une date et une heure à un événement, une information ou une donnée informatique. Il a généralement pour but d'enregistrer l'instant auquel une opération a été effectuée. La valeur représentant la date et l'heure est appelée *timestamp* ou « horodatage ». Le *timestamp* a le format (en bit) illustré en figure 3.5. Il représente un temps positif par rapport à l'époque [3].

FIGURE 3.5 – Format du *timestamp*

Le membre `secondsField` est la partie entière du *timestamp* qui a pour unité des secondes. Le membre `nanosecondsField` est la partie fractionnaire et a pour unité des nanosecondes. Le champ `nanosecondsField` est toujours inférieur à 10^9.

Par exemple :
+2.000 000 001 secondes est représenté par `secondsField` = $0000\,0000\,0002_{16}$ et `nanosecondsField` = $0000\,0001_{16}$

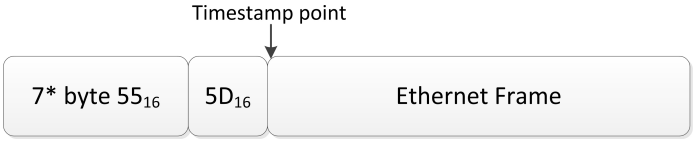


FIGURE 3.6 – Préambule Ethernet

Un *timestamp* est généré à chaque émission et réception des messages `Sync` et `Delay_Req` (cf. figure 2.2 page 11). Le point de référence dans le message pour le *timestamping* est marqué en

2. En anglais *timestamping*
3. Voir section 2.3 page 12

figure 3.6. Il est situé directement après le préambule Ethernet [4]. Pour une précision maximale, chaque opération de *timestamping* doit être effectuée le plus près possible du PHY. La figure 3.7 en illustre la raison. L'exécution du *timestamp* entre la couche PHY et MAC permet de s'affranchir des pertes de précision dues au temps de traitement du message dans les couches IP, UDP et PTP.
Dans ce projet, le *timestamp* est généré par une assistance hardware entre la couche MAC et la couche PHY, soit au point A de la figure 3.8.

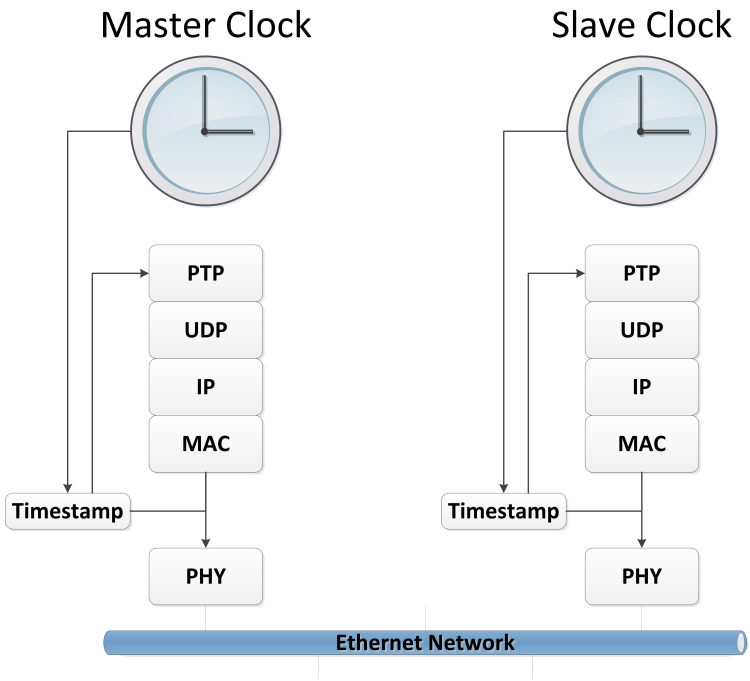


FIGURE 3.7 – *Timestamp* et protocoles

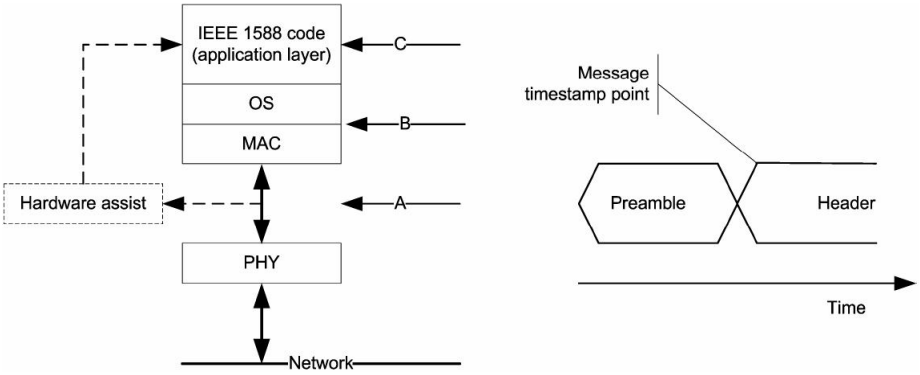


FIGURE 3.8 – Modèle pour la génération des *timestamp*

3.6.1 Implémentation

Le comportement de l'*unité de timestamping* est décrit par la machine d'états de la figure 3.9. Chaque nœud Ethernet en comprend deux, une pour l'émission et une pour la réception.

La machine est par défaut dans l'état `idle`. Elle transite dans l'état `waitForD` lors du commencement d'une trame. À la fin du préambule Ethernet, annoncé par la donnée D_{16}, l'état

4. Le préambule Ethernet contient 7 fois le byte 0x55 suivi du byte 0x5D.

change en `timestamp`. Le *timestamp* est exécuté au prochain flanc d'horloge Ethernet et l'état change en `waitEndOfFrame`. À la fin de la trame, la machine d'états se retrouve dans l'état par défaut `idle`.

Il se peut que des données indésirables soient captées durant un court laps de temps au début de la trame. La machine d'états transite alors dans l'état `waitFor5`, où les données parasites sont ignorées.

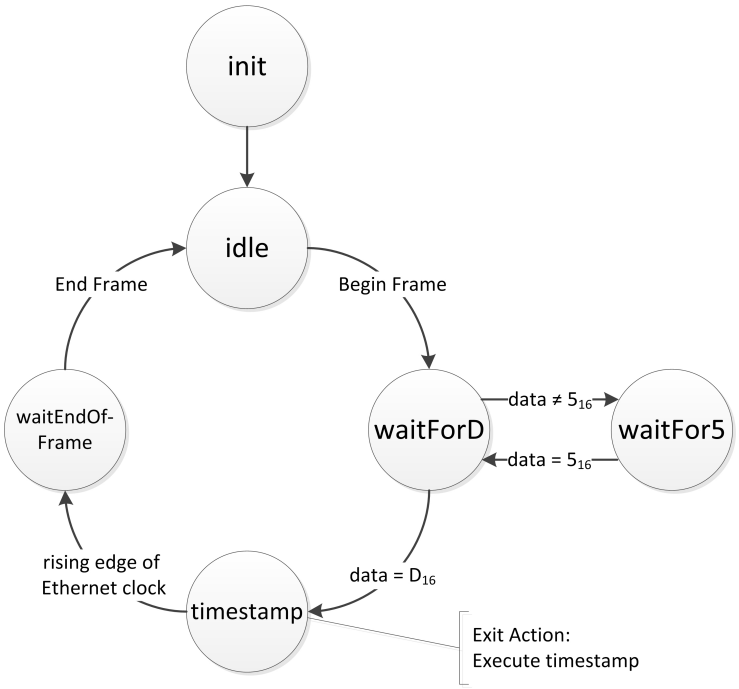


FIGURE 3.9 – Machine d'états de la génération des *timestamp*

Annexe B.1.2 page 40 : Vue graphique de l'*unité de timestamping*
Annexe B.1.2 page 41 : Machine d'états HDL Designer de l'*unité de timestamping*

3.7 Horloge locale

L'horloge locale est une entité qui maintient à jour l'échelle de temps du système et génère un signal 1 PPS. Elle incrémente la valeur du temps instantané de $\frac{1}{f_{clock}}$ à chaque coup d'horloge. L'horloge locale est également capable de subir des mises à jour du temps et des changements de vitesse.

3.7.1 Implémentation

L'horloge locale est implémentée suivant le principe illustré en figure 3.10.

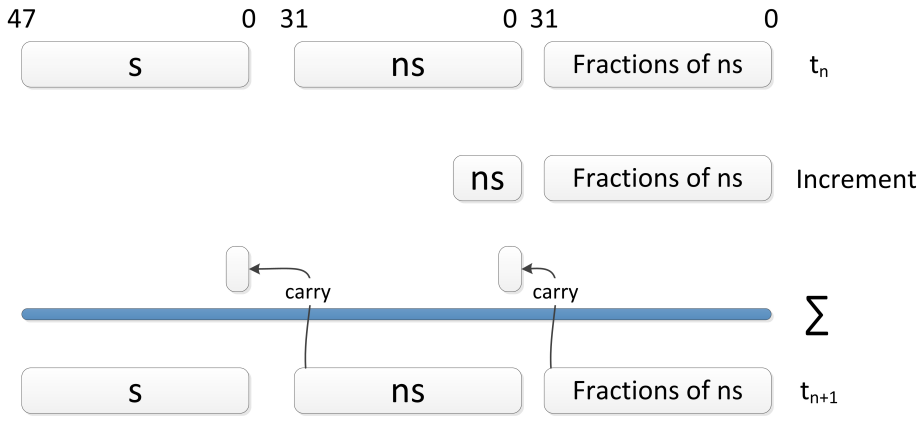


FIGURE 3.10 – Principe de fonctionnement de l'horloge locale

Annexe B.1.2 page 38 bloc n° 7 : Vue graphique de l'horloge (maître)
Annexe C.1.2 page 53 bloc n° 9 : Vue graphique de l'horloge (esclave)
Annexe D.4 page 69 : Code VHDL de l'horloge

3.8 Application

La partie application définit le comportement du système. C'est cette partie qui définit si un nœud Ethernet agit comme maître ou comme esclave. Les comportements du maître et de l'esclave reflètent le principe illustré en figure 2.2 page 11.

3.8.1 Maître

Le comportement de l'entité maître est décrit par la machine d'états de la figure 3.11. La machine est par défaut dans l'état `idle`. Elle commute dans l'état `sendSync` à la réception du signal `Start synchronisation` qui lui indique de démarrer un cycle de synchronisation. Un message `Sync` est envoyé, et la machine d'états transite dans l'état `sendFollowUp`. Un message `FollowUp` est envoyé, et l'état passe à `waitDelayReq`. Lors de la réception d'un message `DelayReq`, la valeur du champ `sequenceID` est controlée. Si la valeur du champ correspond à celle du cycle en cours, la machine passe dans l'état `sendDelayResp`, où un message de type `delayResp` est transmis à l'esclave. Dans le cas contraire, le message reçu n'est pas valide, et la machine d'états abandonne le cycle de synchronisation en cours, pour retourner dans l'état `idle`. Un *timeout* est ajouté dans l'état `waitDelayReq`. Il se peut que des messages soient perdus. Le *timeout* évite que le maître attende indéfiniment un message, bloquant ainsi la machine d'états.
A la fin du cycle, la machine d'états se retrouve dans l'état par défaut `idle`.

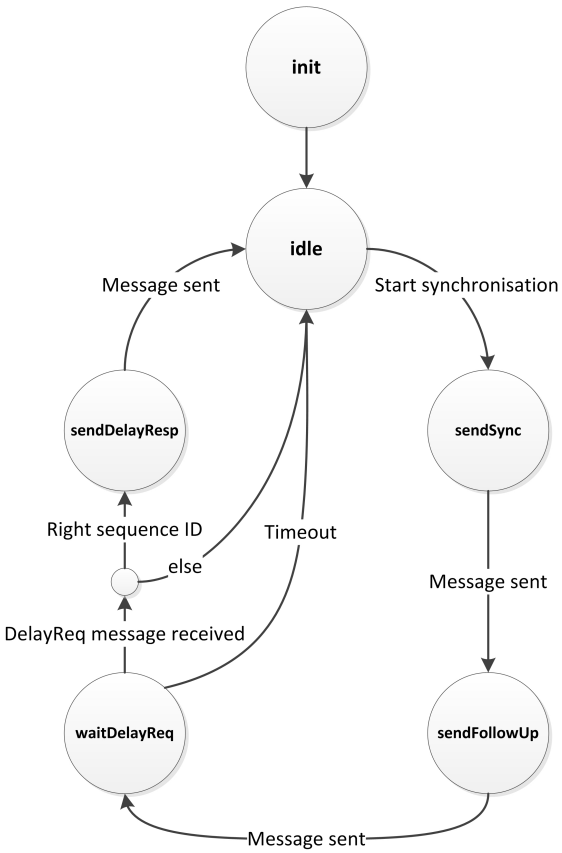


FIGURE 3.11 – Machine d'états du maître

Annexe B.1.2 page 38 bloc n° 6 : Vue graphique de l'application (maître)
Annexe B.1.3 page 43 : Vue graphique du bloc application
Annexe B.1.3 page 47 : Machine d'états HDL Designer de l'application

3.8.2 Esclave

Le comportement de l'esclave est illustré par la machine d'états de la figure 3.12. La machine est par défaut dans l'état `waitSync` où elle attend que le maître initie un cycle de synchronisation. Un message `Sync` marque le début du cycle et le passage à l'état `waitFollowUp`. L'esclave a préalablement stocké en mémoire le temps exact auquel il a reçu le message `Sync`. A la réception du message `FollowUp`, l'esclave enregistre le temps contenu dans ce message, temps qui représente l'instant exact où le message `Sync` a quitté le dispositif maître. Si le champ `sequenceID` du message reçu est celui du cycle en cours, l'état suivant est `sendDelayReq`, sinon le cycle en cours est abandonné. Un message `delayReq` est envoyé au maître, l'instant exact de son départ est stocké et l'état passe à `waitDelayResp`. La réception du message `delayResp` avec le champ `sequenceId` correct entraîne le passage à l'état `compute` où l'offset et le temps de vol sont calculés. Le prochain état se nomme `updateClock`. Les corrections sont passées au régulateur qui s'occupe de l'asservissement de l'horloge. À la fin du cycle, la machine d'états se retrouve dans l'état par défaut `waitSync`. Des *timeout* ont été ajoutés dans les états où un message spécifique est attendu.

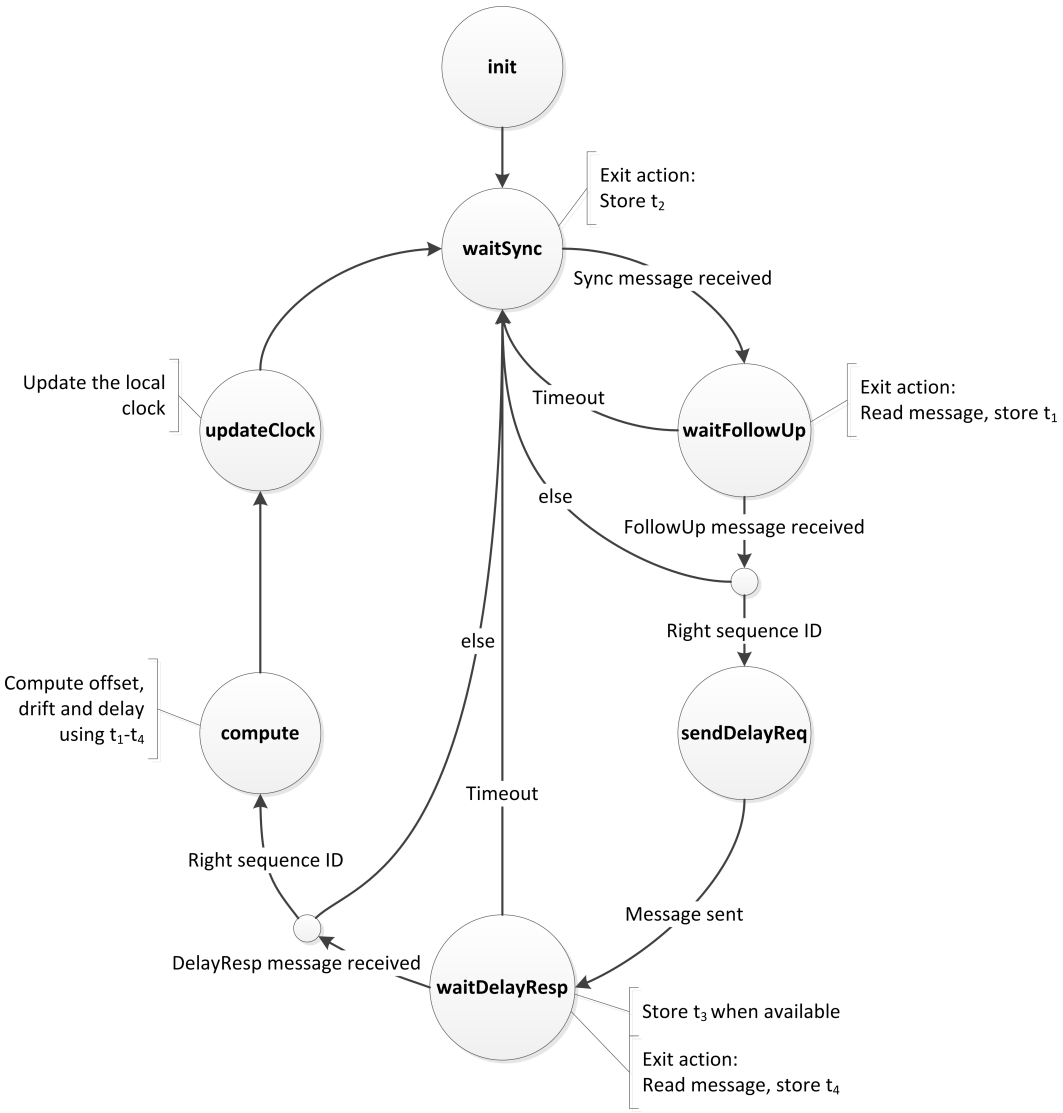


FIGURE 3.12 – Machine d'états de l'esclave

Annexe C.1.2 page 53 bloc n° 8 : Vue graphique de l'application (esclave)
Annexe C.1.3 page 55 : Vue graphique du bloc application
Annexe C.1.3 page 57 : Machine d'états HDL Designer de l'application

3.9 Régulateur

Le régulateur permet d'appliquer des corrections à l'horloge esclave. La valeur de la correction est calculée par l'application à chaque cycle de synchronisation dans l'état `compute` de la figure 3.12.
Par manque de temps, le régulateur n'a pas été implémenté mais il a déjà été dimensionné (cf. annexe F.1 page 79) et simulé par Mr. Joseph Moerschell. Le bilan tiré des résultats des simulations est que l'objectif de synchroniser deux horloges avec une précision submicroseconde est possible pour ce projet.

Annexe F.1 page 79 : Dimensionnement du régulateur
Annexe F.2 page 81 : Simulations du régulateur

3.10 Application PC Qt

L'esclave envoie via RS-232 les données décrites dans le tableau 3.4 page 16 par intervalle de deux secondes. Les données envoyées depuis la FPGA sont sous forme de caractères ASCII, délimitées par des caractères spéciaux (cf. figure 3.13 fenêtre n° 2). Il est nécessaire de réceptionner, traiter et enregistrer les données.

La solution choisie pour réaliser cette tâche est le développement d'une application en C++ avec l'environnement Qt. Cette solution a été choisie pour les raisons suivantes :

- Maîtrise de l'environnement Qt
- L'environnement est déjà installé sur les places de travail de la HES-SO
- Un projet de base est déjà disponible

Le projet de base permet de configurer une liaison sérielle (*Port, BaudRate, DataBits, Parity, StopBits, Timeout et QueryMode*) et d'envoyer et recevoir des caractères. L'interface graphique est présentée en figure 3.13. Elle comporte les éléments suivants :

1. Paramétrage de la connexion sérielle.
2. Fenêtre de réception des données.
3. Fenêtre d'émission des données.

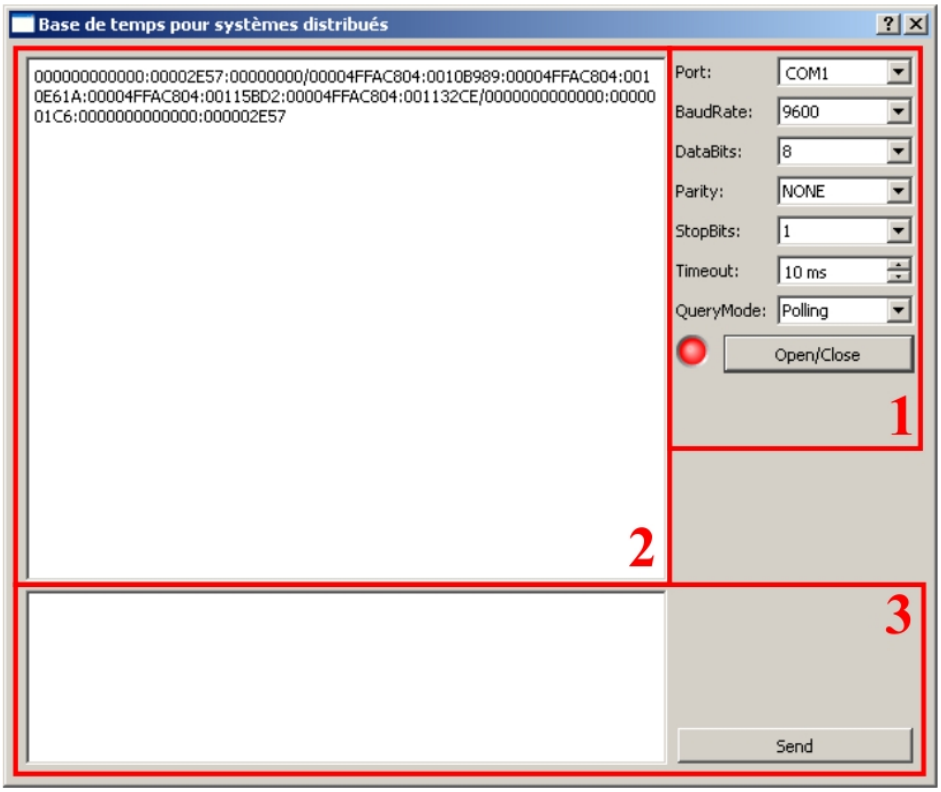


FIGURE 3.13 – Interface graphique de l'application Qt

3.10.1 Implémentation

Le projet Qt préexistant permet d'envoyer et recevoir des caractères par RS-232 et de les afficher dans des objets de types `QPlainTextEdit`.

Seule la classe `Dialog` est modifiée.

```
1 private:
2     QFile *file;
```

```
3     QString *buffer;
4     int *lineNumber;
```

Listing 3.1 – Extrait du fichier `dialog.h`

Un *buffer* d'entrée est ajouté (cf. listing 3.1). Les caractères reçus y sont copiés. Le fichier « mesures.csv » est créer dans le constructeur de la classe `Dialog` (cf. listing 3.2). Un fichier « csv » est un fichier tableur, contenant des données sur chaque ligne séparées par un « ; ». Ce format a été choisi pour les raisons suivantes :

- Simplicité du format
- Possibilité de visualiser le fichier avec Microsoft Excel
- Possibilité d'automatiser les calculs, vérifications dans Excel
- Possibilité d'exporter le fichier vers un format reconnu de Matlab

```
1  // Creation du fichier ou les donnees sont enregistrees.
2     QString filename ="Y://drozj1/mesures.csv";
3     file = new QFile(filename);
4
5     // Ouverture du fichier
6     if ( file->open(QIODevice::WriteOnly | QIODevice::Text))
7     {
8         // Ecriture des en-entetes des colonnes
9         QTextStream stream( file );
10        stream << "Offset seconde;offset nanoseconde;derive;t1 sec;t1 nano;"
11               << "t2 sec;t2 nano;t3 sec;t3 nano;t4 sec;t4 nano;DelayS;DelayN;"
12               << "OffsetSec;OffsetNano;;t1;t2;t3;t4;delay" << endl;
13        file->close();
14    }
15
16    buffer = new QString(50); // Creation du buffer d'entree
17    buffer->clear();          // Initialisation du buffer d'entree
18    lineNumber = new int;     // Compte le nombre de lignes du fichier
19    *lineNumber = 1;          // Le fichier contient une ligne (en-tete)
```

Listing 3.2 – Extrait du constructeur de la classe `Dialog`

Le listing 3.3 présente l'extrait de code C++ qui s'occupe de la gestion des données. Les tâches effectuées sont :

1. Lecture du port RS-232
2. Copie des données dans le *buffer*
3. Lorsque la réception est terminée, les données sont converties d'hexadécimales à long long (non signé)
4. Les données sont converties de la forme non signée à signée
5. Les données sont copiées dans le fichier « mesures.csv »

```
QString receivedData;
if (port->bytesAvailable())
{
      receivedData  = port->readAll();     // Read data
      *buffer += receivedData;             // Copy data into buffer
      if (buffer->contains("%"))           // Reception complete
      {
          if ( file->open(QIODevice::WriteOnly | QIODevice::Text | QIODevice
              ::Append))
          {
              QTextStream stream( file );
              buffer->remove("%");

              // Split data
              QStringList sep = buffer->split("/");
              QStringList list = sep.at(0).split(":");
              QStringList time = sep.at(1).split(":");
              QStringList delay = sep.at(2).split(":");

              // count the lines in the file
              *lineNumber++;

              for (int i = 0; i < list.size(); ++i)
              {
                  // Convert data from hexadecimal to unsigned long long
                  QString str = "0x"+list.at(i);
                  bool ok;
                  long long appId = str.toLongLong(&ok,16);

                  // ...
                  // Convert data from unsigned to signed
                  // ...
              }

              // ...
              // Write data to file
              // ...

              file->close();  // Close file
          }
          buffer->clear();     // Clear input buffer
      }
```

Listing 3.3 – Extrait du fichier `Dialog.c`

CHAPITRE 4

Tests & résultats

4.1 Timestamping

L'objectif du test est de vérifier que les *timestamp* sont exécutés aux bons moments, soit directement après le préambule Ethernet. La vérification est réalisée en mesurant les signaux dont se servent la FPGA et le PHY Ethernet pour communiquer ainsi que le signal qui indique que l'opération de *timestamping* est exécutée.

4.1.1 Résultats

La figure E.1 page 75 est une capture des signaux décrits dans le tableau E.1 page 76 lors de la réception d'une trame. On constate que le signal D_{12} annonce l'exécution du *timstamp* directement après le dernier octet du préambule Ethernet.

La figure E.2 page 76 est une capture des signaux décrits dans le tableau E.1 page 76 au commencement de l'émission d'une trame. On constate que le signal D_{12} annonce l'exécution du *timstamp* directement après le dernier octet du préambule Ethernet.

Le test est un succès. Les mesures effectuées indiquent que les *timestamp* sont exécutés exactement à l'instant voulu.

4.2 Échange de messages

L'offset et la dérive sont déterminés par la méthode décrite à la section 2.2 page 10. Le maître et l'esclave s'échangent cycliquement la série de messages représentés en figure 2.1. Le but de ce test est de vérifier que les messages sont bien formés, c.-à-d. que les en-têtes des différents protocoles contiennent les bonnes valeurs. L'ordre et le type des messages sont également contrôlés.

4.2.1 Résultats

Le test est réalisé avec la configuration matérielle illustrée en figure 4.1. Le maître et l'esclave sont connectés via un HUB Ethernet. De cette manière, un PC équipé de Wireshark peut capturer les paquets échangés sur le réseau. L'esclave est également relié au PC par une connexion RS-232 afin de transmettre et d'enregistrer les données importantes telles que les *timestamping*, l'offset et la dérive calculés. Un programme développé avec Qt réceptionne les données sérielles, les met en forme et les enregistre.

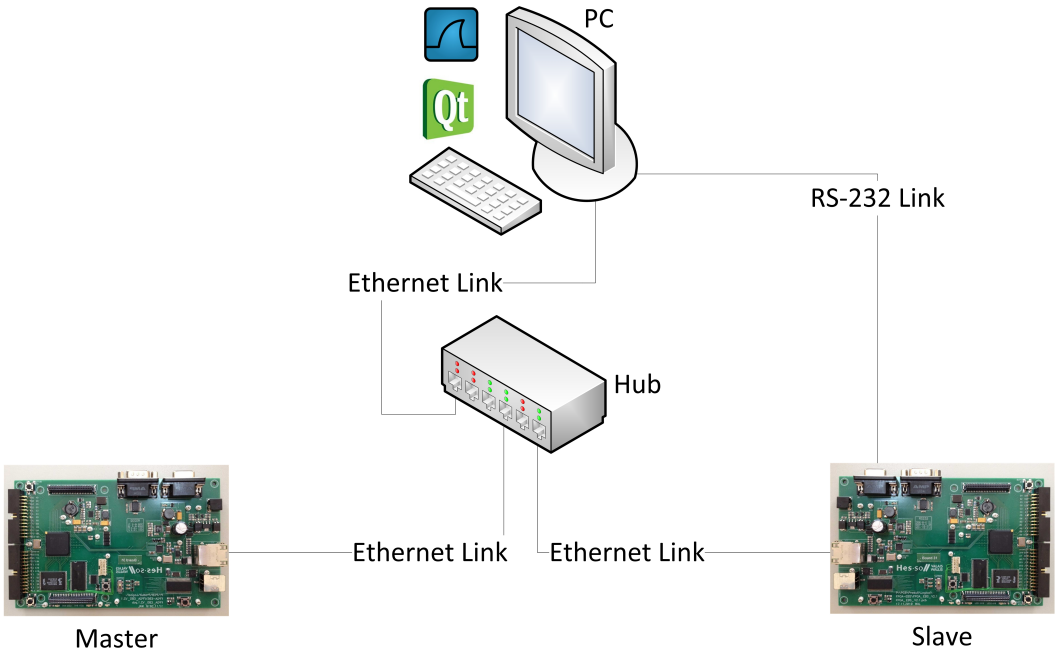


FIGURE 4.1 – Capture des messages

La figure E.3 représente les paquets capturés par wireshark. L'ordre des paquets respecte le diagramme de séquence de la figure 2.1 et la synchronisation se répète chaque 2 secondes. Les paquets PTP sont formés correctement et les différents champs correspondent aux résultats attendus.

Ce test permet de valider le comportement de :
- La partie communication, c.-à-d. que les protocoles Ethernet, IP, UDP, PTP sont gérés correctement aussi bien à l'émission qu'à la réception.
- La partie application du dispositif maître. Les messages envoyés à l'esclave et les réponses aux messages reçus correspondent au diagramme de la figure 2.1 page 10.
- La partie application du dispositif esclave. Le comportement concorde avec celui décrit en figure 2.1 page 10.

4.3 Détermination de l'offset et de la dérive

À la fin de chaque cycle de synchronisation, l'esclave connaît les temps t_1 à t_4. Ils sont utilisés pour déterminer l'offset avec le maître et le temps de vol du message. Le temps de vol est déterminé selon l'équation 2.1 page 11. Le but de ce test est de vérifier que les calculs réalisés par l'esclave sont exacts.

4.3.1 Résultats

Les données listées dans le tableau 3.4 page 16 sont envoyées au PC par la liaison RS-232 représentée en figure 4.1 page 29. Les données de chaque cycle de synchronisation sont enregistrées dans un fichier du type de celui de l'annexe E.2.2 page 77. Les données utilisées pour la vérification sont :

Colonnes A et B	: offset calculé par la FPGA
Colonnes C à J	: temps t_1 à t_4
Colonne K et L	: temps de vol calculé par la FPGA
Colonne M	: temps de vol calculé par Excel à partir des données des colonnes C à J
Colonne N et O	: offset calculé par Excel à partir des données des colonnes C à J

Le formatage conditionnel proposé par Excel est utilisé pour colorer la cellule M en vert dans le cas où le temps de vol calculé par Excel concorde avec celui calculé par la FPGA. Le même principe est utilisé avec les cellules N et O pour vérifier le calcul de l'offset.

Le test est un succès. Il confirme le fonctionnement du calcul de l'offset et du temps de vol.

4.4 Synchronisation

Les différents tests précédents ont certifié le fonctionnement de l'intégralité du *design*. Il ne manque plus que la synchronisation.

La synchronisation n'est malheureusement pas opérationnelle. Il manque l'implémentation du régulateur. Néanmoins, les résultats provenant des simulations sont prometteurs (cf. section 3.9).

CHAPITRE 5

Conclusion

Ce projet a abouti à deux nœuds Ethernet capables de communiquer entre eux afin de se synchroniser. La combinaison entre l'horloge locale et l'unité de *timestamping* permet de déterminer précisément les temps d'émission et réception des différents messages. Les tests effectués valident le comportement de la partie communication. Plus concrètement, les dispositifs maître et esclave sont capables de s'échanger la totalité des informations nécessaires à une synchronisation.

Le module esclave est capable de calculer correctement la correction à effectuer pour une synchronisation avec le maître. Néanmoins, il n'est pas capable d'adapter la vitesse de l'horloge locale afin de corriger le décalage avec l'horloge maître. Pour remédier à cette situation, un régulateur doit être dimensionné et réalisé en VHDL. À l'heure actuelle, des simulations de ce régulateur ont été réalisées. Les résultats sont prometteurs, il semblerait que l'objectif de parvenir à une précision en dessous de la microseconde soit atteignable.

Plusieurs améliorations peuvent être ajoutées à la version actuelle du projet. Lorsque le maître envoie un message `Sync` *multicast*, l'esclave répond immédiatement. Cette réponse immédiate engendre des collisions dès qu'il y a plus d'un esclave sur le réseau. Pour en synchroniser plusieurs, il suffit d'ajouter à chaque esclave une logique qui attend un temps aléatoire avant de répondre à un message *multicast*. Cette méthode permet d'éviter les collisions systématiques et ainsi de permettre l'utilisation de messages *multicast* pour la synchronisation de plusieurs entités.

Le PHY Ethernet signale les collisions à la FPGA, mais celle-ci ne les traite pas. Une modification de la logique qui gère la communication Ethernet pour qu'elle puisse gérer les collisions serait un atout non négligeable, surtout si le nombre de nœuds à synchroniser augmente. En effet, l'utilisation de la bande passante du réseau et donc la probabilité de collision augmentent proportionnellement aux nombres de nœuds à synchroniser.

A l'heure actuelle, la partie du protocole PTP qui permet la synchronisation de deux nœuds dans un réseau Ethernet point à point est implémentée. Une implémentation plus complète du protocole permettrait d'aboutir à un système plus polyvalent et performant. Par exemple, les horloges pourraient se synchroniser à travers des switch Ethernet.

La dernière amélioration appréciable serait d'équiper l'horloge maître d'une antenne GPS. Avec les informations relatives au temps fournies par les satellites GPS, le maître pourrait accorder son calendrier au standard international.

Sion, le 9 juillet 2012
Johan Droz

CHAPITRE 6

Bibliographie

[1] IEEE Instrumentation and Measurement Society, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Revision of IEEE Std 1588-2002, July 2008

[2] Wikipedia, *Global Positioning System*, http://fr.wikipedia.org/wiki/Global_Positioning_System

[3] Wikipedia, *Synchronisation d'horloges*, http://fr.wikipedia.org/wiki/Synchronisation_d'horloges

[4] Wikipedia, *Epoch*, http://fr.wikipedia.org/wiki/Epoch

[5] Epoch Converter, *Epoch & Unix Timestamp Conversion Tools*, http://www.epochconverter.com/

[6] Wikipedia, *Temps universel coordonné*, http://fr.wikipedia.org/wiki/Temps_universel_coordonné

[7] Wikipedia, *Horodatage*, http://fr.wikipedia.org/wiki/Horodatage

[8] JTelec S.A., *Les oscillateurs à quartz*, http://www.jtelec.fr/documentation_oscillateur.php

ANNEXE A

Kit FPGA

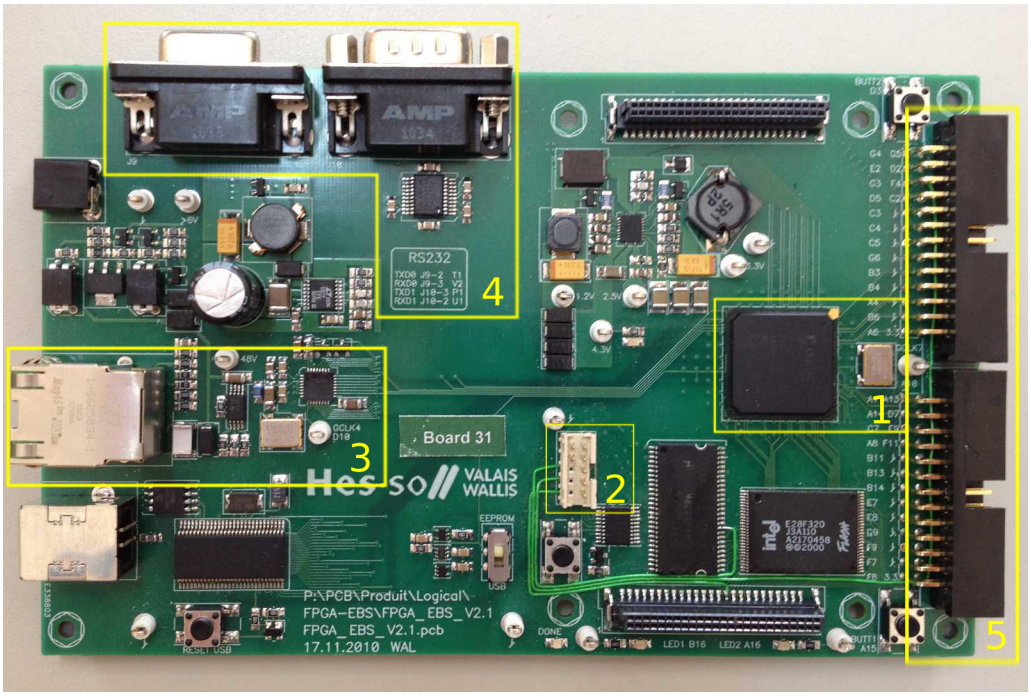


FIGURE A.1 – Kit FPGA v2

N°	Ressource
1	FPGA, Oscillateur 66 [MHz]
2	JTAG
3	PHY Ethernet
4	Module RS-232
5	Connecteurs pour les cartes d'extensions

TABLE A.1 – Utilisation du kit FPGA

ANNEXE B

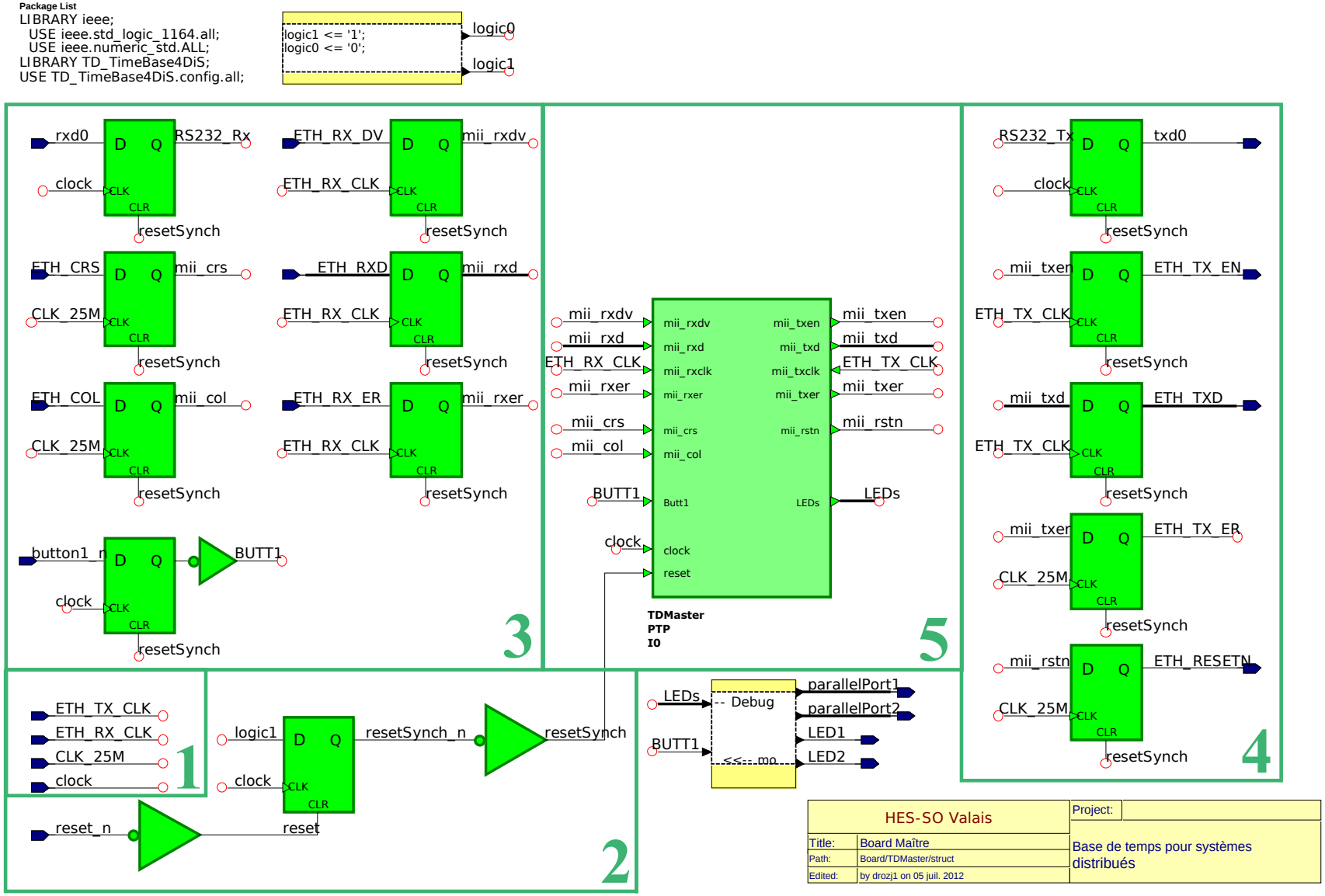
FPGA - Maître

B.1 Vue Graphique

B.1.1 Board

La description ci-dessous fait référence à l'annexe Board page 35.

1. Ces quatre signaux sont de haut en bas, l'horloge spécifique à la transmission Ethernet, l'horloge spécifique à la réception Ethernet, l'horloge 25 [MHz] du PHY Ethernet et l'horloge 66 [MHz] de la FPGA.
2. Synchronisation du signal reset sur l'horloge FPGA.
3. Les bascules D synchronisent les signaux d'entrées sur leurs horloges respectives.
4. Les bascules D synchronisent les signaux de sorties sur leurs horloges respectives.
5. Le bloc contient l'intégralité du projet VHDL, donné en annexe B.1.2 en pages 37 et 38.

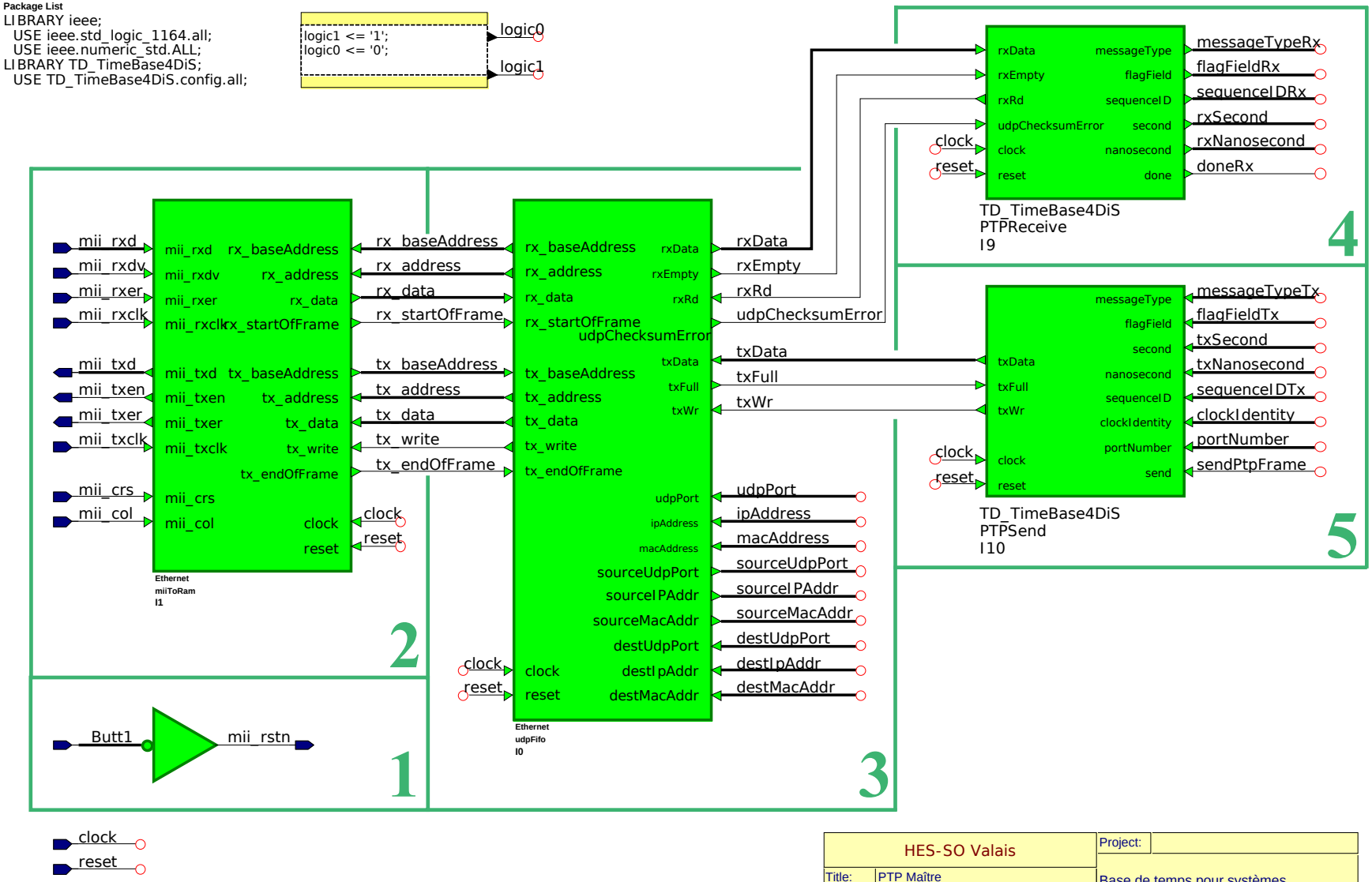

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
USE TD_TimeBase4DiS.config.all;
logic1 <= '1';
logic0 <= '0';
logic0
logic1
rxd0
RS232_Rx
clock
resetSynch
ETH_RX_DV
mii_rxdv
ETH_RX_CLK
ETH_CRS
mii_crs
CLK_25M
ETH_RXD
mii_rxd
ETH_COL
mii_col
ETH_RX_ER
mii_rxer
button1_n
BUTT1
3
TDMaster
PTP
IO
mii_txen
mii_txd
mii_txclk
mii_txer
mii_rstn
LEDs
Butt1
reset
5
RS232_Tx
txd0
ETH_TX_EN
ETH_TX_CLK
ETH_TXD
ETH_TX_ER
ETH_RESETN
4
ETH_TX_CLK
ETH_RX_CLK
CLK_25M
1
logic1
resetSynch_n
reset_n
reset
2
Debug
parallelPort1
parallelPort2
LED1
LED2
HES-SO Valais
Project:
Title: Board Maître
Path: Board/TDMaster/struct
Edited: by drozj1 on 05 juil. 2012
Base de temps pour systèmes distribués

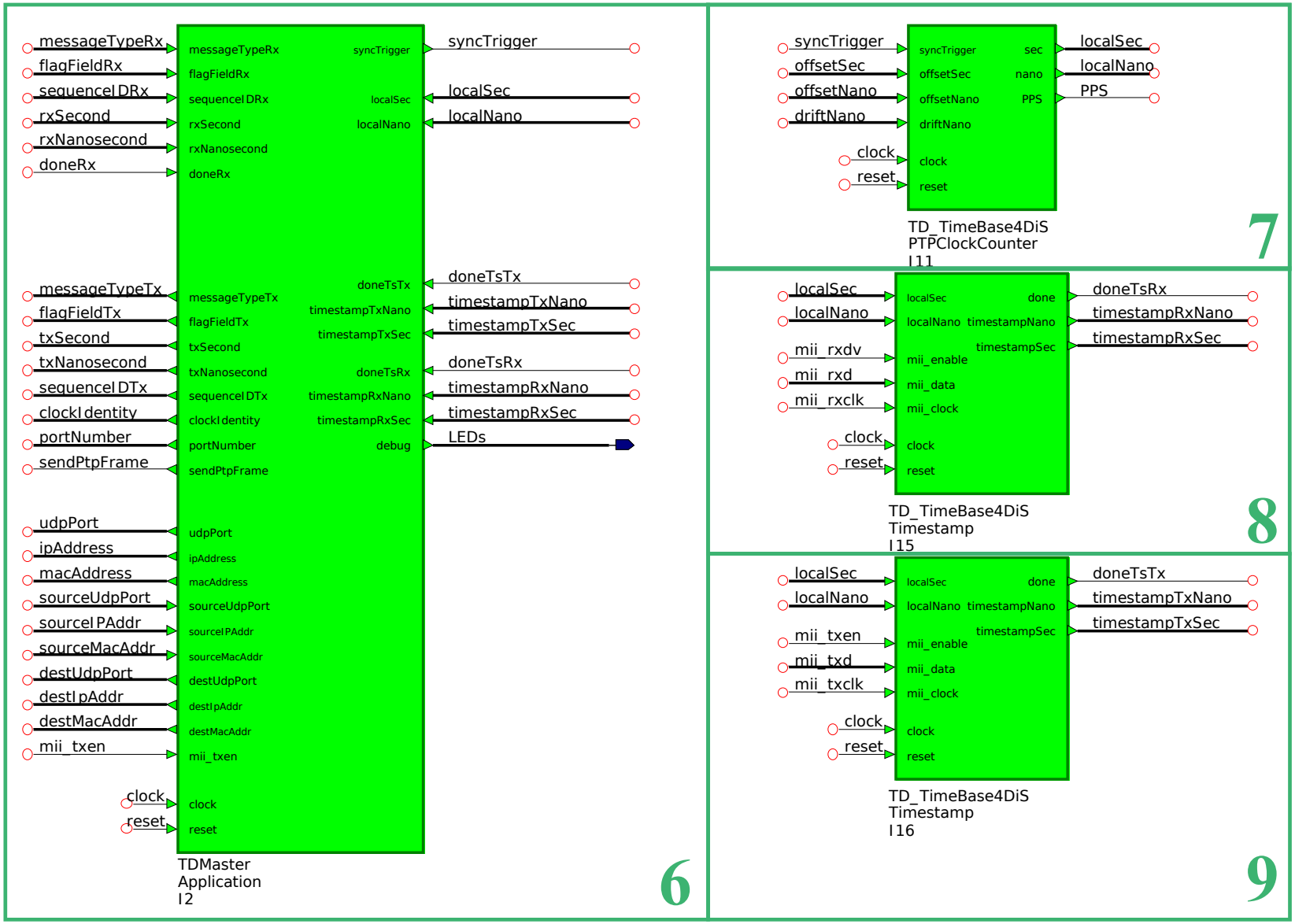
B.1.2 PTP

L'annexe pages 37 et 38 est contenue dans le bloc 5 de l'annexe page 35. Elle est décrite ci-dessous.

1. Reset du PHY Ethernet.
2. Copie les trames Ethernet reçues en mémoire RAM et envoie les trames Ethernet copiées en mémoire RAM.
3. UDP FIFO : gère l'encapsulation (Ethernet, IP, UDP) lors de l'émission et la réception de trames Ethernet.
4. Décode les messages PTP reçus. Les informations utiles sont envoyées à la partie application.
5. Créé le message PTP selon les paramètres donnés par l'application.
6. Application maître.
7. Horloge locale.
8. *Timestamp* de réception.
9. *Timestamp* d'émission.

Les blocs 2 et 3 ont été fournis par l'école au début du projet.
Le code VHDL du bloc n° 4 est disponible en annexe D.2 page 62.
Le code VHDL du bloc n° 5 est disponible en annexe D.3 page 65.
Le bloc n° 6 contient l'annexe B.1.3 page 43.
Le code VHDL du bloc n° 7 est disponible en annexe D.4 page 69.
Les blocs n° 8 et n° 9 sont identiques et contiennent la vue en page 40.


Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
USE TD_TimeBase4DiS.config.all;
logic1 <= '1';
logic0 <= '0';
logic0
logic1
mii_rxd
mii_rxdv
mii_rxer
mii_rxclk
mii_txd
mii_txen
mii_txer
mii_txclk
mii_crs
mii_col
rx_baseAddress
rx_address
rx_data
rx_startOfFrame
tx_baseAddress
tx_address
tx_data
tx_write
tx_endOfFrame
clock
reset
Ethernet
miiToRam
I1
2
Butt1
mii_rstn
1
rxData
rxEmpty
rxRd
udpChecksumError
txData
txFull
txWr
udpPort
ipAddress
macAddress
sourceUdpPort
sourceIPAddr
sourceMacAddr
destUdpPort
destIpAddr
destMacAddr
Ethernet
udpFifo
I0
3
messageTypeRx
flagFieldRx
sequenceIDRx
rxSecond
rxNanosecond
doneRx
TD_TimeBase4DiS
PTPReceive
I9
4
messageTypeTx
flagFieldTx
txSecond
txNanosecond
sequenceIDTx
clockIdentity
portNumber
sendPtpFrame
TD_TimeBase4DiS
PTPSend
I10
5
HES-SO Valais
Project:
Title: PTP Maître
Path: TDMaster/PTP/struct
Edited: by drozj1 on 05 juil. 2012
Base de temps pour systèmes distribués


messageTypeRx
flagFieldRx
sequenceIDRx
rxSecond
rxNanosecond
doneRx
messageTypeTx
flagFieldTx
txSecond
txNanosecond
sequenceIDTx
clockIdentity
portNumber
sendPtpFrame
udpPort
ipAddress
macAddress
sourceUdpPort
sourceIPAddr
sourceMacAddr
destUdpPort
destIpAddr
destMacAddr
mii_txen
clock
reset
syncTrigger
localSec
localNano
doneTsTx
timestampTxNano
timestampTxSec
doneTsRx
timestampRxNano
timestampRxSec
debug
LEDs
TDMaster
Application
I2
6
offsetSec
offsetNano
driftNano
sec
nano
PPS
TD_TimeBase4DiS
PTPClockCounter
I11
7
mii_rxdv
mii_rxd
mii_rxclk
mii_enable
mii_data
mii_clock
done
timestampNano
timestampSec
TD_TimeBase4DiS
Timestamp
I15
8
mii_txen
mii_txd
mii_txclk
TD_TimeBase4DiS
Timestamp
I16
9

Timestamp

Les signaux `mii_enable`, `mii_clk` et `mii_data` proviennent directement du PHY Ethernet. C'est pourquoi ils sont synchronisés sur l'horloge de la FPGA avec les bascules D de la sélection n° 1. Le bloc `TimestampFSM` (n° 2) contient la machine d'états de la page 41.

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

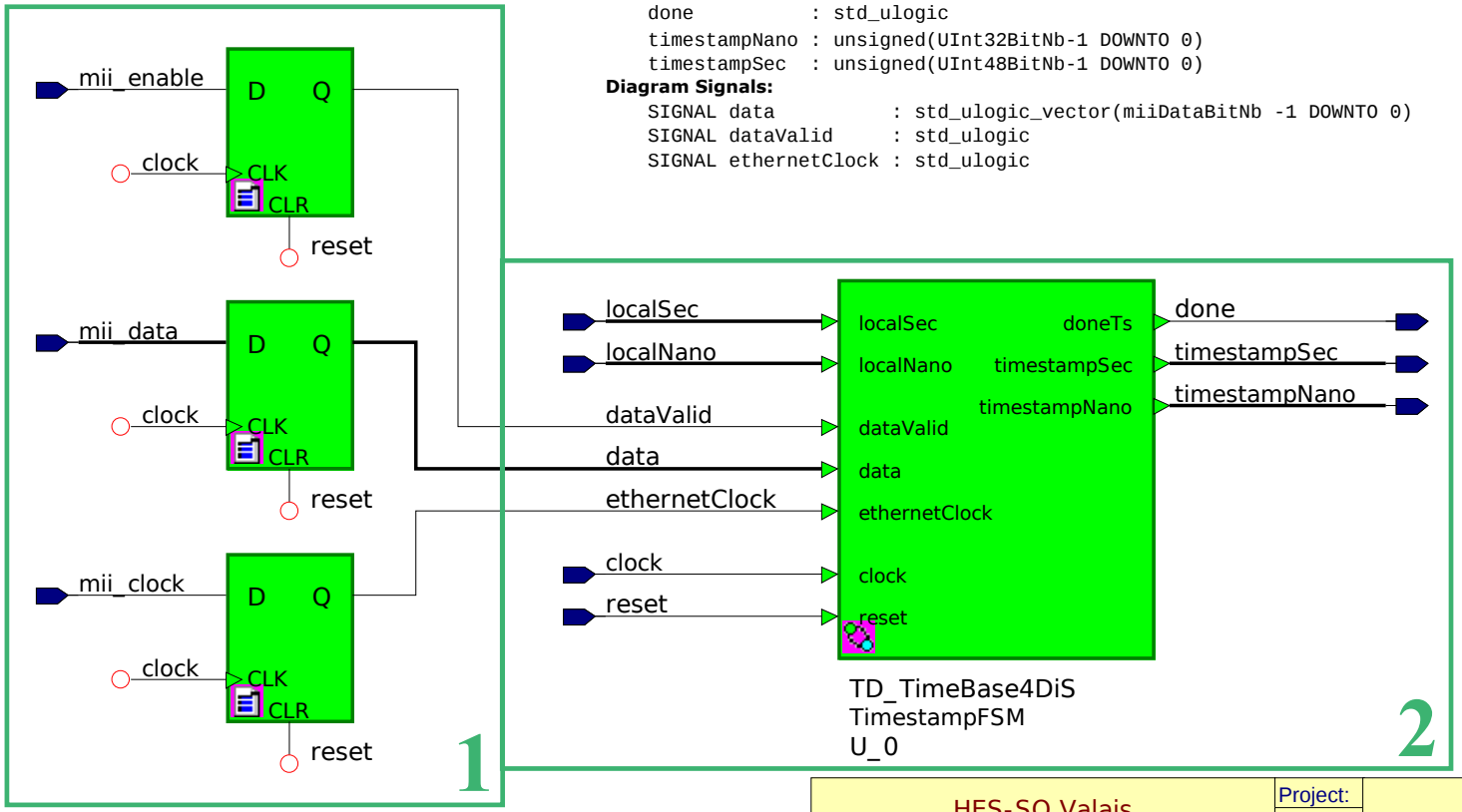
Declarations

Ports:

```
clock         : std_ulogic
localNano     : unsigned(UInt32BitNb-1 DOWNTO 0)
localSec      : unsigned(UInt48BitNb-1 DOWNTO 0)
mii_clock     : std_ulogic
mii_data      : std_ulogic_vector(miiDataBitNb -1 DOWNTO 0)
mii_enable    : std_ulogic
reset         : std_ulogic
done          : std_ulogic
timestampNano : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampSec  : unsigned(UInt48BitNb-1 DOWNTO 0)
```

Diagram Signals:

```
SIGNAL data          : std_ulogic_vector(miiDataBitNb -1 DOWNTO 0)
SIGNAL dataValid     : std_ulogic
SIGNAL ethernetClock : std_ulogic
```



HES-SO Valais		Project:
Title:	Timestamp Unit	Base de temps pour systèmes distribués
Path:	TD_TimeBase4DiS/Timestamp/struct	
Edited:	by drozj1 on 05 juil. 2012	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements
Architecture Declarations

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
doneTs	OUT	'0'	'0'	CLKD
timestampNano	OUT		(others => '0')	CLKD
timestampSec	OUT		(others => '0')	CLKD
oldEthClock	LOCAL	ethernetClock	'0'	CLKD

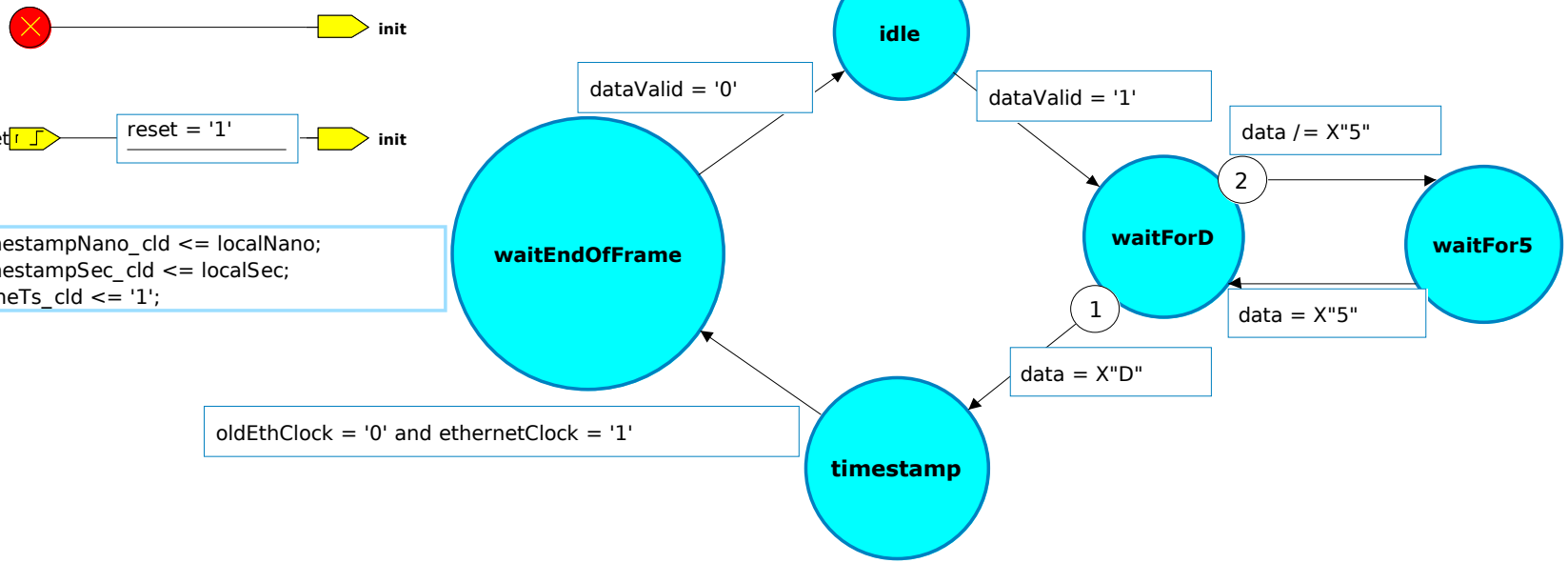
State Register Statements

Process Declarations
Clocked Process:
Output Process:

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

clock — rising_edge(clock)



reset — reset = '1' — init

init

HES-SO Valais		Project:
Title:	Timestamp FSM	Base de temps pour systèmes distribués
Path:	TD_TimeBase4DiS/TimestampFSM/fsm	
Edited:	by drozj1 on 05 juil. 2012	

B.1.3 Application PTP

L'annexe page 43 est contenue dans le bloc 6 de l'annexe page 38. Elle contient :

1. Contrôle le départ d'un cycle de synchronisation.
2. Contrôle les *timeout* utilisés par la machine d'états de l'application.
3. Gère l'état du réseau (occupé, libre).
4. Machine d'états du maître.

Le bloc n° 1 contient la machine d'états de la page 44.
Le bloc n° 2 contient la machine d'états de la page 45.
Le bloc n° 3 contient la machine d'états de la page 46.
Le bloc n° 4 contient la machine d'états de la page 47.

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

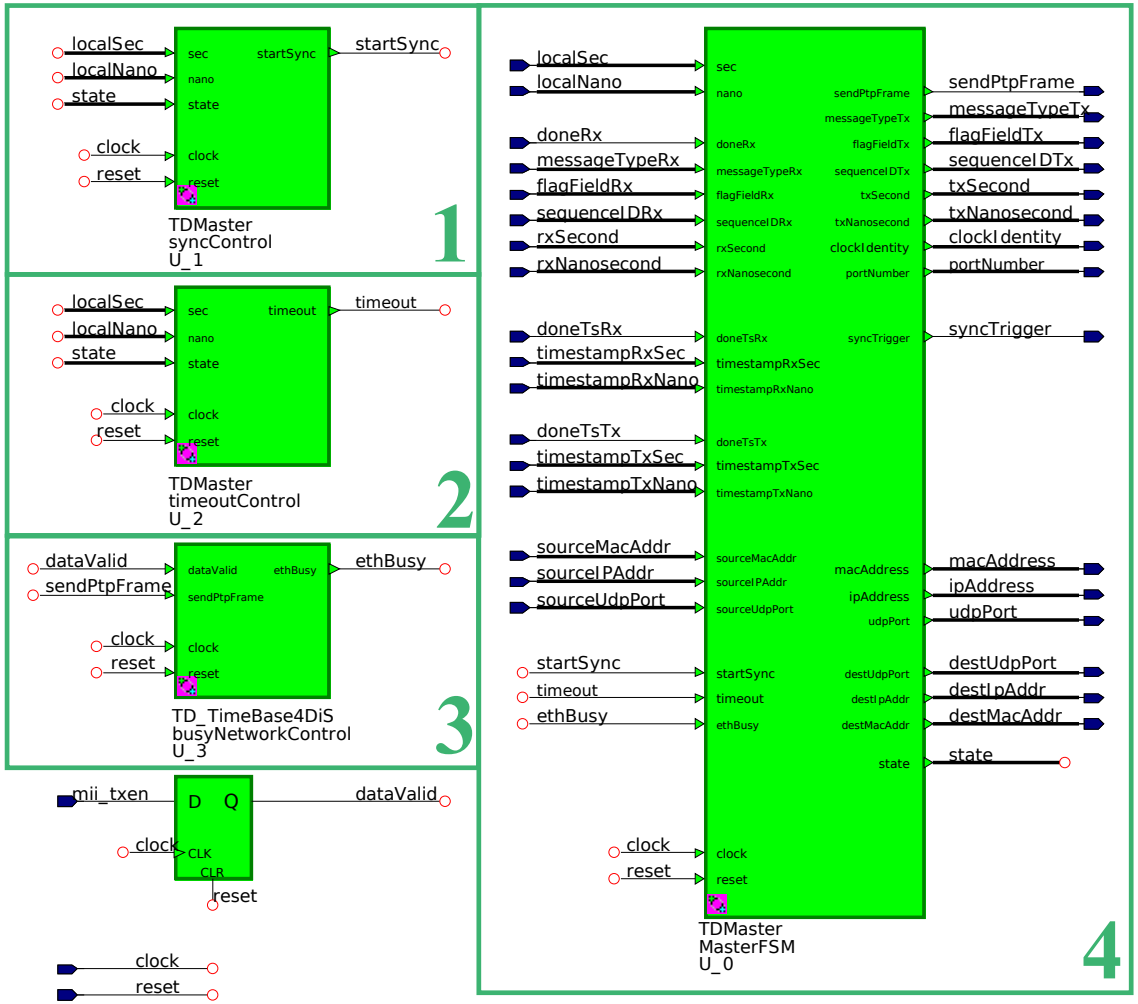
Declarations

Ports:

```
clock           : std_ulogic
doneRx          : std_ulogic
doneTsRx        : std_ulogic
doneTsTx        : std_ulogic
flagFieldRx     : std_ulogic_vector(15 DOWNTO 0)
localNano       : unsigned(UInt32BitNb-1 DOWNTO 0)
localSec        : unsigned(UInt48BitNb-1 DOWNTO 0)
messageTypeRx   : std_ulogic_vector(3 DOWNTO 0)
mii_txen        : std_uLogic
reset           : std_ulogic
rxNanosecond    : unsigned(UInt32BitNb-1 DOWNTO 0)
rxSecond        : unsigned(UInt48BitNb-1 DOWNTO 0)
sequenceIDRx    : std_ulogic_vector(15 DOWNTO 0)
sourceIPAddr    : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
sourceMacAddr   : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
sourceUdpPort   : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
timestampRxNano : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampRxSec  : unsigned(UInt48BitNb-1 DOWNTO 0)
timestampTxNano : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampTxSec  : unsigned(UInt48BitNb-1 DOWNTO 0)
clockIdentity   : std_ulogic_vector(79 DOWNTO 0)
debug           : std_ulogic_vector(15 DOWNTO 0)
destIpAddr      : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
destMacAddr     : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
destUdpPort     : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
flagFieldTx     : std_ulogic_vector(15 DOWNTO 0)
ipAddress       : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
macAddress      : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
messageTypeTx   : std_ulogic_vector(3 DOWNTO 0)
portNumber      : std_ulogic_vector(15 DOWNTO 0)
sendPtpFrame    : std_ulogic
sequenceIDTx    : std_ulogic_vector(15 DOWNTO 0)
syncTrigger     : std_ulogic
txNanosecond    : unsigned(UInt32BitNb-1 DOWNTO 0)
txSecond        : unsigned(UInt48BitNb-1 DOWNTO 0)
udpPort         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```

Diagram Signals:

```
SIGNAL dataValid       : std_uLogic
SIGNAL ethBusy         : std_ulogic
SIGNAL startSync       : std_ulogic
SIGNAL state           : std_ulogic_vector(3 DOWNTO 0)
SIGNAL timeout         : std_ulogic
```



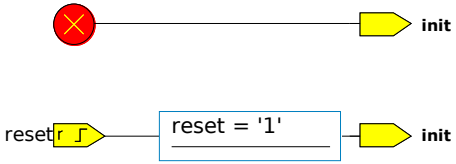
HES-SO Valais		Project:	
Title:	Application Maître		Base de temps pour systèmes distribués
Path:	TDMaster/Application/struct		
Edited:	by drozj1 on 05 juil. 2012		

Global Actions **Concurrent Statements** **Architecture Declarations**
Pre Actions:
Post Actions:

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

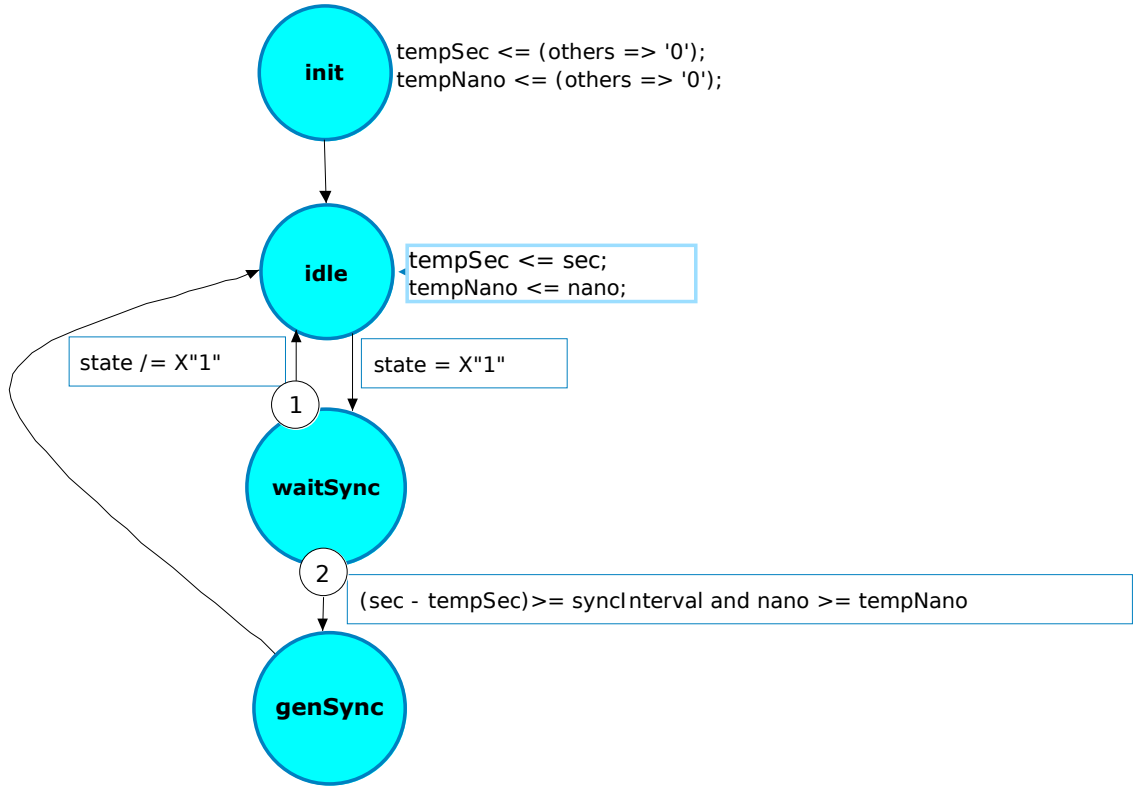
clock ⊂ ⎍ rising_edge(clock)



Signal Status

SIGNAL	MODE	State Register Statements DEFAULT	RESET	SCHEME
startSync	OUT	'0'		CLKD
tempSec	LOCAL			CLKD
tempNano	LOCAL			CLKD
syncInterval	LOCAL	X"02"		CLKD

Process Declarations
Clocked Process:
Output Process:



HES-SO Valais		Project:	
Title:	Synchronisation FSM	Base de temps pour systèmes distribués	
Path:	TDMaster/syncControl/fsm		
Edited:	by drozj1 on 05 juil. 2012		

Global Actions **Concurrent Statements** **Architecture Declarations**
Pre Actions:
Post Actions:

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
timeout	OUT	'0'	'0'	CLKD
tempS	LOCAL		(others => '0')	CLKD
tempN	LOCAL		(others => '0')	CLKD
timeoutValue	LOCAL	X"02"	X"02"	CLKD

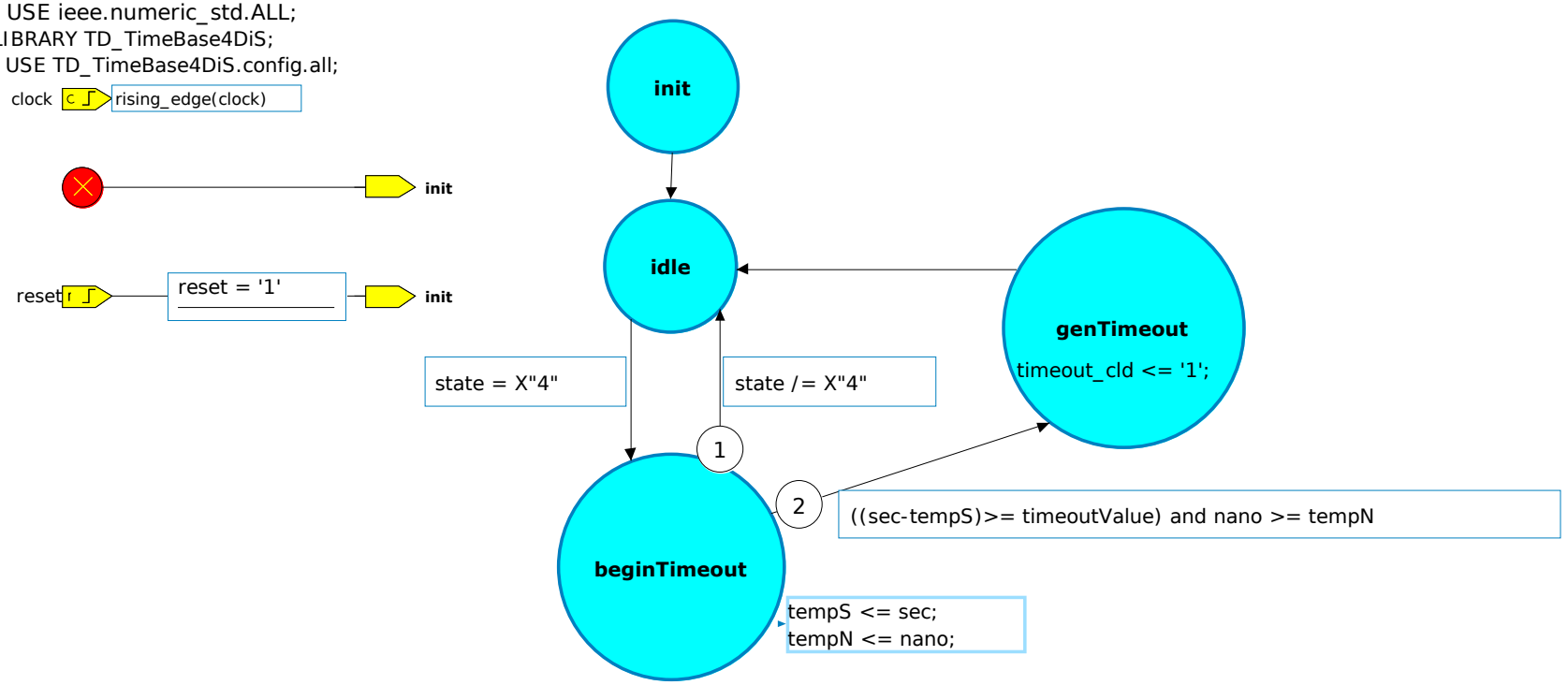
State Register Statements

Process Declarations
Clocked Process:
Output Process:

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

clock C rising_edge(clock)



HES-SO Valais		Project:
Title:	Timeout FSM	Base de temps pour systèmes distribués
Path:	TDMaster/timeoutControl/fsm	
Edited:	by drozj1 on 05 juil. 2012	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

```
constant tToWait:positive := 634;
```

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
ethBusy	OUT	'1'		CLKD
counter	LOCAL		(others => '0')	CLKD

State Register Statements

Process Declarations
Clocked Process:
Output Process:

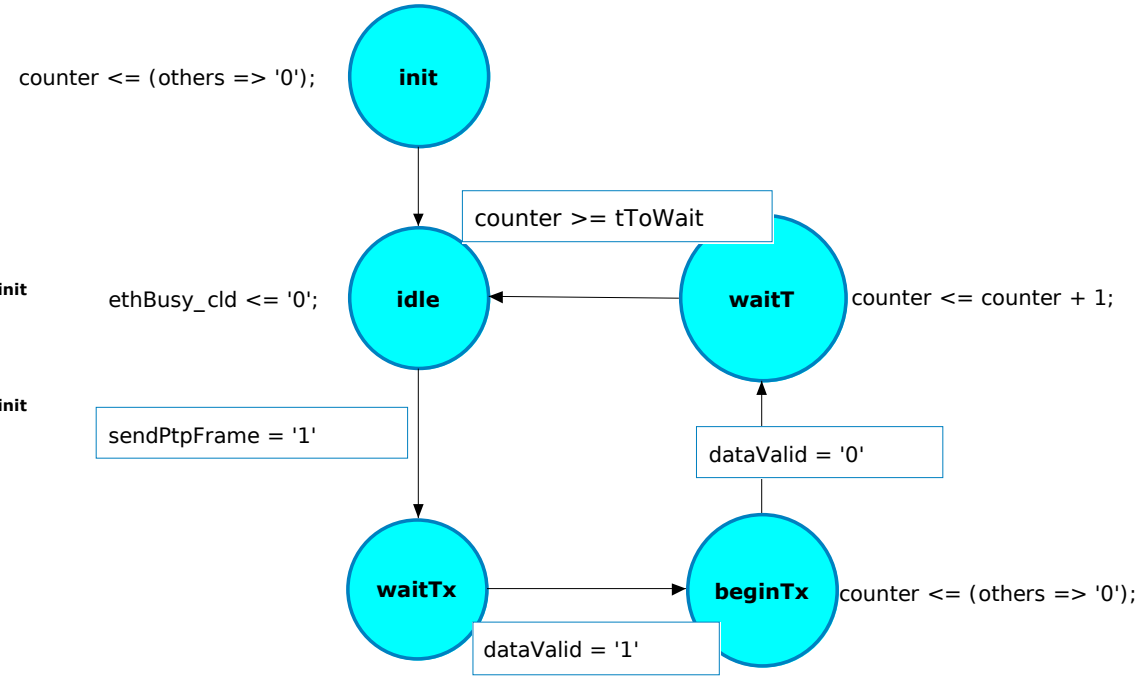
Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

clock rising_edge(clock)

init

reset reset = '1' init



HES-SO Valais		Project:	
Title:	Network Control FSM	Base de temps pour systèmes distribués	
Path:	TD_TimeBase4DiS/busyNetworkControl/fsm		
Edited:	by drozj1 on 05 juil. 2012		

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

```
sendPtpFrame <= sendPTP;
destIpAddr <= X"A9FEF05B";
destMacAddr <= X"E4AFA1391001";
destUdpPort <= X"013F";
ipAddress <= X"A9FEF05C";
macAddress <= X"E4AFA1391000";
udpPort <= X"013F";
```

Architecture Declarations

Package List

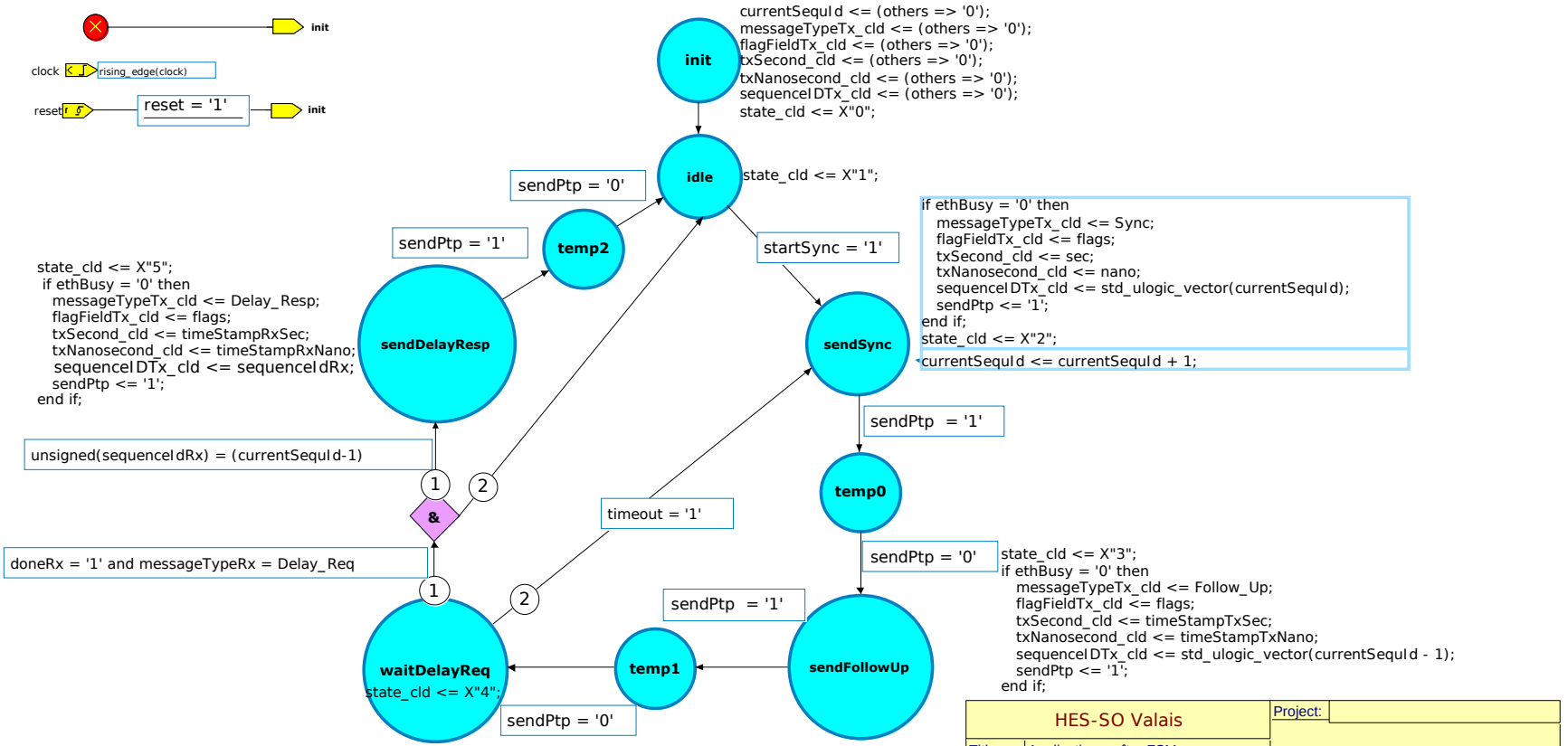
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
clockIdentity	OUT	(others => '0')		COMB
state	OUT	(others => '0')	(others => '0')	CLKD
destIpAddr	OUT			COMB
destMacAddr	OUT			COMB
destUdpPort	OUT			COMB
flagFieldTx	OUT		(others => '0')	CLKD
ipAddress	OUT			COMB
macAddress	OUT			COMB
messageTypeTx	OUT		(others => '0')	CLKD
portNumber	OUT	(others => '0')		COMB
sequenceIDTx	OUT		(others => '0')	CLKD
syncTrigger	OUT	'0'	'0'	CLKD
txNanosecond	OUT		(others => '0')	CLKD
txSecond	OUT		(others => '0')	CLKD
udpPort	OUT			COMB
sendPtp	LOCAL	'0'	'0'	CLKD
currentSequId	LOCAL		(others => '0')	CLKD
flags	LOCAL	X"0200"		COMB
sendPtpFrame	OUT			COMB

State Register Statements

Process Declarations
Clocked Process:
Output Process:



HES-SO Valais		Project:	
Title:	Application maître FSM	Base de temps pour systèmes distribués	
Path:	TDMaster/MasterFSM/fsm		
Edited:	by drozj1 on 05 juil. 2012		

ANNEXE C

FPGA - Esclave

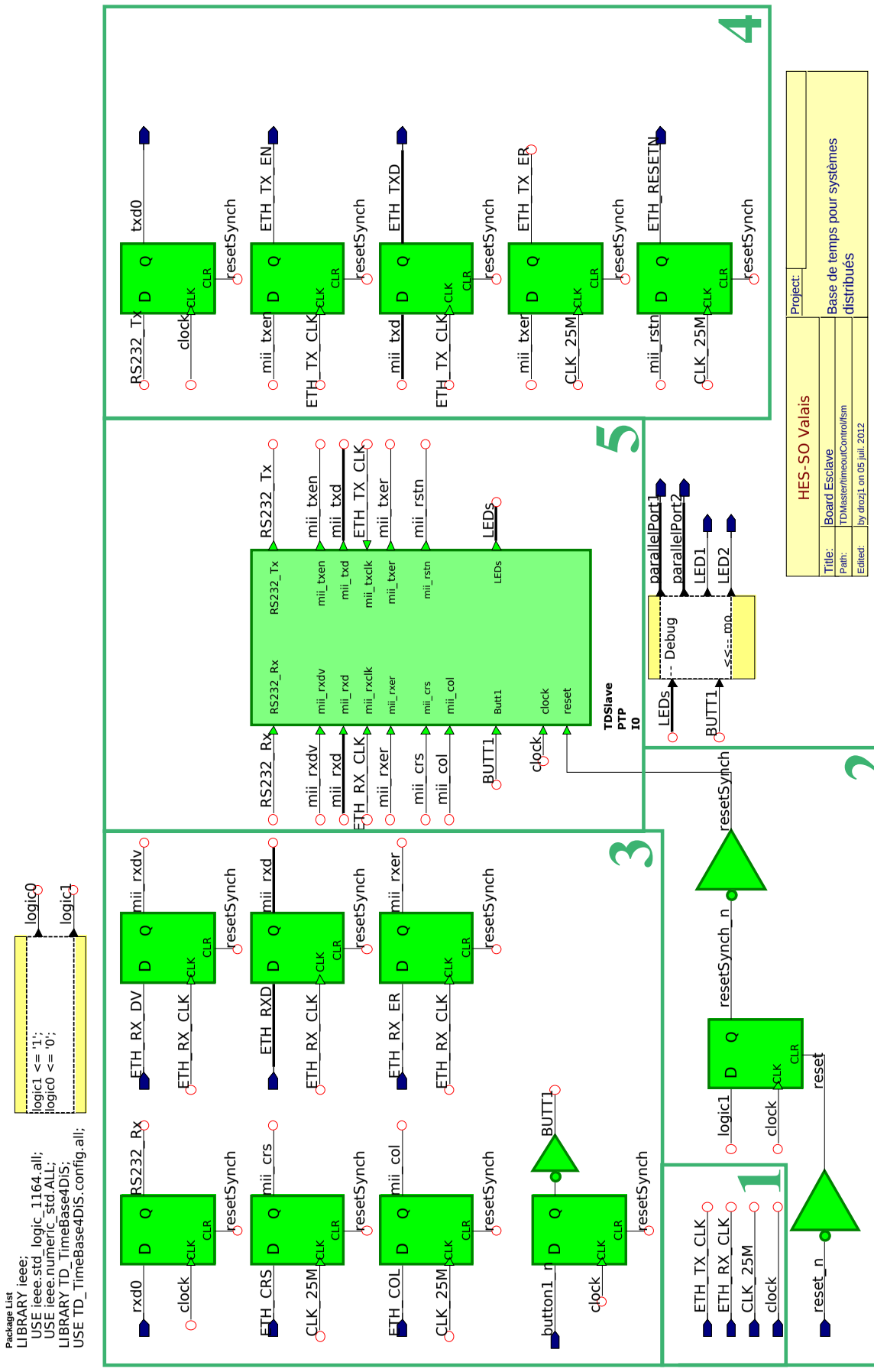
C.1 Vue graphique

C.1.1 Board

La description ci-dessous fait référence à l'annexe Board page 49.

1. Ces quatre signaux sont de haut en bas, l'horloge spécifique à la transmission Ethernet, l'horloge spécifique à la réception Ethernet, l'horloge 25 [MHz] du PHY Ethernet et l'horloge 66 [MHz] de la FPGA.
2. Synchronisation du signal reset sur l'horloge FPGA.
3. Les bascules D synchronisent les signaux d'entrées sur leurs horloges respectives.
4. Les bascules D synchronisent les signaux de sorties sur leurs horloges respectives.
5. Le bloc contient l'intégralité du projet VHDL, donné en annexe C.1.2 en pages 51 à 53.

La différence entre la vue graphique Board du maître et de l'esclave est que le bloc n° 5 de l'esclave possède les signaux RS232_Tx et RS232_Rx qui lui permettent de communiquer avec un PC.

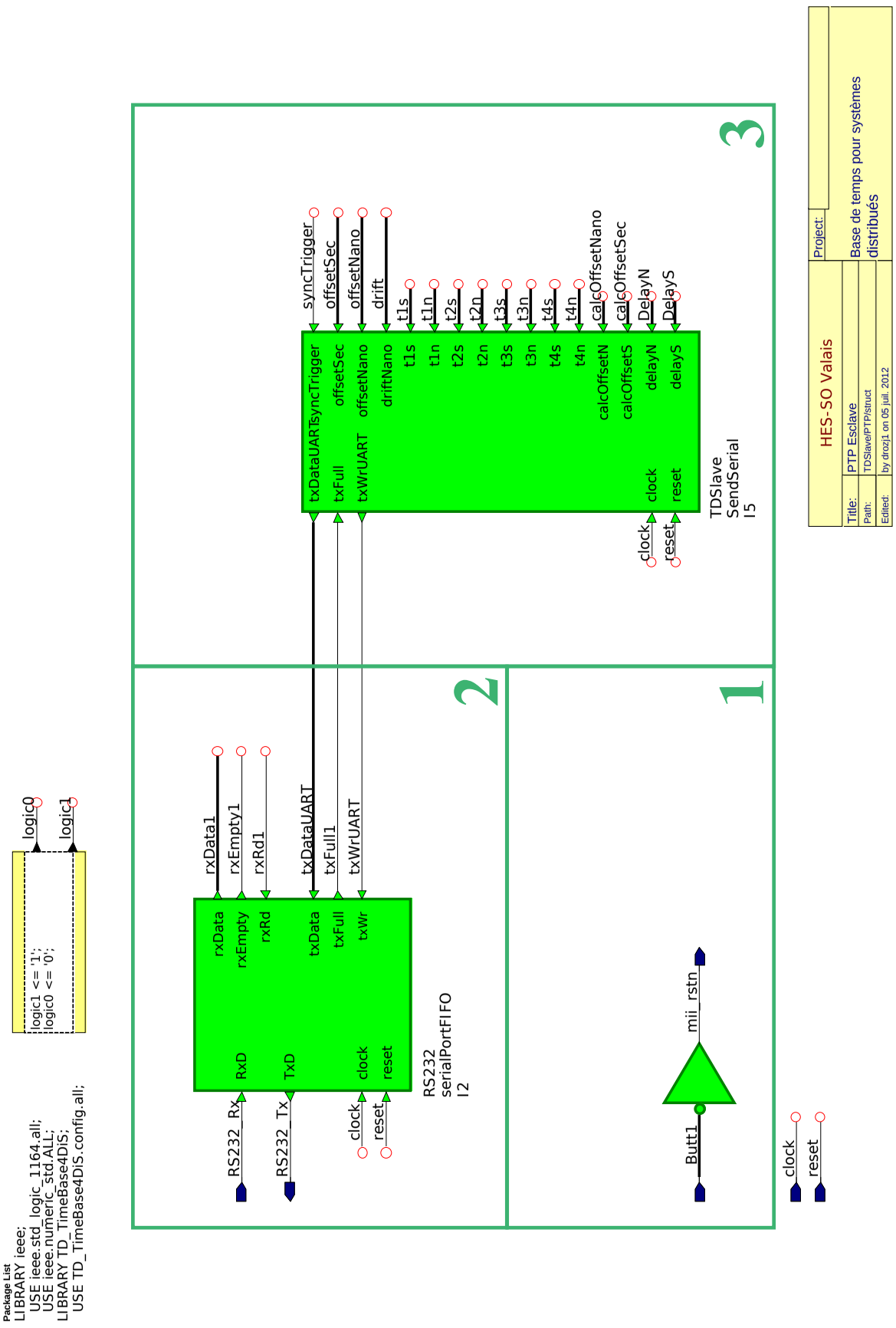

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
USE TD_TimeBase4DiS.config.all;
logic1 <= '1';
logic0 <= '0';
logic0
logic1
1
2
3
4
5
txd0
RS232_Tx
clock
resetSynch
ETH_TX_EN
mii_txer
ETH_TX_CLK
ETH_TXD
mii_txd
ETH_TX_ER
CLK_25M
ETH_RESETN
mii_rstn
RS232_Rx
mii_rxdv
mii_rxd
ETH_RX_CLK
mii_rxer
mii_crs
mii_col
BUTT1
LEDs
mii_txen
mii_txclk
TDSlave
PTP
IO
parallelPort1
parallelPort2
LED1
LED2
Debug
resetSynch_n
reset
button1_n
reset_n
ETH_RX_DV
ETH_RXD
ETH_RX_ER
ETH_CRS
ETH_COL
HES-SO Valais
Project:
Title:
Board Esclave
Base de temps pour systèmes distribués
Path:
TDMaster/timeoutControl/fsm
Edited:
by droz1 on 05 juil. 2012

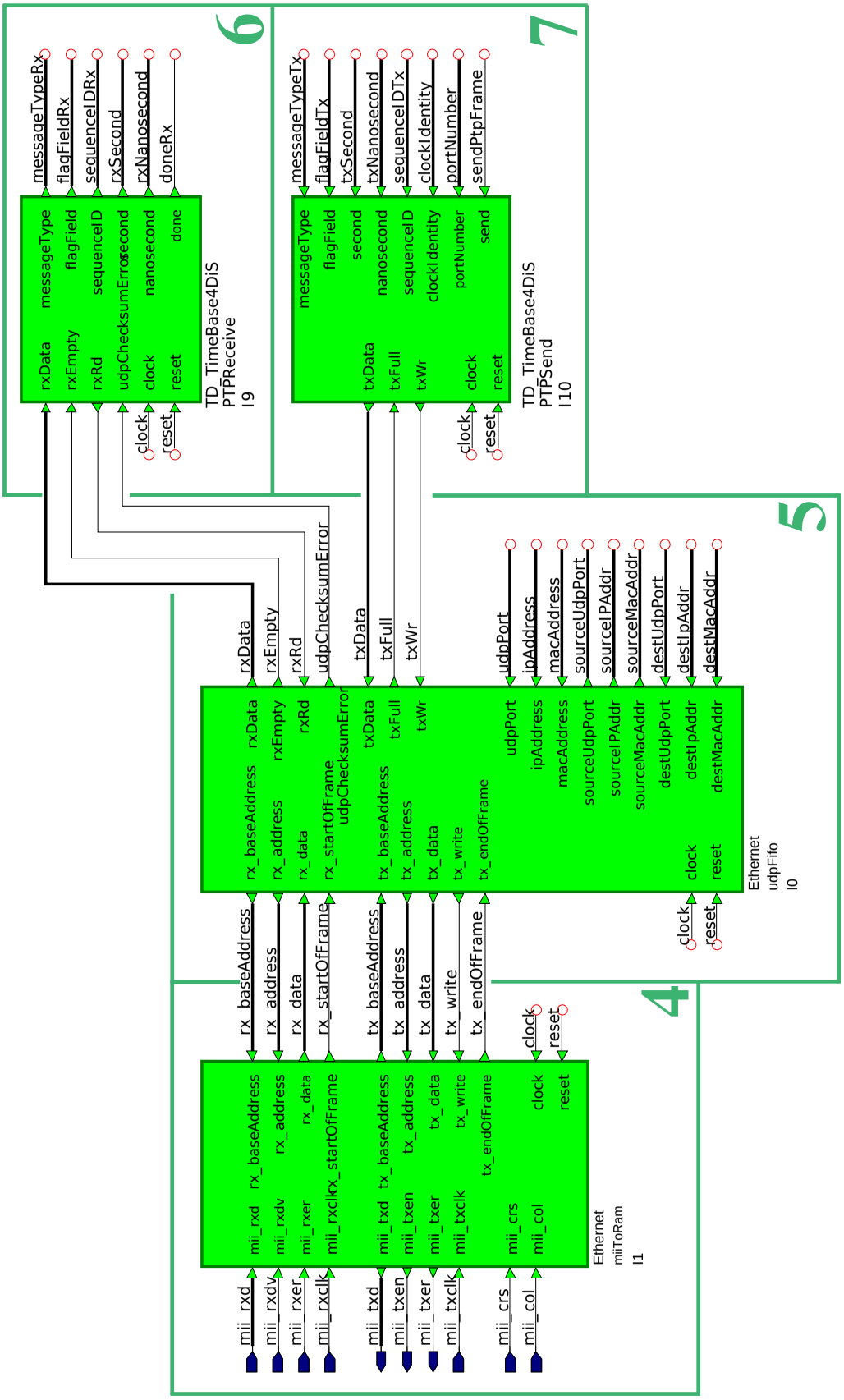
C.1.2 PTP

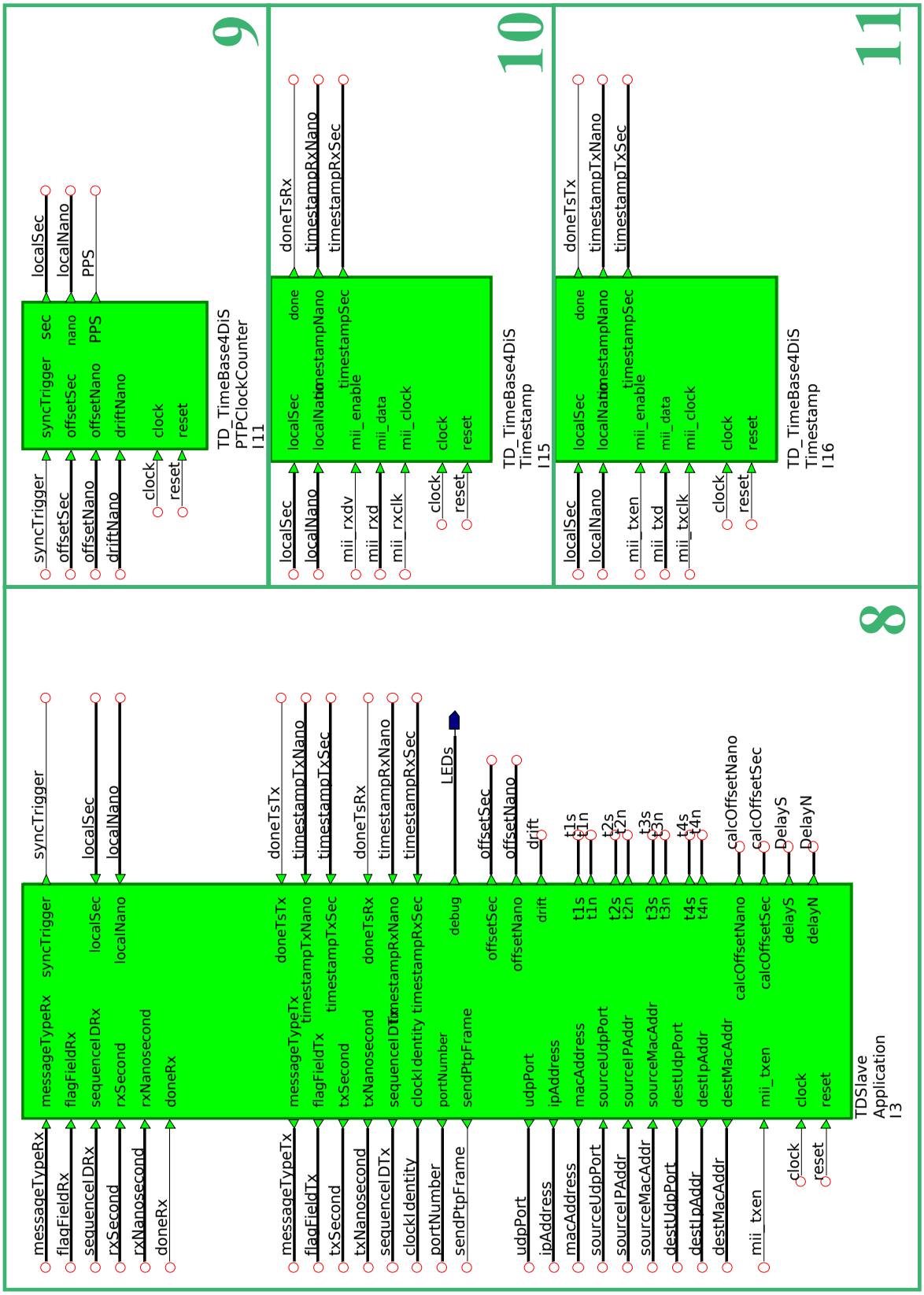
L'annexe pages 51 à 53 est contenue dans le bloc 5 de l'annexe page 49. Elle est décrite ci-dessous.

1. Reset du PHY Ethernet.
2. Port série.
3. Contrôle des données envoyées sur le port série.
4. Copie les trames Ethernet reçues en mémoire RAM et envoie les trames Ethernet copiées en mémoire RAM.
5. UDP FIFO : gère l'encapsulation (Ethernet, IP, UDP) lors de l'émission et la réception de trames Ethernet.
6. Décode les messages PTP reçus. Les informations utiles sont envoyées à la partie application.
7. Créé le message PTP selon les paramètres donnés par l'application.
8. Application esclave.
9. Horloge locale.
10. *Timestamp* de réception.
11. *Timestamp* d'émission.

Les blocs 2, 4 et 5 ont été fournis par l'école au début du projet.
Le code VHDL du bloc n° 2 est disponible en annexe D.5 page 72.
Le code VHDL du bloc n° 6 est disponible en annexe D.2 page 62.
Le code VHDL du bloc n° 7 est disponible en annexe D.3 page 65.
Le bloc n° 8 contient l'annexe C.1.3 page 55.
Le code VHDL du bloc n° 9 est disponible en annexe D.4 page 69.
Les blocs n° 10 et n° 11 sont identiques et contiennent la vue en page 40.


Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
USE TD_TimeBase4DiS.config.all;
logic0
logic1
logic1 <= '1';
logic0 <= '0';
HES-SO Valais
Project:
Title: PTP Esclave
Path: TDSlave/PTP/struct
Edited: by droz1 on 05 juil. 2012
Base de temps pour systèmes distribués
1
2
3
TDSlave
SendSerial
I5
RS232
serialPortFIFO
I2
syncTrigger
offsetSec
offsetNano
drift
t1s
t1n
t2s
t2n
t3s
t3n
t4s
t4n
calcOffsetNano
calcOffsetSec
DelayN
DelayS
txDataUART
txFull
txWrUART
driftNano
calcOffsetN
calcOffsetS
delayN
delayS
clock
reset
rxData1
rxEmpty1
rxRd1
txDataUART
txFull1
txWrUART
rxData
rxEmpty
rxRd
txData
txFull
txWr
RxD
TxD
RS232_Rx
RS232_Tx
mii_rstn
Butt1


6
messageTypeRx
flagFieldRx
sequenceIDRx
rxSecond
rxNanosecond
doneRx
rxData
rxEmpty
rxRd
udpChecksumError
clock
reset
messageType
flagField
sequenceID
rxSecond
rxNanosecond
done
clock
reset
TD_TimeBase4DiS
PTPReceive
I9
7
messageTypeTx
flagFieldTx
txSecond
txNanosecond
sequenceIDTx
clockIdentity
portNumber
sendPtpFrame
txData
txFull
txWr
clock
reset
messageType
flagField
second
nanosecond
sequenceID
clockIdentity
portNumber
send
clock
reset
TD_TimeBase4DiS
PTPSend
I10
5
rxData
rxEmpty
rxRd
udpChecksumError
txData
txFull
txWr
udpPort
ipAddress
macAddress
sourceUdpPort
sourceIPAddr
sourceMacAddr
destUdpPort
destIpAddr
destMacAddr
rx_baseAddress
rx_address
rx_data
rx_startOfFrame
udpChecksumError
tx_baseAddress
tx_address
tx_data
tx_write
tx_endOfFrame
udpPort
ipAddress
macAddress
sourceUdpPort
sourceIPAddr
sourceMacAddr
destUdpPort
destIpAddr
destMacAddr
clock
reset
Ethernet
udpFifo
I0
4
rx_baseAddress
rx_address
rx_data
rx_startOfFrame
tx_baseAddress
tx_address
tx_data
tx_write
tx_endOfFrame
clock
reset
mii_rxd
mii_rxdv
mii_rxer
mii_rxclk
rx_baseAddress
rx_address
rx_data
rx_startOfFrame
mii_txd
mii_txen
mii_txer
mii_txclk
tx_baseAddress
tx_address
tx_data
tx_write
tx_endOfFrame
mii_crs
mii_col
clock
reset
Ethernet
miiToRam
I1
mii_rxd
mii_rxdv
mii_rxer
mii_rxclk
mii_txd
mii_txen
mii_txer
mii_txclk
mii_crs
mii_col


9
syncTrigger
offsetSec
offsetNano
driftNano
clock
reset
localSec
localNano
PPS
TD_TimeBase4DiS
PTPClockCounter
I11
10
localSec
localNano
mii_rxdv
mii_rxd
mii_rxclk
clock
reset
doneTsRx
timestampRxNano
timestampRxSec
TD_TimeBase4DiS
Timestamp
I15
11
localSec
localNano
mii_txen
mii_txd
mii_txclk
clock
reset
doneTsTx
timestampTxNano
timestampTxSec
TD_TimeBase4DiS
Timestamp
I16
8
messageTypeRx
flagFieldRx
sequenceIDRx
rxSecond
rxNanosecond
doneRx
syncTrigger
localSec
localNano
messageTypeTx
flagFieldTx
txSecond
txNanosecond
sequenceIDTx
clockIdentity
portNumber
sendPtpFrame
doneTsTx
timestampTxNano
timestampTxSec
doneTsRx
timestampRxNano
timestampRxSec
LEDs
udpPort
ipAddress
macAddress
sourceUdpPort
sourceIPAddr
sourceMacAddr
destUdpPort
destIpAddr
destMacAddr
mii_txen
clock
reset
offsetSec
offsetNano
drift
t1s
t1n
t2s
t2n
t3s
t3n
t4s
t4n
calcOffsetNano
calcOffsetSec
delayS
delayN
TDSlave
Application
I3

C.1.3 Application PTP

L'annexe page 55 est contenue dans le bloc 8 de l'annexe page 53. Elle contient :

1. Contrôle les *timeout* utilisés par la machine d'états de l'application.
2. Gère l'état du réseau (occupé, libre).
3. Machine d'états de l'esclave.

Le bloc n° 1 contient la machine d'états de la page 56.
Le bloc n° 2 est semblable a celui utilisé par le maître et contient la machine d'états de la page 46.
Le bloc n° 3 contient la machine d'états de la page 57.

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

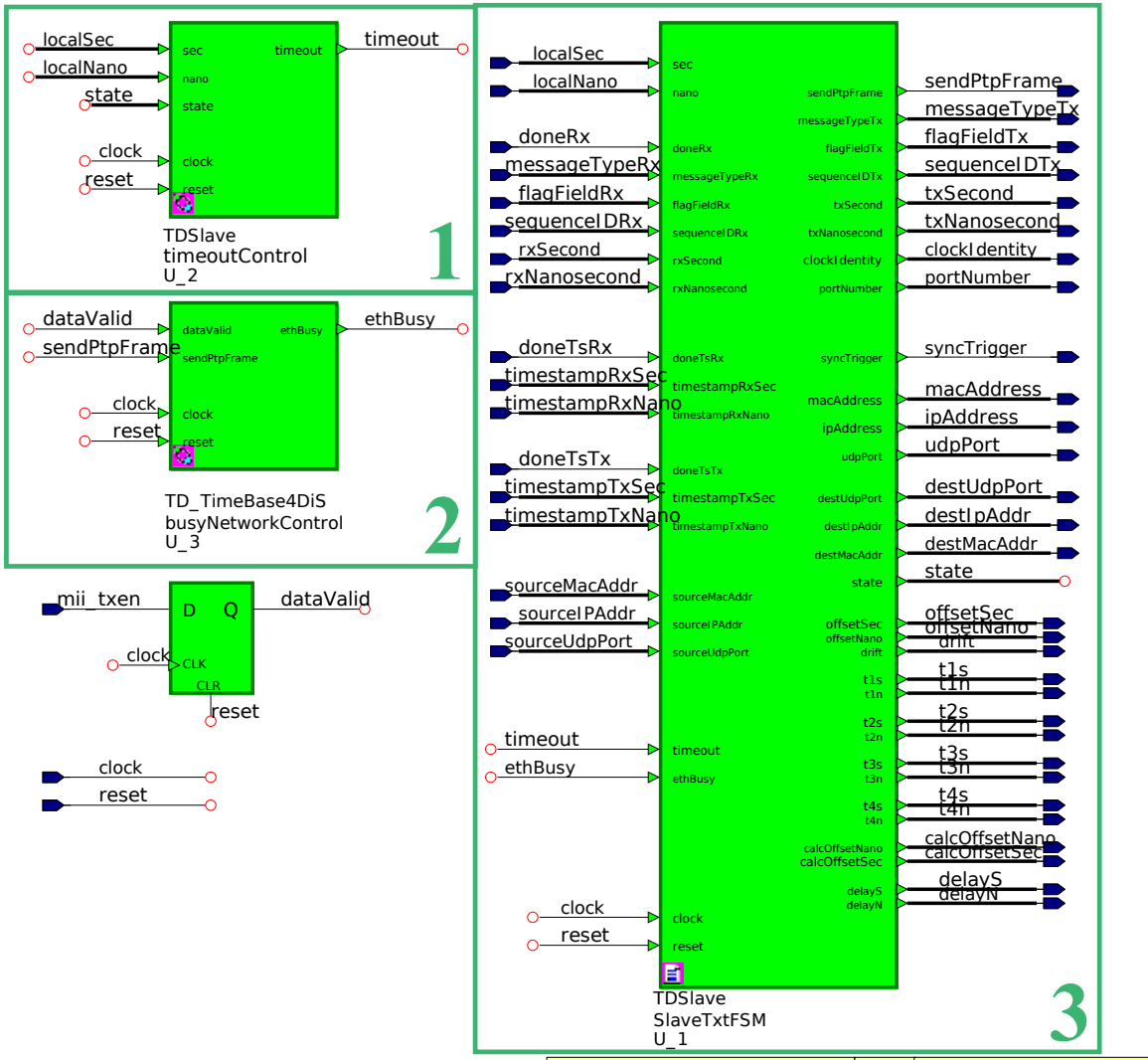
Declarations

Ports:

```
clock           : std_ulogic
doneRx          : std_ulogic
doneTsRx        : std_ulogic
doneTsTx        : std_ulogic
flagFieldRx     : std_ulogic_vector(15 DOWNTO 0)
localNano       : unsigned(UInt32BitNb-1 DOWNTO 0)
localSec        : unsigned(UInt48BitNb-1 DOWNTO 0)
messageTypeRx   : std_ulogic_vector(3 DOWNTO 0)
mii_txen        : std_uLogic
reset           : std_ulogic
rxNanosecond    : unsigned(UInt32BitNb-1 DOWNTO 0)
rxSecond        : unsigned(UInt48BitNb-1 DOWNTO 0)
sequenceIDRx    : std_ulogic_vector(15 DOWNTO 0)
sourceIPAddr    : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
sourceMacAddr   : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
sourceUdpPort   : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
timestampRxNano : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampRxSec  : unsigned(UInt48BitNb-1 DOWNTO 0)
timestampTxNano : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampTxSec  : unsigned(UInt48BitNb-1 DOWNTO 0)
calcOffsetNano  : signed(UInt32BitNb-1+4 DOWNTO 0)
calcOffsetSec   : signed(UInt48BitNb-1+4 DOWNTO 0)
clockIdentity   : std_ulogic_vector(79 DOWNTO 0)
debug           : std_ulogic_vector(15 DOWNTO 0)
delayN          : signed(UInt32BitNb-1+4 DOWNTO 0)
delayS          : signed(UInt48BitNb-1+4 DOWNTO 0)
destIpAddr      : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
destMacAddr     : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
destUdpPort     : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
drift           : signed(UInt32BitNb-1 DOWNTO 0)
flagFieldTx     : std_ulogic_vector(15 DOWNTO 0)
ipAddress       : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
macAddress      : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
messageTypeTx   : std_ulogic_vector(3 DOWNTO 0)
offsetNano      : signed(UInt32BitNb-1 DOWNTO 0)
offsetSec       : signed(UInt48BitNb-1 DOWNTO 0)
portNumber      : std_ulogic_vector(15 DOWNTO 0)
sendPtpFrame    : std_ulogic
sequenceIDTx    : std_ulogic_vector(15 DOWNTO 0)
syncTrigger     : std_ulogic
t1n             : unsigned(UInt32BitNb-1 DOWNTO 0)
t1s             : unsigned(UInt48BitNb-1 DOWNTO 0)
t2n             : unsigned(UInt32BitNb-1 DOWNTO 0)
t2s             : unsigned(UInt48BitNb-1 DOWNTO 0)
t3n             : unsigned(UInt32BitNb-1 DOWNTO 0)
t3s             : unsigned(UInt48BitNb-1 DOWNTO 0)
t4n             : unsigned(UInt32BitNb-1 DOWNTO 0)
t4s             : unsigned(UInt48BitNb-1 DOWNTO 0)
txNanosecond    : unsigned(UInt32BitNb-1 DOWNTO 0)
txSecond        : unsigned(UInt48BitNb-1 DOWNTO 0)
udpPort         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```

Diagram Signals:

```
SIGNAL dataValid       : std_uLogic
SIGNAL ethBusy         : std_ulogic
SIGNAL state           : std_ulogic_vector(3 DOWNTO 0)
SIGNAL timeout         : std_ulogic
```



HES-SO Valais		Project:
Title:	Application Esclave	Base de temps pour systèmes distribués
Path:	TDSlave/Application/struct	
Edited:	by drozj1 on 05 juil. 2012	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
timeout	OUT	'0'	'0'	CLKD
tempS	LOCAL		(others => '0')	CLKD
tempN	LOCAL		(others => '0')	CLKD
timeoutValue	LOCAL	X"02"	X"02"	CLKD

State Register Statements

Process Declarations
Clocked Process:
Output Process:

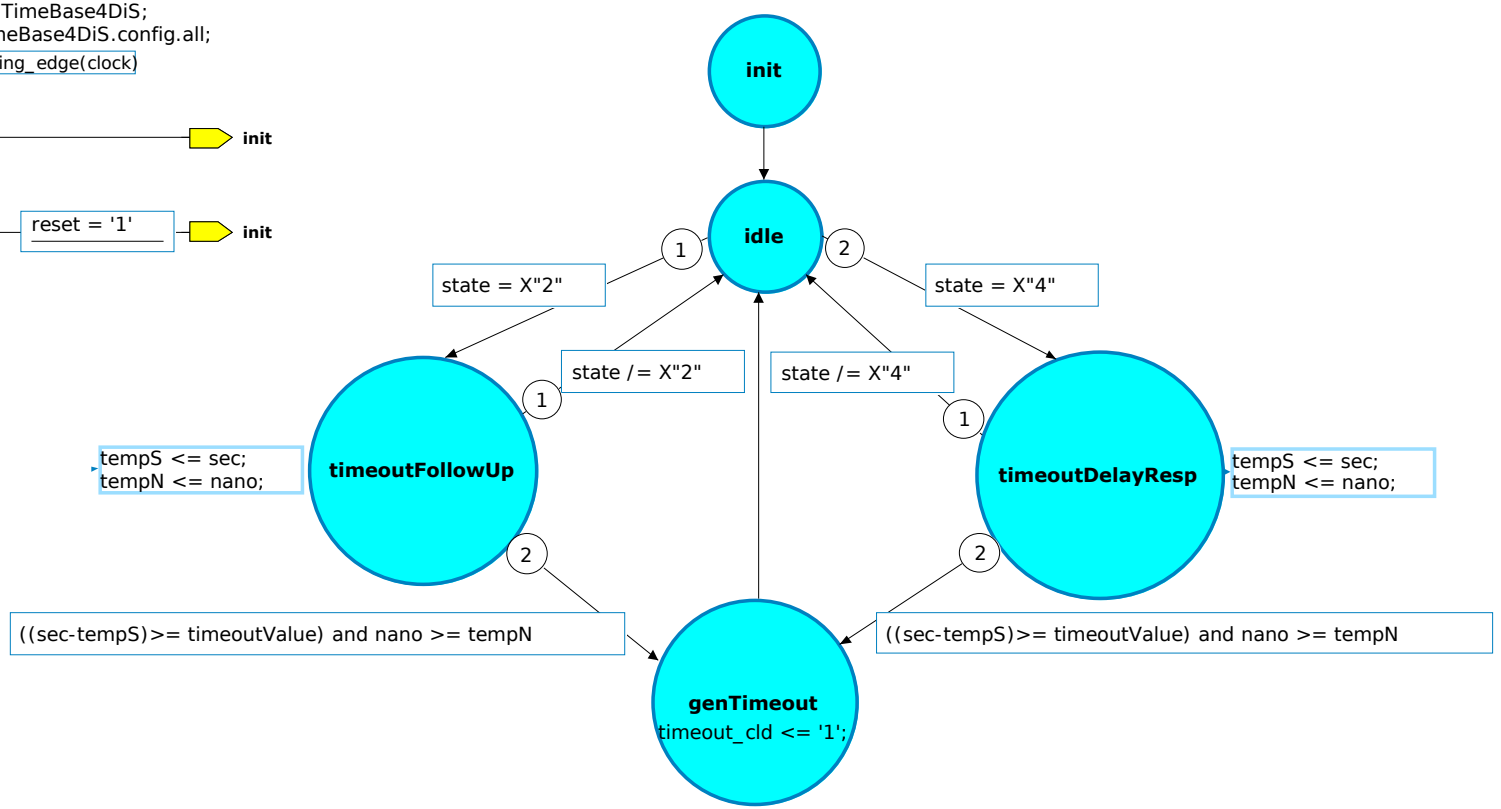
Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

clock: rising_ edge(clock)

init

reset: reset = '1' → init



HES-SO Valais		Project:	
Title:	Timeout Esclave FSM	Base de temps pour systèmes distribués	
Path:	TDSlave/timeoutControl/fsm		
Edited:	by drozj1 on 05 juil. 2012		

Global Actions
Pre Actions:
Post Actions:

Package List
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY TD_TimeBase4DiS;
  USE TD_TimeBase4DiS.config.all;
```

Concurrent Statements
```
sendPtpFrame <= sendPTP;
destIpAddr <= X"A9FEF05C";
destMacAddr <= X"E4AFA1391000";
destUdpPort <= X"013F";
ipAddress <= X"A9FEF05B";
macAddress <= X"E4AFA1391001";
udpPort <= X"013F";
```

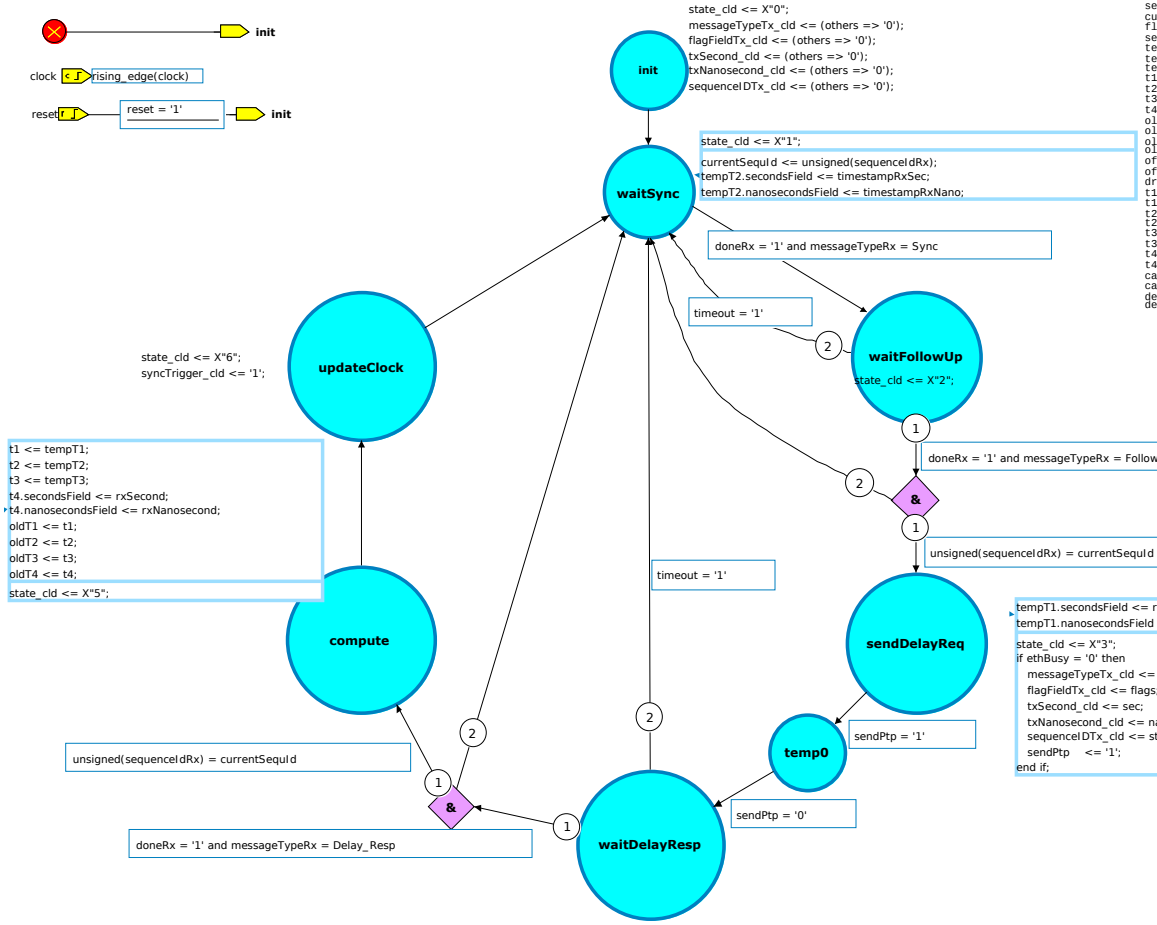
Architecture Declarations

Process Declarations
Clocked Process:
Output Process:

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
clockIdentity	OUT	(others => '0')		COMB
state	OUT	(others => '0')	(others => '0')	CLKD
destIpAddr	OUT			COMB
destMacAddr	OUT			COMB
destUdpPort	OUT			COMB
flagFieldTx	OUT		(others => '0')	CLKD
ipAddress	OUT			COMB
macAddress	OUT			COMB
messageTypeTx	OUT		(others => '0')	CLKD
portNumber	OUT	(others => '0')		COMB
sequenceIDTx	OUT		(others => '0')	CLKD
syncTrigger	OUT	'0'	'0'	CLKD
txNanosecond	OUT		(others => '0')	CLKD
txSecond	OUT		(others => '0')	CLKD
udpPort	OUT			COMB
sendPtp	LOCAL	'0'	'0'	CLKD
currentSequId	LOCAL		(others => '0')	CLKD
flags	LOCAL	X"0200"		COMB
sendPtpFrame	OUT			COMB
tempT1	LOCAL		((others => '0'),(others => '0'))	CLKD
tempT2	LOCAL		((others => '0'),(others => '0'))	CLKD
tempT3	LOCAL		((others => '0'),(others => '0'))	CLKD
t1	LOCAL		((others => '0'),(others => '0'))	CLKD
t2	LOCAL		((others => '0'),(others => '0'))	CLKD
t3	LOCAL		((others => '0'),(others => '0'))	CLKD
t4	LOCAL		((others => '0'),(others => '0'))	CLKD
oldT1	LOCAL		((others => '0'),(others => '0'))	CLKD
oldT2	LOCAL		((others => '0'),(others => '0'))	CLKD
oldT3	LOCAL		((others => '0'),(others => '0'))	CLKD
oldT4	LOCAL		((others => '0'),(others => '0'))	CLKD
offsetSec	OUT			COMB
offsetNano	OUT			COMB
drift	OUT			COMB
t1s	OUT			COMB
t1n	OUT			COMB
t2s	OUT			COMB
t2n	OUT			COMB
t3s	OUT			COMB
t3n	OUT			COMB
t4s	OUT			COMB
t4n	OUT			COMB
calcOffsetNano	OUT			COMB
calcOffsetSec	OUT			COMB
delayS	OUT			COMB
delayN	OUT			COMB

State Register Statements



HES-SO Valais		Project:
Title:	Application esclave FSM	Base de temps pour systèmes distribués
Path:	TDMaster/MasterFSM/fsm	
Edited:	by drozj1 on 05 juil. 2012	

ANNEXE D

Code VHDL

D.1 Package

```
1  --
2  -- VHDL Package Header TD_TimeBase4DiS.config
3  --
4  -- Created:
5  --          by - drozj1.UNKNOWN (WE4136)
6  --          at - 15:31:45 24.05.2012
7  --
8  -- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
9  --
10 LIBRARY ieee;
11 USE ieee.std_logic_1164.all;
12 USE ieee.numeric_std.ALL;
13 PACKAGE config IS
14
15   constant Enum4BitNb     : positive := 4;
16   constant Enum8BitNb     : positive := 8;
17   constant Enum16BitNb    : positive := 16;
18   constant UInt4BitNb     : positive := 4;
19   constant UInt8BitNb     : positive := 8;
20   constant UInt16BitNb    : positive := 16;
21   constant UInt32BitNb    : positive := 32;
22   constant UInt48BitNb    : positive := 48;
23   constant Int8BitNb      : positive := 8;
24   constant Int16BitNb     : positive := 16;
25   constant Int32BitNb     : positive := 32;
26   constant Int64BitNb     : positive := 64;
27   constant NibbleBitNb    : positive := 4;
28   constant OctetBitNb     : positive := 8;
29
30   ------------------------------------------------------------------------------
31   -- PTP messageTypes;
32   ------------------------------------------------------------------------------
33   constant Sync                   : std_ulogic_vector(3 downto 0) := X"0";
34   constant Delay_Req              : std_ulogic_vector(3 downto 0) := X"1";
35   constant Pdelay_Req             : std_ulogic_vector(3 downto 0) := X"2";
36   constant Pdelay_Resp            : std_ulogic_vector(3 downto 0) := X"3";
37   constant Reserved4              : std_ulogic_vector(3 downto 0) := X"4";
38   constant Reserved5              : std_ulogic_vector(3 downto 0) := X"5";
39   constant Reserved6              : std_ulogic_vector(3 downto 0) := X"6";
40   constant Reserved7              : std_ulogic_vector(3 downto 0) := X"7";
41   constant Follow_Up              : std_ulogic_vector(3 downto 0) := X"8";
42   constant Delay_Resp             : std_ulogic_vector(3 downto 0) := X"9";
43   constant Pdelay_Resp_Follow_Up  : std_ulogic_vector(3 downto 0) := X"A";
44   constant Announce               : std_ulogic_vector(3 downto 0) := X"B";
45   constant Signaling              : std_ulogic_vector(3 downto 0) := X"C";
46   constant Management             : std_ulogic_vector(3 downto 0) := X"D";
47   constant ReservedE              : std_ulogic_vector(3 downto 0) := X"E";
```

```
48  constant ReservedF              : std_ulogic_vector(3 downto 0) := X"F";
49
50
51  ------------------------------------------------------------------------------
52  -- Primitive PTP data types
53  ------------------------------------------------------------------------------
54
55  type Boolean            is (false, true);
56  subtype Enumeration4    is std_ulogic_vector(Enum4BitNb-1 downto 0);
57  subtype Enumeration8    is std_ulogic_vector(Enum8BitNb-1 downto 0);
58  subtype Enumeration16   is std_ulogic_vector(Enum16BitNb-1 downto 0);
59  subtype UInteger4       is unsigned(UInt4BitNb-1 downto 0);
60  subtype Integer8        is signed(Int8BitNb-1 downto 0);
61  subtype UInteger8       is unsigned(UInt8BitNb-1 downto 0);
62  subtype Integer16       is signed(Int16BitNb-1 downto 0);
63  subtype UInteger16      is unsigned(UInt16BitNb-1 downto 0);
64  subtype Integer32       is signed(Int32BitNb-1 downto 0);
65  subtype UInteger32      is unsigned(UInt32BitNb-1 downto 0);
66  subtype UInteger48      is unsigned(UInt48BitNb-1 downto 0);
67  subtype Integer64       is signed(Int64BitNb-1 downto 0);
68  subtype Nibble          is std_ulogic_vector(NibbleBitNb-1 downto 0);
69  subtype Octet           is std_ulogic_vector(OctetBitNb-1 downto 0);
70
71  type OctetArray is array (natural range <>) of Octet;
72  ------------------------------------------------------------------------------
73  -- Derived data type specifications
74  ------------------------------------------------------------------------------
75
76  type TimeInterval is record
77    scaledNanoseconds : Integer64;
78  end record;
79
80  type Timestamp is record
81    secondsField      : UInteger48;
82    nanosecondsField  : UInteger32;
83  end record;
84
85  subtype ClockIdentity is std_ulogic_vector(63 downto 0);
86
87  type PortIdentity is record
88    clockIdentity : ClockIdentity;
89    portNumber    : UInteger16;
90  end record;
91
92  type PortAddress is record
93    networkProtocol : Enumeration16;
94    addressLength   : UInteger16;
95    --addressField    : OctetArray(0 to to_integer(addressLength-1));
96  end record;
97
98  type ClockQuality is record
99    clockClass              : UInteger8;
100   clockAccuracy           : Enumeration8;
101   offsetScaledLogVariance : UInteger16;
102 end record;
103
104 type TLV is record
105   tlvType     : Enumeration16;
106   lengthField : UInteger16;
107   --valueField  : OctetArray(0 to to_integer(lengthField-1));
108 end record;
109
110 type PTPText is record
```

```
111     lengthField : UInteger8;
112     --textField   : OctetArray(0 to to_integer(lengthField-1));
113   end record;
114
115   type FaultRecord is record
116     faultRecordLength : UInteger16;
117     faultTime         : Timestamp;
118     severityCode      : Enumeration8;
119     faultName         : PTPText;
120     faultValue        : PTPText;
121     faultDescription  : PTPText;
122   end record;
123
124   ------------------------------------------------------------------------------
125   -- PTP data sets
126   ------------------------------------------------------------------------------
127
128   type defaultDS is record
129     twoStepFlag   : Boolean;
130     clockIdentity : ClockIdentity;
131     numberPorts   : UInteger16;
132     clockQuality  : ClockQuality;
133     priority1     : UInteger8;
134     priority2     : UInteger8;
135     domainNumber  : UInteger8;
136     slaveOnly     : Boolean;
137   end record;
138
139   type currentDS is record
140     stepsRemoved      : UInteger16;
141     offsetFromMaster  : TimeInterval;
142     meanPathDelay     : TimeInterval;
143   end record;
144
145   type parentDS is record
146     parentPortIdentity                    : PortIdentity;
147     parentStats                           : Boolean;
148     observedParentOffsetScaledLogVariance : UInteger16;
149     observedParentClockPhaseChangeRate    : Integer32;
150     grandmasterIdentity                   : ClockIdentity;
151     grandmasterClockQuality               : ClockQuality;
152     grandmasterPriority1                  : UInteger8;
153     grandmasterPriority2                  : UInteger8;
154   end record;
155
156   type timePropertiesDS is record
157     currentUtcOffset      : Integer16;
158     currentUtcOffsetValid : Boolean;
159     leap59                : Boolean;
160     leap61                : Boolean;
161     timeTraceable         : Boolean;
162     frequencyTraceable    : Boolean;
163     ptpTimescale          : Boolean;
164     timeSource            : Enumeration8;
165   end record;
166
167   type portDS is record
168     portIdentity            : PortIdentity;
169     portState               : Enumeration8;
170     logMinDelayReqInterval  : Integer8;
171     peerMeanPathDelay       : TimeInterval;
172     logAnnounceInterval     : Integer8;
173     announceReceiptTimeout  : UInteger8;
```

```
174     logSyncInterval          : Integer8;
175     delayMechanism           : Enumeration8;
176     logMinPdelayReqInterval  : Integer8;
177     versionNumber            : UInteger4;
178   end record;
179
180   type foreignMasterDS is record
181     foreignMasterPortIdentity     : PortIdentity;
182     --foreignMasterAnnounceMessages :
183   end record;
184 END config;
```

Code/config_pkg.vhd

D.2 Réception PTP

```
1  --
2  -- VHDL Architecture TD_TimeBase4DiS.PTPReceive.rtl
3  --
4  -- Created:
5  --          by - drozj1.UNKNOWN (WE4136)
6  --          at - 12:51:55 30.05.2012
7  --
8  -- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
9  --
10 ARCHITECTURE rtl OF PTPReceive IS
11
12   -- PTP Header
13   signal transportSpecific: std_ulogic_vector(3 downto 0);
14   signal msgType: std_ulogic_vector(3 downto 0);
15   signal versionPTP: std_ulogic_vector(7 downto 0);
16   signal messageLength: std_ulogic_vector(15 downto 0);
17   signal domainNumber: std_ulogic_vector(7 downto 0);
18   signal flags: std_ulogic_vector(15 downto 0);
19   signal correctionField: std_ulogic_vector(63 downto 0);
20   signal sourcePortIdentity: PortIdentity;
21   signal sequID: std_ulogic_vector(15 downto 0);
22   signal controlField: std_ulogic_vector(7 downto 0);
23   signal logMessageInterval: std_ulogic_vector(7 downto 0);
24   -- PTP Data
25   signal originTimestamp: Timestamp;
26   signal requestingPortIdentity: PortIdentity;
27
28   -- FSM Declaration
29   type unpackStateType is (
30     idle,
31     readHeader, readData
32   );
33   signal unpackState: unpackStateType;
34
35   signal addressCounter: unsigned(7 downto 0);
36   signal dataU: unsigned(rxData'range);
37   signal dataShiftRegister: unsigned(correctionField'range);
38   signal rxRead: std_ulogic;
39
40 BEGIN
41
42   ----------------------------------------------------------------------------
43                                                                   -- sequencer
44   unpackFsm: process(reset, clock)
45   begin
46     if reset = '1' then
47       unpackState <= idle;
48       done <= '0';
49     elsif rising_edge(clock) then
50       done <= '0';
51       case unpackState is
52         when idle =>
53           if rxEmpty = '0' then
54             unpackState <= readHeader;
55           end if;
56         when readHeader =>
57           if addressCounter = 3 and transportSpecific /= X"8" then
58             unpackState <= idle;
59           elsif addressCounter = 34 then
60             unpackState <= readData;
61           end if;
```

```
62          when readData =>
63            if addressCounter >= unsigned(messageLength) then
64              done <= '1';
65              unpackState <= idle;
66            end if;
67          when others => null;
68        end case;
69      end if;
70    end process unpackFsm;
71
72    ------------------------------------------------------------------------
73                                                          -- address counter
74    incrementAddress: process(reset, clock)
75    begin
76      if reset = '1' then
77        addressCounter <= (others => '0');
78      elsif rising_edge(clock) then
79        if unpackState = idle then
80          addressCounter <= ((0) => '1', others => '0');
81        elsif rxRead = '1' then
82          addressCounter <= addressCounter + 1;
83        end if;
84     end if;
85    end process incrementAddress;
86
87
88  ------------------------------------------------------------------------
89                                                          -- shift register
90    shiftData: process(reset, clock)
91    begin
92      if reset = '1' then
93        dataShiftRegister <= (others => '0');
94      elsif rising_edge(clock) then
95      if rxRead = '1' then
96        dataShiftRegister <= shift_left(dataShiftRegister, dataU'length);
97        dataShiftRegister(dataU'range) <= dataU;
98        end if;
99      end if;
100   end process shiftData;
101
102 ------------------------------------------------------------------------
103                                                       -- store informations
104   storeInfo: process(reset, clock)
105   begin
106     if reset = '1' then
107       transportSpecific <= (others => '0');
108       messageType <= (others => '0');
109       versionPTP <= (others => '0');
110       messageLength <= (others => '1');
111       domainNumber <= (others => '0');
112       flags <= (others => '0');
113       correctionField <= (others => '0');
114       sourcePortIdentity <= ((others => '0'),(others => '0'));
115       sequID <= (others => '0');
116       controlField <= (others => '0');
117       logMessageInterval <= (others => '0');
118       originTimestamp <= ((others => '0'),(others => '0'));
119       requestingPortIdentity <= ((others => '0'),(others => '0'));
120       second <= (others => '0');
121       nanosecond <= (others => '0');
122
123     elsif rising_edge(clock) then
124
```

```
125       if unpackState = readHeader then
126         if addressCounter = 1 then
127           transportSpecific <= std_ulogic_vector(dataShiftRegister(7 downto 4));
128           messageType <= std_ulogic_vector(dataShiftRegister(messageType'range));
129         elsif addressCounter = 2 then
130           versionPTP <= std_ulogic_vector(dataShiftRegister(versionPTP'range));
131         elsif addressCounter = 4 then
132           messageLength <= std_ulogic_vector(dataShiftRegister(messageLength'range))
                ;
133         elsif addressCounter = 5 then
134           domainNumber <= std_ulogic_vector(dataShiftRegister(domainNumber'range));
135         elsif addressCounter = 8 then
136           flags <= std_ulogic_vector(dataShiftRegister(flags'range));
137         elsif addressCounter = 16 then
138           correctionField <= std_ulogic_vector(dataShiftRegister(correctionField'
                range));
139         elsif addressCounter = 28 then
140           sourcePortIdentity.clockIdentity <= std_ulogic_vector(dataShiftRegister(
                sourcePortIdentity.clockIdentity'range));
141         elsif addressCounter = 30 then
142           sourcePortIdentity.portNumber <= dataShiftRegister(sourcePortIdentity.
                portNumber'range);
143         elsif addressCounter = 32 then
144           sequID <= std_ulogic_vector(dataShiftRegister(sequID'range));
145         elsif addressCounter = 33 then
146           controlField <= std_ulogic_vector(dataShiftRegister(controlField'range));
147         elsif addressCounter = 34 then
148           logMessageInterval <= std_ulogic_vector(dataShiftRegister(
                logMessageInterval'range));
149         end if;
150       elsif unpackState = readData then
151         if addressCounter = 40 then
152           second <= dataShiftRegister(second'range);
153         elsif addressCounter = 44 then
154           nanosecond <= dataShiftRegister(nanosecond'range);
155         end if;
156       end if;
157     end if;
158   end process storeInfo;
159
160   rxRd <= rxRead;
161   flagField <= flags;
162   sequenceID <= sequID;
163   dataU <= unsigned(rxData);
164   rxRead <= not rxEmpty;
165
166 END ARCHITECTURE rtl;
```

Code/PTPReceive_rtl.vhd

D.3 Émission PTP

```
--
-- VHDL Architecture TD_TimeBase4DiS.PTPSend.rtl
--
-- Created:
--          by - drozj1.UNKNOWN (WE4136)
--          at - 11:29:52 31.05.2012
--
-- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
--
ARCHITECTURE rtl OF PTPSend IS
  -- PTP Header
  signal transportSpecific: std_ulogic_vector(3 downto 0);
  signal msgType: std_ulogic_vector(3 downto 0);
  signal versionPTP: std_ulogic_vector(7 downto 0);
  signal messageLength: std_ulogic_vector(15 downto 0);
  signal domainNumber: std_ulogic_vector(7 downto 0);
  signal flags: std_ulogic_vector(15 downto 0);
  signal correctionField: std_ulogic_vector(63 downto 0);
  signal sourcePortIdentity: PortIdentity;
  signal sequID: std_ulogic_vector(15 downto 0);
  signal controlField: std_ulogic_vector(7 downto 0);
  signal logMessageInterval: std_ulogic_vector(7 downto 0);
  -- PTP Data
  signal originTimestamp: Timestamp;
  signal requestingPortIdentity: PortIdentity;

  type packStateType is (
    idle,
    writeHeader, writeData, endOfFrame
  );
  signal packState: packStateType;

  signal addressCounter: unsigned(7 downto 0);
  signal headerData: unsigned(txData'range);
  signal txWrite: std_ulogic;

BEGIN

  ----------------------------------------------------------------------------
                                                        -- Acquire Data
  process(clock, reset)
  begin
    if reset = '1' then
      transportSpecific <= (others =>'0');
      msgType <= (others =>'0');
      versionPTP <= (others =>'0');
      messageLength <= (others =>'0');
      domainNumber <= (others =>'0');
      flags <= (others =>'0');
      correctionField <= (others =>'0');
      sourcePortIdentity <= ((others =>'0'),(others =>'0'));
      sequID <= (others =>'0');
      controlField <= (others =>'0');
      logMessageInterval <= (others =>'0');
      -- PTP Data
      originTimestamp <= ((others =>'0'),(others =>'0'));
      requestingPortIdentity <= ((others =>'0'),(others =>'0'));
    elsif rising_edge(clock) then
      if send = '1' then
        transportSpecific <= X"8";
        msgType <= messageType;
```

```
62          versionPTP <= X"02";
63          if messageType = X"9" then
64            messageLength <= X"0036";
65          else
66            messageLength <= X"002C";
67          end if;
68          domainNumber <= (others =>'0');
69          flags <= flagField;
70          correctionField <= (others =>'0');
71          sourcePortIdentity <= ((others =>'0'),(others =>'0'));
72          sequID <= sequenceID;
73          controlField <= X"00";
74          logMessageInterval <= X"7F";
75          -- PTP Data
76          originTimestamp <= (second, nanosecond);
77          requestingPortIdentity <= (X"0123456789ABCDEF",X"ABEF");
78        end if;
79      end if;
80    end process;
81
82    ------------------------------------------------------------------------
83                                                          -- sequencer
84    packFsm: process(reset, clock)
85    begin
86      if reset = '1' then
87        packState <= idle;
88      elsif rising_edge(clock) then
89        case packState is
90          when idle =>
91            if send = '1' then
92              packState <= writeHeader;
93            end if;
94          when writeHeader =>
95            if addressCounter >= 34 then
96              packState <= writeData;
97            end if;
98          when writeData =>
99            if addressCounter = unsigned(messageLength)-1 then
100             packState <= endOfFrame;
101           end if;
102         when endOfFrame =>
103             packState <= idle;
104         when others => null;
105       end case;
106     end if;
107   end process packFsm;
108
109   ------------------------------------------------------------------------
110                                                      -- address counter
111   incrementAddress: process(reset, clock)
112   begin
113     if reset = '1' then
114       addressCounter <= (others => '0');
115     elsif rising_edge(clock) then
116       case packState is
117         when idle =>
118           addressCounter <= (others => '0');
119         when writeHeader | writeData =>
120           if txFull = '0' then
121             addressCounter <= addressCounter + 1;
122           end if;
123         when others => null;
124       end case;
```

```
125     end if;
126   end process incrementAddress;
127   ------------------------------------------------------------------------------
128                                                                   -- build header data
129   buildHeaderData: process(
130     addressCounter, txWrite
131   )
132   begin
133     case to_integer(addressCounter) is
134       -- PTP Header
135       when  0 =>
136         headerData(7 downto 4) <= unsigned(transportSpecific);
137         headerData(3 downto 0) <= unsigned(msgType);
138       when  1 => headerData <= unsigned(versionPTP);
139       when  2 => headerData <= resize(shift_right(unsigned(messageLength),
              headerData'length), headerData'length);
140       when  3 => headerData <= resize(unsigned(messageLength), headerData'length);
141       when  4 => headerData <= unsigned(domainNumber);
142       when  5 => headerData <= X"00";
143       when  6 => headerData <= resize(shift_right(unsigned(flags), headerData'length
              ), headerData'length);
144       when  7 => headerData <= resize(unsigned(flags), headerData'length);
145       when  8 | 9 |10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19  => headerData <=
            X"00";
146       when  20 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 7*headerData'length), headerData'length);
147       when  21 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 6*headerData'length), headerData'length);
148       when  22 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 5*headerData'length), headerData'length);
149       when  23 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 4*headerData'length), headerData'length);
150       when  24 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 3*headerData'length), headerData'length);
151       when  25 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 2*headerData'length), headerData'length);
152       when  26 => headerData <= resize(shift_right(unsigned(sourcePortIdentity.
            clockIdentity), 1*headerData'length), headerData'length);
153       when  27 => headerData <= resize(unsigned(sourcePortIdentity.clockIdentity),
            headerData'length);
154       when  28 => headerData <= resize(shift_right(sourcePortIdentity.portNumber,
            headerData'length), headerData'length);
155       when  29 => headerData <= resize(sourcePortIdentity.portNumber,headerData'
            length);
156       when  30 => headerData <= resize(shift_right(unsigned(sequID),headerData'
            length), headerData'length);
157       when  31 => headerData <= resize(unsigned(sequID),headerData'length);
158       when  32 => headerData <= unsigned(controlField);
159       when  33 => headerData <= unsigned(logMessageInterval);
160       when  34 => headerData <= resize(shift_right(originTimestamp.secondsField,5*
            headerData'length), headerData'length);
161       when  35 => headerData <= resize(shift_right(originTimestamp.secondsField,4*
            headerData'length), headerData'length);
162       when  36 => headerData <= resize(shift_right(originTimestamp.secondsField,3*
            headerData'length), headerData'length);
163       when  37 => headerData <= resize(shift_right(originTimestamp.secondsField,2*
            headerData'length), headerData'length);
164       when  38 => headerData <= resize(shift_right(originTimestamp.secondsField,1*
            headerData'length), headerData'length);
165       when  39 => headerData <= resize(originTimestamp.secondsField, headerData'
            length);
166       when  40 => headerData <= resize(shift_right(originTimestamp.nanosecondsField
            ,3*headerData'length), headerData'length);
```

```
167       when  41 => headerData <= resize(shift_right(originTimestamp.nanosecondsField
              ,2*headerData'length), headerData'length);
168       when  42 => headerData <= resize(shift_right(originTimestamp.nanosecondsField
              ,1*headerData'length), headerData'length);
169       when  43 => headerData <= resize(originTimestamp.nanosecondsField, headerData'
              length);
170       when  44 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 7*headerData'length), headerData'length);
171       when  45 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 6*headerData'length), headerData'length);
172       when  46 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 5*headerData'length), headerData'length);
173       when  47 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 4*headerData'length), headerData'length);
174       when  48 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 3*headerData'length), headerData'length);
175       when  49 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 2*headerData'length), headerData'length);
176       when  50 => headerData <= resize(shift_right(unsigned(requestingPortIdentity.
              clockIdentity), 1*headerData'length), headerData'length);
177       when  51 => headerData <= resize(unsigned(requestingPortIdentity.clockIdentity
              ), headerData'length);
178       when  52 => headerData <= resize(shift_right(requestingPortIdentity.portNumber
              ,headerData'length), headerData'length);
179       when  53 => headerData <= resize(requestingPortIdentity.portNumber,headerData'
              length);
180       when others => headerData <= (others => '-');
181     end case;
182   end process buildHeaderData;
183
184   packControls: process(
185     packState,
186     addressCounter, headerData,
187     txFull
188   )
189   begin
190     txWrite <= '0';
191     case packState is
192        when writeHeader | writeData =>
193         if txFull = '0' then
194           txWrite <= '1';
195           txData <= std_ulogic_vector(headerData);
196         end if;
197       when others => null;
198     end case;
199   end process packControls;
200
201   txWr <= txWrite;
202 END ARCHITECTURE rtl;
```

Code/PTPSend_rtl.vhd

D.4 Horloge

```
1  --
2  -- VHDL Architecture TD_TimeBase4DiS.PTPClockCounter.rtl
3  --
4  -- Created:
5  --          by - drozj1.UNKNOWN (WE4136)
6  --          at - 11:26:24 04.06.2012
7  --
8  -- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
9  --
10 ARCHITECTURE rtl OF PTPClockCounter IS
11   constant margeBit : positive := 2;
12   signal second: unsigned(sec'left+margeBit downto 0);
13   signal nanosecond: signed(nano'left+margeBit downto 0);
14   signal fractionNano: signed(nano'left+margeBit downto 0);
15   signal carryNano: std_ulogic;
16   signal carrySec: std_ulogic;
17   signal incrementFractionNano: signed(nano'left+margeBit downto 0);
18   signal incrementNano: signed(nano'left+margeBit downto 0);
19   constant defaultFractionNanoIncrement: signed(nano'left+margeBit downto 0) :=
       to_signed(151515151, nanosecond'length);
20   constant defaultNanoIncrement: signed(nano'left+margeBit downto 0) := to_signed
       (15,nanosecond'length);
21
22 BEGIN
23
24   sec <= resize(unsigned(second),sec'length);
25
26   nanosecondsOutput:process(nanosecond,reset)
27   begin
28     if nanosecond < 0 or reset = '1' then
29       nano <= (others => '0');
30     else
31       nano <= resize(unsigned(nanosecond),nano'length);
32     end if;
33   end process;
34
35   -- increment the fraction of nanosecond every rising edge of clock signal
36   -- update carry signal and manage overflow
37   fractionsNano:process(clock, reset)
38   begin
39     if reset = '1' then
40       fractionNano <= (others => '0');
41       carryNano <= '0';
42       incrementFractionNano <= defaultFractionNanoIncrement;
43     elsif rising_edge(clock) then
44
45       if fractionNano >= 1e9 then
46         fractionNano <= fractionNano + incrementFractionNano - 1e9;
47         carryNano <= '1';
48       else
49         fractionNano <= fractionNano + incrementFractionNano;
50         carryNano <= '0';
51       end if;
52
53     end if;
54   end process;
55
56   -- increment the nanosecond every rising edge of clock signal
57   -- update carry signal and manage overflow
58   -- possibility to change nanosecond value
59   ns:process(clock, reset)
```

```
60  begin
61    if reset = '1' then
62      nanosecond <= (others => '0');
63      carrySec <= '0';
64      incrementNano <= defaultNanoIncrement;
65    elsif rising_edge(clock) then
66
67      if syncTrigger = '1' then           -- update clock
68        if nanosecond >= 1e9 then
69          if carryNano = '1' then
70            nanosecond <= resize(nanosecond + incrementNano - 1e9 + 1 - offsetNano,
                  nanosecond'length);
71          else
72            nanosecond <= resize(nanosecond + incrementNano - 1e9 - offsetNano,
                  nanosecond'length);
73          end if;
74          carrySec <= '1';
75        else
76          if carryNano = '1' then
77            nanosecond <= resize(nanosecond + incrementNano + 1 - offsetNano,
                  nanosecond'length);
78          else
79            nanosecond <= resize(nanosecond + incrementNano - offsetNano, nanosecond
                  'length);
80          end if;
81          carrySec <= '0';
82        end if;
83      else
84        if nanosecond >= 1e9 then
85          if carryNano = '1' then
86            nanosecond <= nanosecond + incrementNano - 1e9 + 1;
87          else
88            nanosecond <= nanosecond + incrementNano - 1e9;
89          end if;
90          carrySec <= '1';
91        else
92          if carryNano = '1' then
93            nanosecond <= nanosecond + incrementNano + 1;
94          else
95            nanosecond <= nanosecond + incrementNano;
96          end if;
97          carrySec <= '0';
98        end if;
99      end if;
100
101   end if;
102 end process;
103
104 -- increment the second every carry from nanosecond process
105 -- possibility to change second value
106 -- generate PPS signal
107 s:process(clock, reset)
108 begin
109   if reset = '1' then
110     second <= to_unsigned(1341835200,second'length); -- set a date in reset
111     PPS <= '0';
112   elsif rising_edge(clock) then
113
114     PPS <= '0';
115     if syncTrigger = '1' then
116       if carrySec = '1' then
117         second <= resize(unsigned(signed(second + 1) - offsetSec),second'length);
118         PPS <= '1';
```

```
119         else
120           second <= resize(unsigned(signed(second) - offsetSec), second'length);
121         end if;
122       else
123         if carrySec = '1' then
124           second <= second + 1;
125           PPS <= '1';
126         end if;
127       end if;
128
129     end if;
130   end process;
131
132 END ARCHITECTURE rtl;
```

Code/PTPClockCounter_rtl.vhd

D.5 Émission RS-232

```
1  --
2  -- VHDL Architecture TD_TimeBase4DiS.SendSerial.rtl
3  --
4  -- Created:
5  --          by - drozj1.UNKNOWN (WE4136)
6  --          at - 08:11:45 12.06.2012
7  --
8  -- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
9  --
10
11 ARCHITECTURE rtl OF SendSerial IS
12   signal offsetSecReg: signed(offsetSec'range);
13   signal offsetNanoReg: signed(offsetNano'range);
14   signal driftNanoReg:  signed(driftNano'range);
15   signal addressCounter: unsigned(7 downto 0);
16   signal dataShiftRegister: std_ulogic_vector(offsetSec'range);
17   signal fifoWrite:std_ulogic;
18   constant addressMax: positive := 168;
19   type sendStateType is (
20     idle,
21     send
22   );
23   signal sendState: sendStateType;
24 BEGIN
25
26   ----------------------------------------------------------------------
27                                                          -- sequencer
28  sendFsm: process(reset, clock)
29   begin
30     if reset = '1' then
31       sendState <= idle;
32       offsetSecReg <= (others => '0');
33       offsetNanoReg <= (others => '0');
34       driftNanoReg <= (others => '0');
35     elsif rising_edge(clock) then
36       case sendState is
37         when idle =>
38           if syncTrigger = '1' then
39             sendState <= send;
40             offsetNanoReg <= offsetNano;
41             offsetSecReg <= offsetSec;
42             driftNanoReg <= driftNano;
43           end if;
44         when send =>
45           if addressCounter >= addressMax then
46             sendState <= idle;
47           end if;
48       end case;
49     end if;
50   end process sendFsm;
51
52   tranmitControls: process(
53     sendState,
54     txFull,
55     addressCounter
56   )
57   begin
58     fifoWrite <= '0';
59     case sendState is
60       when send =>
61         if addressCounter > 0 then
```

```
62            fifoWrite <= not txFull;
63          end if;
64        when others => null;
65      end case;
66    end process tranmitControls;
67
68    txWrUART <= fifoWrite;
69  ------------------------------------------------------------------------------
70                                                              -- shift register
71    process(dataShiftRegister,addressCounter)
72      variable data8 : unsigned(txDataUart'range);
73    begin
74
75      data8 := resize(unsigned(dataShiftRegister(dataShiftRegister'high downto
          dataShiftRegister'high-3)),txDataUart'length);
76
77      if data8 >= 0 and data8 <= 9 then
78        txDataUart <= std_ulogic_vector(data8 + 48);
79      elsif data8 >= 10 and data8 <= 15 then
80        txDataUart <= std_ulogic_vector(data8 + 55);
81      else
82        txDataUart <= std_ulogic_vector(data8);
83      end if;
84      if addressCounter = 13 or addressCounter = 22 or addressCounter = 44
85      or addressCounter = 53 or addressCounter = 66 or addressCounter = 75
86      or addressCounter = 88 or addressCounter = 97 or addressCounter = 110
87      or addressCounter = 133 or addressCounter = 143 or addressCounter = 157 then
88        txDataUart <= X"3A";
89      elsif addressCounter = 31 or addressCounter = 119 then
90        txDataUart <= X"2F";
91      elsif addressCounter = 167 then
92        txDataUart <= X"25"; --% = end of transmission
93      end if;
94    end process;
95
96    ------------------------------------------------------------------------------
97                                                                  -- build data
98    shiftData: process(reset, clock)
99    begin
100     if reset = '1' then
101       dataShiftRegister <= (others => '0');
102     elsif rising_edge(clock) then
103       if addressCounter = 0 then
104         dataShiftRegister <= std_ulogic_vector(offsetSecReg);
105       end if;
106       if fifoWrite = '1' then
107         dataShiftRegister(dataShiftRegister'high downto dataShiftRegister'low+4)
108             <= dataShiftRegister(dataShiftRegister'high-4 downto dataShiftRegister'
                  low);
109         if addressCounter = 9 then
110           dataShiftRegister(offsetNanoReg'range) <= std_ulogic_vector(offsetNanoReg)
                ;
111         elsif addressCounter = 18 then
112           dataShiftRegister(driftNanoReg'range) <= std_ulogic_vector(driftNanoReg);
113         elsif addressCounter = 31 then
114           dataShiftRegister <= std_ulogic_vector(t1s);
115         elsif addressCounter = 40 then
116           dataShiftRegister(t1n'range) <= std_ulogic_vector(t1n);
117         elsif addressCounter = 53 then
118           dataShiftRegister <= std_ulogic_vector(t2s);
119         elsif addressCounter = 62 then
120           dataShiftRegister(t1n'range) <= std_ulogic_vector(t2n);
121         elsif addressCounter = 75 then
```

```
122           dataShiftRegister <= std_ulogic_vector(t3s);
123         elsif addressCounter = 84 then
124           dataShiftRegister(t1n'range) <= std_ulogic_vector(t3n);
125         elsif addressCounter = 97 then
126           dataShiftRegister <= std_ulogic_vector(t4s);
127         elsif addressCounter = 106 then
128           dataShiftRegister(t1n'range) <= std_ulogic_vector(t4n);
129         elsif addressCounter = 119 then
130           dataShiftRegister <= std_ulogic_vector(delayS(delayS'high downto 4));
131         elsif addressCounter = 120 then
132           dataShiftRegister(3 downto 0) <= std_ulogic_vector(delayS(3 downto 0));
133         elsif addressCounter = 130 then
134           dataShiftRegister(35 downto 0) <= std_ulogic_vector(delayN);
135         elsif addressCounter = 143 then
136           dataShiftRegister <= std_ulogic_vector(calcOffsetS(calcOffsetS'high downto
                4));
137         elsif addressCounter = 144 then
138           dataShiftRegister(3 downto 0) <= std_ulogic_vector(calcOffsetS(3 downto 0)
              );
139         elsif addressCounter = 154 then
140           dataShiftRegister(35 downto 0) <= std_ulogic_vector(calcOffsetN);
141         end if;
142
143       end if;
144     end if;
145   end process shiftData;
146
147   ------------------------------------------------------------------------------
148                                                                 -- address counter
149   incrementAddress: process(reset, clock)
150   begin
151     if reset = '1' then
152       addressCounter <= (others => '0');
153     elsif rising_edge(clock) then
154       if sendState = idle then
155         addressCounter <= (others => '0');
156       elsif sendState = send then
157         if txFull = '0' then
158           addressCounter <= addressCounter + 1;
159         end if;
160       end if;
161     end if;
162   end process incrementAddress;
163
164 END ARCHITECTURE rtl;
```

Code/SendSerial_rtl.vhd

ANNEXE E

Tests

E.1 Timestamp

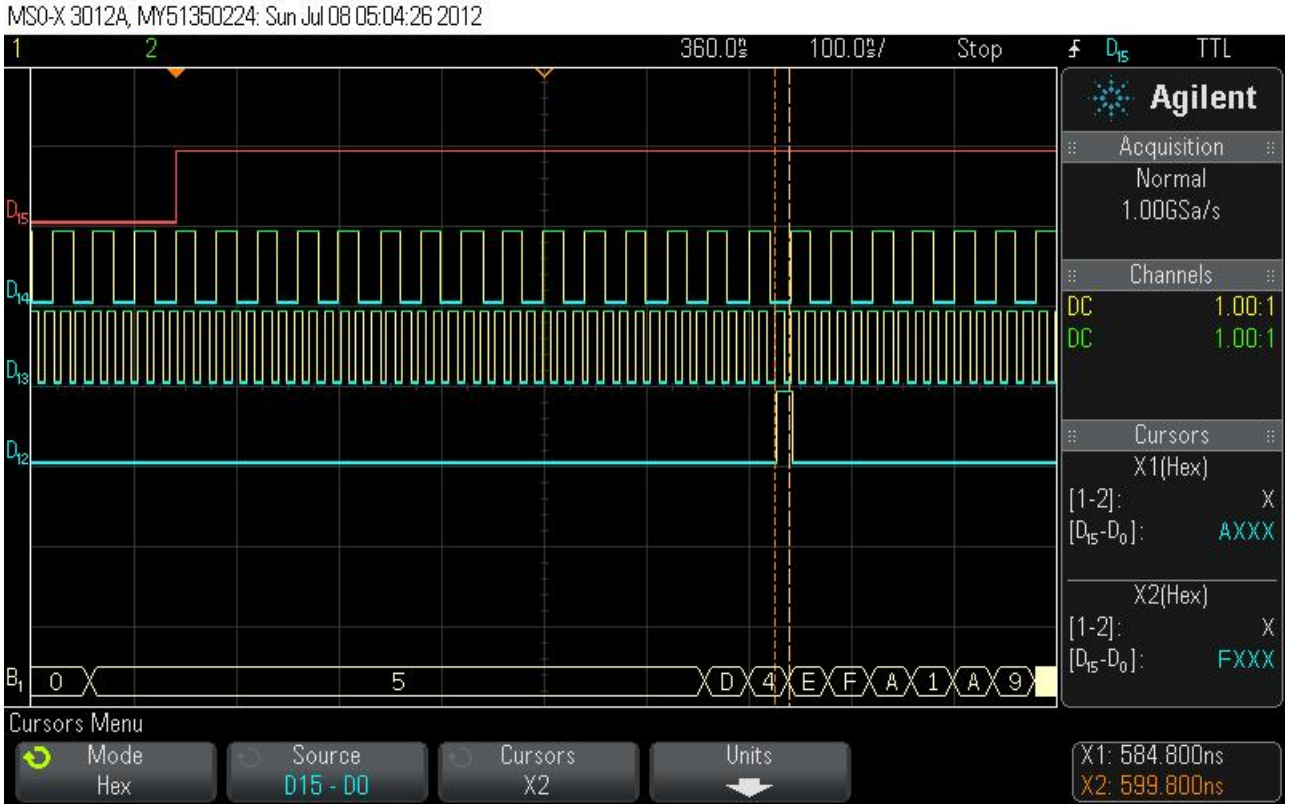


FIGURE E.1 – *Timestamp* à la réception

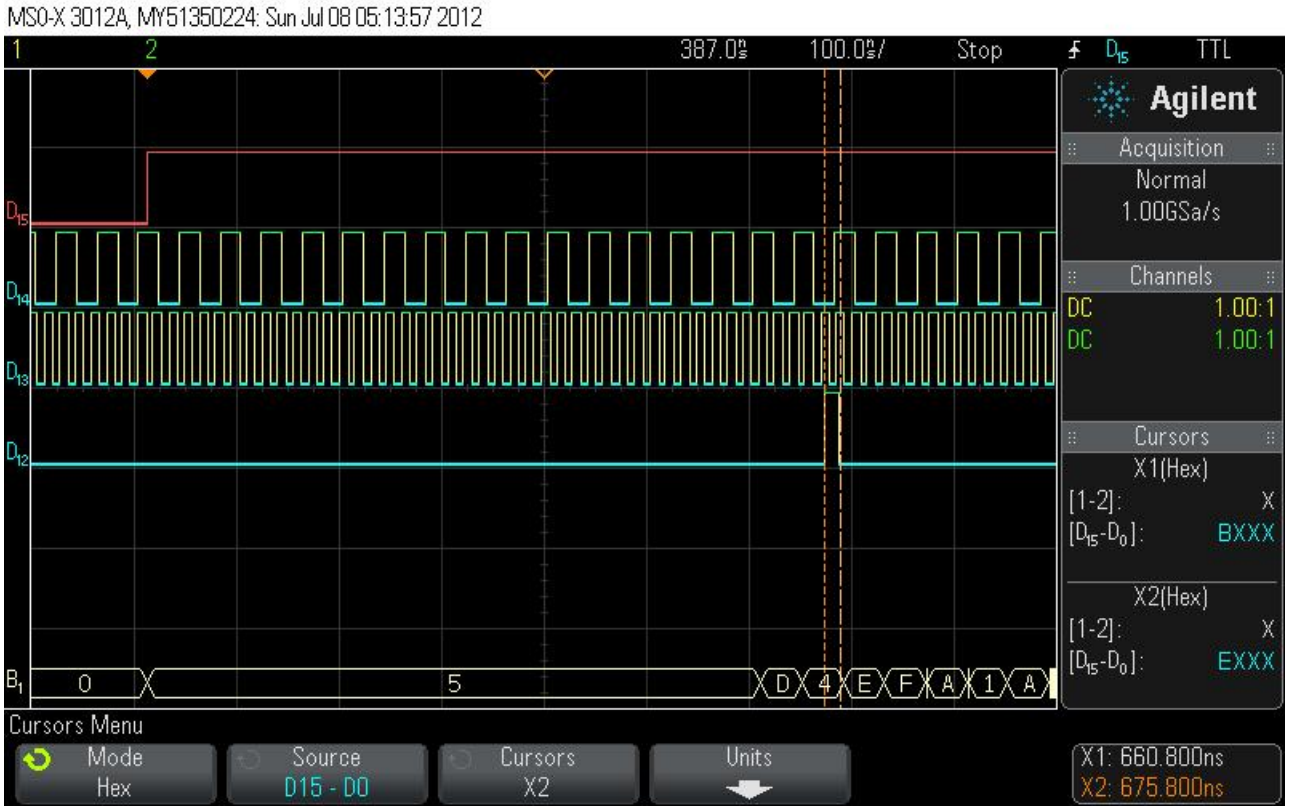


FIGURE E.2 – *Timestamp* à l'émission

Signal n°	Description
D_{15}	Données Valides
D_{14}	Horloge spécifique à l'émission/réception d'une trame Ethernet
D_{13}	Horloge de la FPGA
D_{12}	Exécution du timstamp
B_1	Bus qui représente les données sous forme hexadécimale

TABLE E.1 – Description des signaux des figures E.1 et E.2

E.2 Échange de messages

E.2.1 Paquets Ethernet

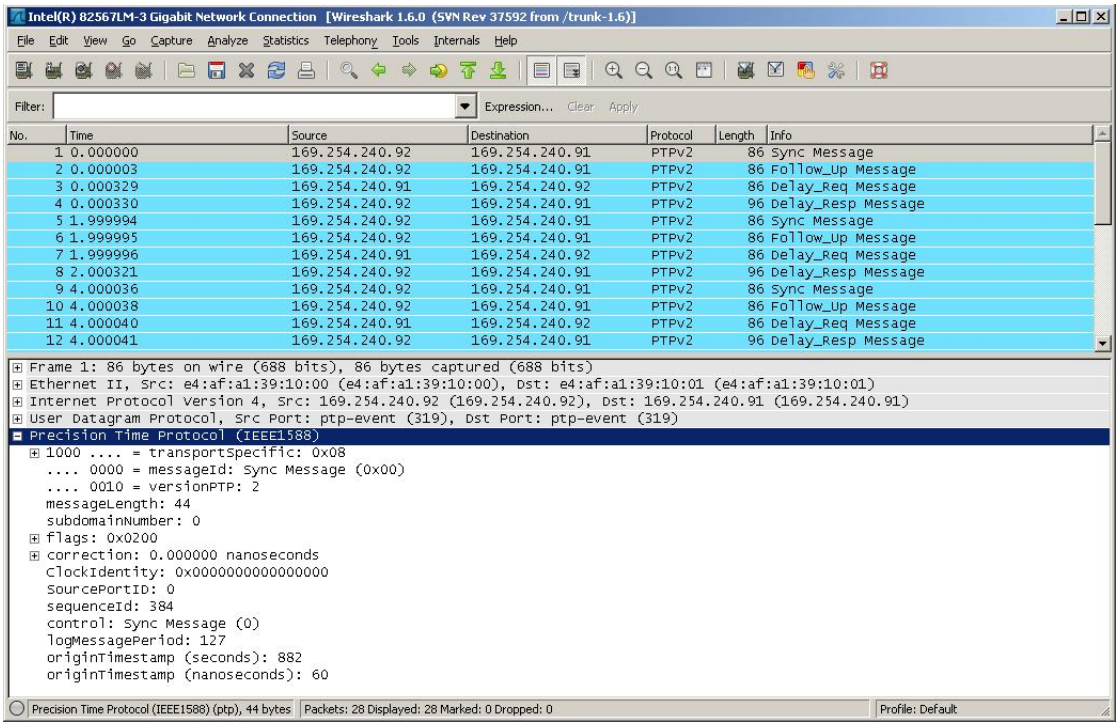


FIGURE E.3 – Capture des messages échangés entre le maître et l'esclave

E.2.2 Données RS-232

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Offset seconde	Offset nanoseconde	t1 sec	t1 nano	t2 sec	t2 nano	t3 sec	t3 nano	t4 sec	t4 nano	DelayS	DelayN	Vérification Delay	Vérification Offset seconde	Vérification Offset nanoseconde
2	10	326469652	1341835202	9226	1341835212	326475211	1341835212	326649741	1341835202	191090	0	3667	3667	10	326469652
3	10	326481545	1341835204	275120	1341835214	326752968	1341835214	326927771	1341835204	457317	0	3697	3697	10	326481545
4	10	326493189	1341835206	541423	1341835216	327031105	1341835216	327205787	1341835206	723120	0	3507	3507	10	326493189
5	10	326505484	1341835208	806938	1341835218	327308756	1341835218	327483423	1341835208	988938	0	3666	3666	10	326505484
6	10	326517038	1341835210	1072847	1341835220	327586408	1341835220	327761044	1341835210	1254438	0	3477	3477	10	326517038
7	10	326529356	1341835212	1338362	1341835222	327864074	1341835222	328038680	1341835212	1520256	0	3644	3644	10	326529356
8	10	326541069	1341835214	1604271	1341835224	328141726	1341835224	328316301	1341835214	1786074	0	3614	3614	10	326541069
9	10	326553226	1341835216	1870150	1341835226	328419786	1341835226	328594332	1341835216	2051877	0	3590	3590	10	326553226
10	10	326564977	1341835218	2136029	1341835228	328697438	1341835228	328871953	1341835218	2317680	0	3568	3568	10	326564977
11	10	326577379	1341835220	2401892	1341835230	328975498	1341835230	329150362	1341835220	2584302	0	3773	3773	10	326577379
12	10	326588977	1341835222	2668135	1341835232	329253544	1341835232	329428377	1341835222	2850105	0	3568	3568	10	326588977
13	10	326600803	1341835224	2933968	1341835234	329531195	1341835234	329705998	1341835224	3115923	0	3576	3576	10	326600803
14	10	326613189	1341835226	3199786	1341835236	329809240	1341835236	329984013	1341835226	3382029	0	3735	3735	10	326613189
15	10	326625030	1341835228	3465983	1341835238	330087301	1341835238	330262028	1341835228	3648135	0	3712	3712	10	326625030
16	10	326636885	1341835230	3732165	1341835240	330365361	1341835240	330540058	1341835230	3914241	0	3689	3689	10	326636885
17	10	326648773	1341835232	3998331	1341835242	330643422	1341835242	330818074	1341835232	4180347	0	3682	3682	10	326648773
18	10	326660652	1341835234	4264498	1341835244	330921483	1341835244	331096104	1341835234	4446453	0	3667	3667	10	326660652
19	10	326672561	1341835236	4530649	1341835246	331199543	1341835246	331374134	1341835236	4712574	0	3667	3667	10	326672561
20	10	326684280	1341835238	4796786	1341835248	331477604	1341835248	331652164	1341835238	4978271	0	3462	3462	10	326684280
21	10	326696204	1341835240	5062119	1341835250	331754861	1341835250	331929391	1341835240	5243574	0	3462	3462	10	326696204
22	10	326708326	1341835242	5327437	1341835252	332032134	1341835252	332207028	1341835242	5509589	0	3629	3629	10	326708326
23	10	326720090	1341835244	5593543	1341835254	332310194	1341835254	332485058	1341835244	5775286	0	3439	3439	10	326720090
24	10	326732219	1341835246	5859225	1341835256	332587937	1341835256	332762679	1341835246	6040982	0	3507	3507	10	326732219
25	10	326744355	1341835248	6124922	1341835258	332865694	1341835258	333040300	1341835248	6306695	0	3583	3583	10	326744355

FIGURE E.4 – Extrait d'un fichier de données reçues par RS-232

E.3 Temps de propagation

La moyenne du temps de propagation et l'écart-type sont calculés sur 200 mesures. La source des mesures est un fichier du types de celui de la figure E.4.

Longueur du câble	HUB	Temps de propagation moyen	Écart-type
20 m	√	3637 ns	82 ns
	×	450 ns	4.3 ns
100 m	√	4159 ns	78 ns
	×	843 ns	6.5 ns

TABLE E.2 – Temps de propagation

ANNEXE F

Régulateur

F.1 Dimensionnement

Fonction de transfert du temps local t_l par rapport au temps éloigné t_r (cf. figure F.1).

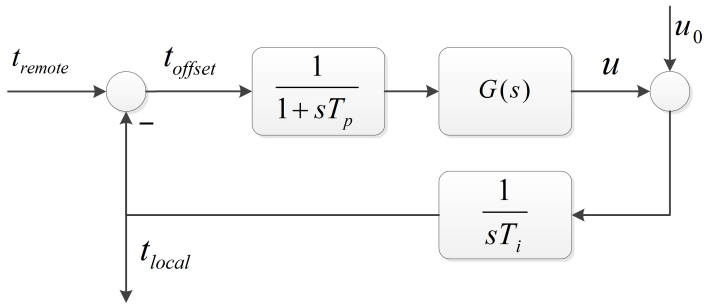


FIGURE F.1 – Régulateur

Période de mesure du décalage : $T_s = 2\,s$
Constante de temps de l'intégrateur de l'horloge locale : $T_i = 1\,s$

Fonction de transfert continue, sans tenir compte du maintien de l'échantillonnage : $\frac{t_l(s)}{t_r(s)} = \frac{G(s)}{sT_i+G(s)}$

Fonction de transfert (pseudo-) continue de t_r vers t_l en tenant compte de l'échantillonnage : $\frac{t_l(s)}{t_r(s)} = \frac{G(s)}{sT_i(1+sT_p)+G(s)}$

Fonction de transfert de l'oscillateur local en roue libre vers le temps local : $\frac{t_l(s)}{u_n(s)} = \frac{1+sT_p}{sT_i(1+sT_p)+G(s)}$ (1)

$\Rightarrow G_s$ doit inclure un intégrateur pour compenser u_n constant

Fonction de transfert d'un régulateur PI : $G_s(s) = K_p\frac{1+sT_n}{sT_n}$ (2)

(1)+(2) $\Rightarrow$

$$\frac{t_l(s)}{t_p(s)} = \frac{K_p(1+sT_n)}{sT_nT_i(1+sT_p)+K_p(1+sT_n)}$$

$$\frac{t_l(s)}{u_n(s)} = \frac{sT_n}{s^2T_nT_i(1+sT_p)+K_p(1+sT_n)}$$

Si le temps local est mis à jour à chaque coup de l'horloge locale : $T_s = 2\,s,\ u = \frac{1}{f_{clk}} = \frac{1}{66\,[MHz]}$

Petite constante de temps équivalente, égale à la moitié de la période d'échantillonnage T_s : $T_p = 1\,s$

En introduisant le régulateur PI dans la fonction de transfert en boucle fermée, on remarque qu'elle devient d'ordre 3, donc elle possède 3 pôles : $\frac{t_l(s)}{t_p(s)} = \frac{1+sT_n}{1+sT_n+\frac{s^2T_nT_i}{K_p}+\frac{s^3T_nT_iT_p}{K_p}}$

il y a 3 pôles : $p_0 = -\omega_0,\ p_{1,2} = -\omega_0(\alpha \pm j\beta)$

Il s'agit de placer ces 3 pôles de manière à obtenir un comportement stable et bien amorti.

$$\left(1-\frac{s}{p_0}\right)\left(1-\frac{s}{p_1}\right)\left(1-\frac{s}{p_2}\right) = 1 - s\left(\frac{1}{p_0}+\frac{1}{p_1}+\frac{1}{p_2}\right) + s^2\left(\frac{1}{p_0p_1}+\frac{1}{p_1p_2}+\frac{1}{p_2p_0}\right) - s^3\left(\frac{1}{p_0p_1p_2}\right) =$$

$$= 1 + \frac{s}{\omega_0}\left(1+\frac{2\alpha}{\alpha^2+\beta^2}\right) + \frac{s^2}{\omega_0^2}\ \frac{1+2\alpha}{\alpha^2+\beta^2} + \frac{s^3}{\omega_0^3}\ \frac{1}{\alpha^2+\beta^2}$$

Cas simple : 3 pôles réels identiques $\alpha = 1,\ \beta = 0$: $1 + 3\frac{s}{\omega_0} + 3\frac{s^2}{\omega_0^2} + \frac{s^3}{\omega_0^3} = \left(1+\frac{s}{\omega_0}\right)^3$

Identification des coefficients : $T_n = \frac{1}{\omega_0}\left(1+\frac{2\alpha}{\alpha^2+\beta^2}\right)$

Il y deux coefficients du régulateur PI (K_p et T_n) à dimensionner. Comme nous avons 3 contraintes de par l'identification des coefficients des polynômes au dénominateur, il reste une condition inhérente qui ne peut pas être choisie librement.

$$\left.\begin{array}{l} \frac{T_nT_i}{K_p} = \frac{1}{\omega_0^2}\ \frac{1+2\alpha}{\alpha^2+\beta^2} \\ \frac{T_nT_iT_p}{K_p} = \frac{1}{\omega_0^3}\ \frac{1}{\alpha^2+\beta^2} \end{array}\right\} T_p = \frac{1}{\omega_0(1+2\alpha)}$$

$K_p = \omega_0 T_i \frac{\alpha^2+\beta^2+2\alpha}{1+2\alpha}$

3 pôles réels identiques : $\alpha = 1,\ \beta = 0$

Le choix de 3 pôles réels identiques assure un bon amortissement avec une réponse indicielle sans dépassement

$\omega_0 = \frac{1}{3T_p} = \frac{2}{3T_s},\ K_p = \frac{2T_i}{3T_s},\ T_n = \frac{9}{2}T_s,\ \frac{K_p}{T_n} = \frac{4T_i}{27T_s^2}$

incrément de l'horloge locale, nominal : $\Delta T_0 = h = u_0$

incrément actuel de l'horloge locale : $\Delta T = \Delta T_0 + \Delta t$

sortie du régulateur G(s) : $\Delta t = u$

La sortie u du régulateur correspond à un incrément supplémentaire de l'horloge, qu'il faut répartir de manière uniforme sur une période T_s.

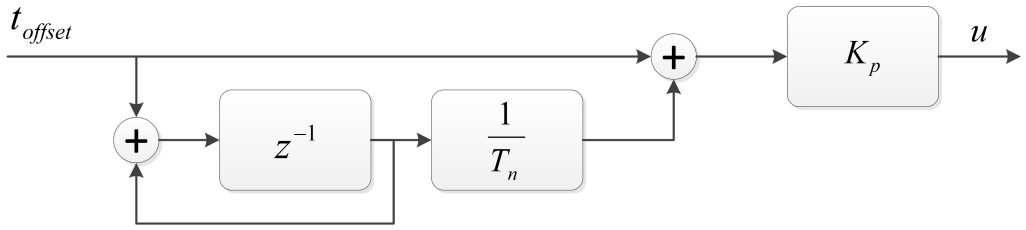


FIGURE F.2 – Implémentation du régulateur

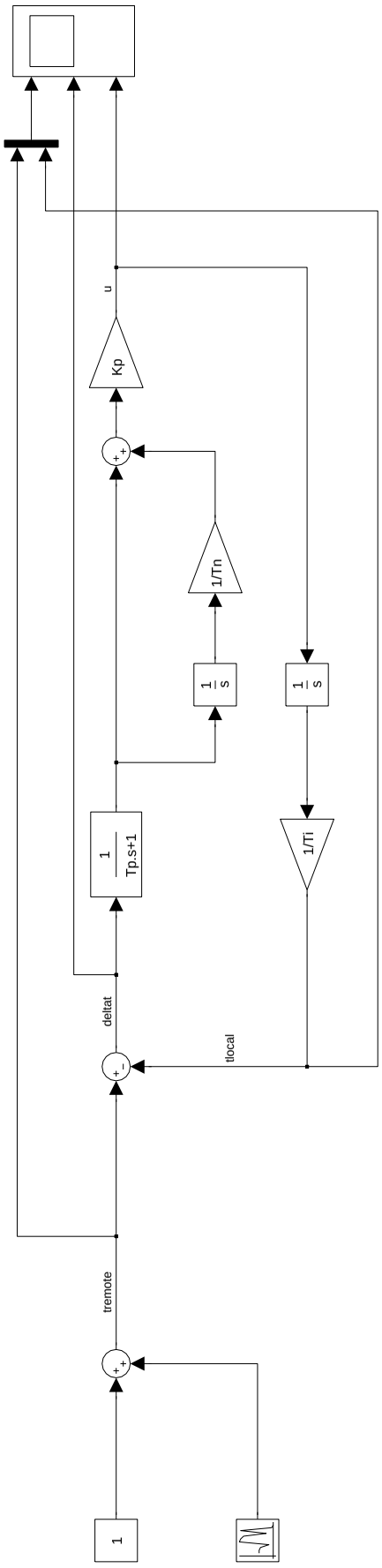
F.2 Simulations

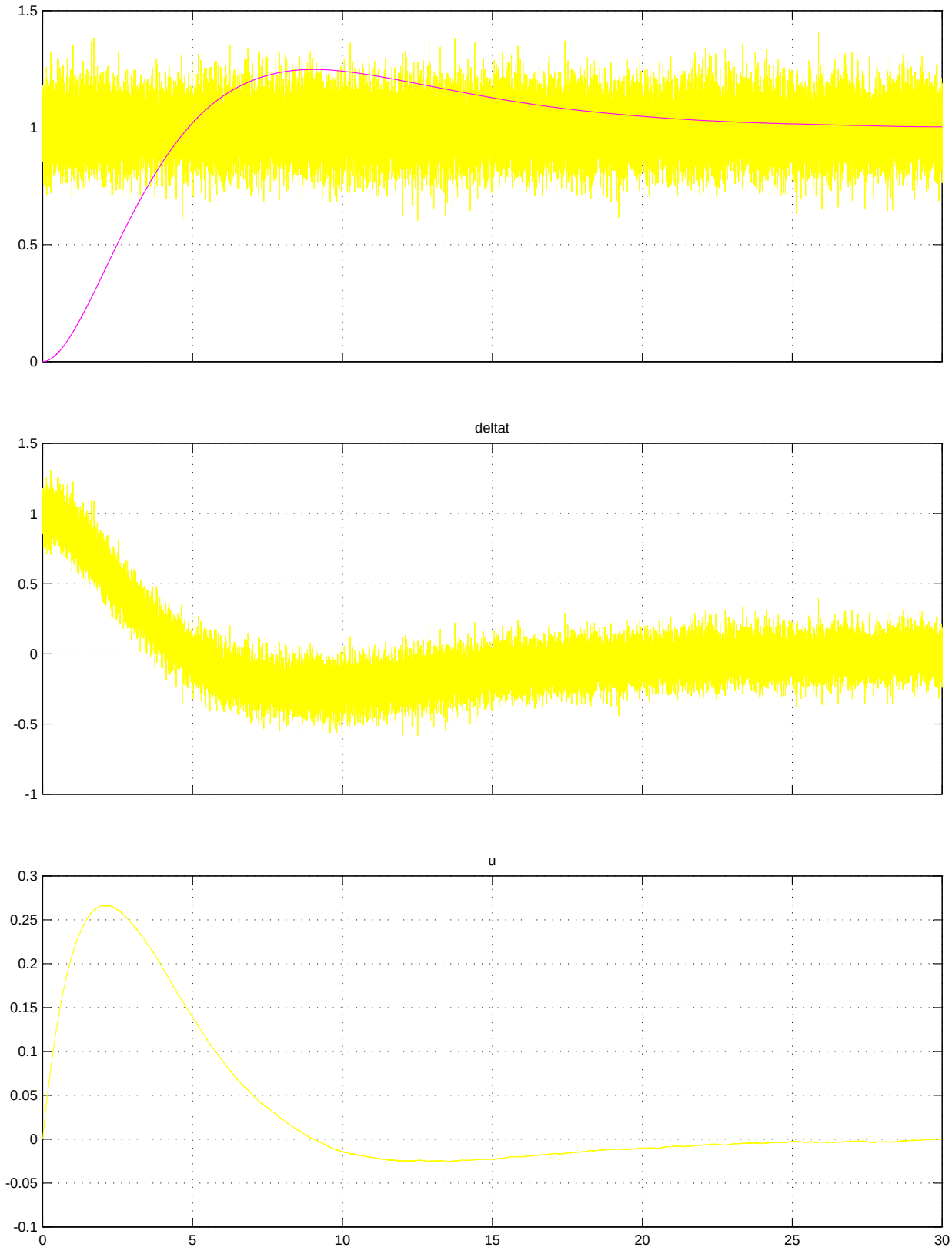
L'annexe page 82 représente la boucle continue, c.-à-d. sans tenir compte de l'échantillonnage. La simulation est donnée en page 83. On constate que le Δ_t tend vers 0.

L'annexe page 84 représente la boucle digitale (l'échantillonnage est pris en compte). Seulement la partie fractionnelle de l'offset est prise en compte. La simulation est donnée en page 85.

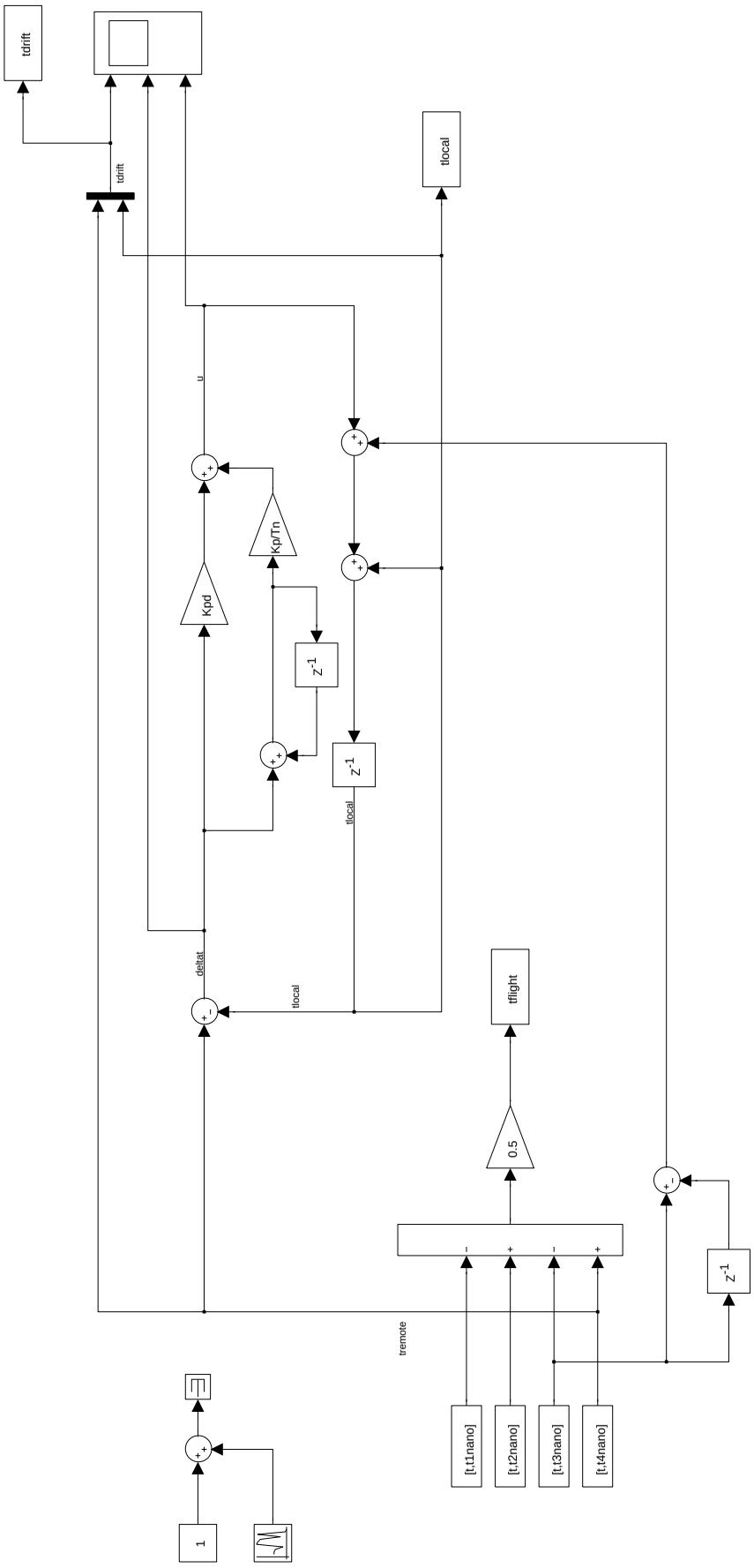
L'annexe page 86 représente la boucle digitale. Mais cette fois la mise à jour de l'horloge locale à la fréquence du quartz est simulée. Pour réduire le temps de simulation, la fréquence du clock est fixée à 66 [kHz] à la place de 66 [MHz]. La simulation est donnée en page 87. Elle semble fonctionner, cependant :

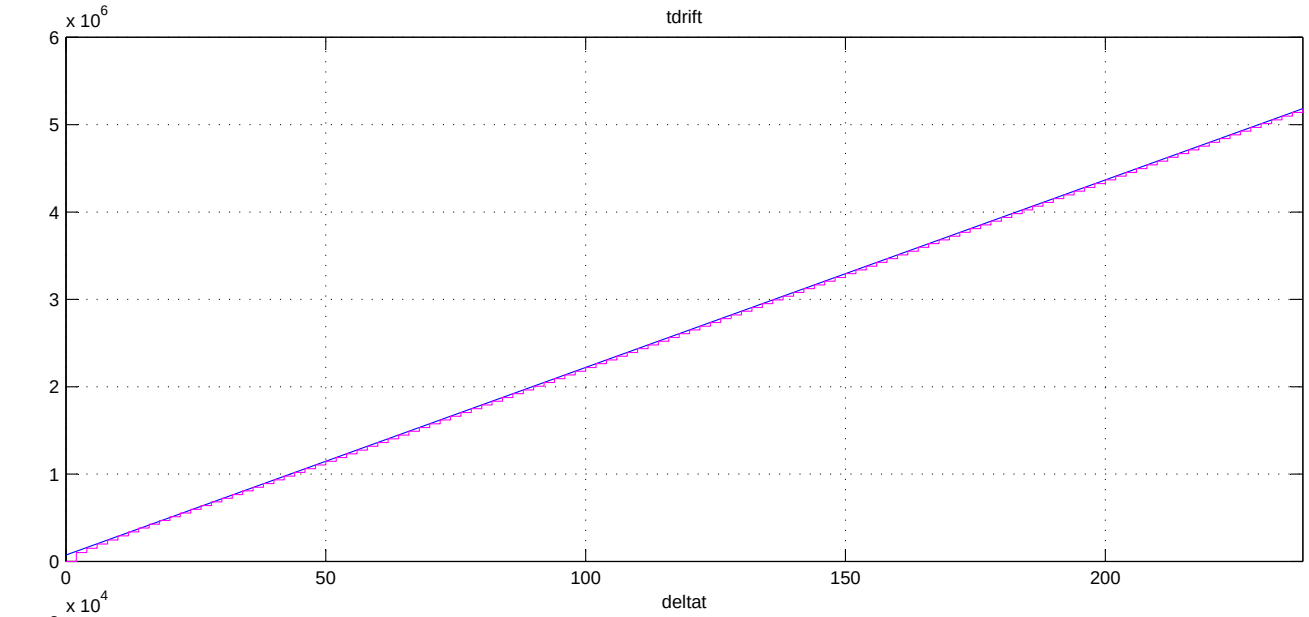
- Il va falloir revoir le placement des pôles du régulateur pour accélérer la convergence.
- Les parties *sec* et *nsec* du temps sont simulées. Tous les calculs sont en *nsec*. Les formats arithmétiques à virgule fixe.
- Un instant de temps supplémentaire t_5 doit être défini. C'est le temps à partir duquel la correction calculée par le régulateur est mise en œuvre.

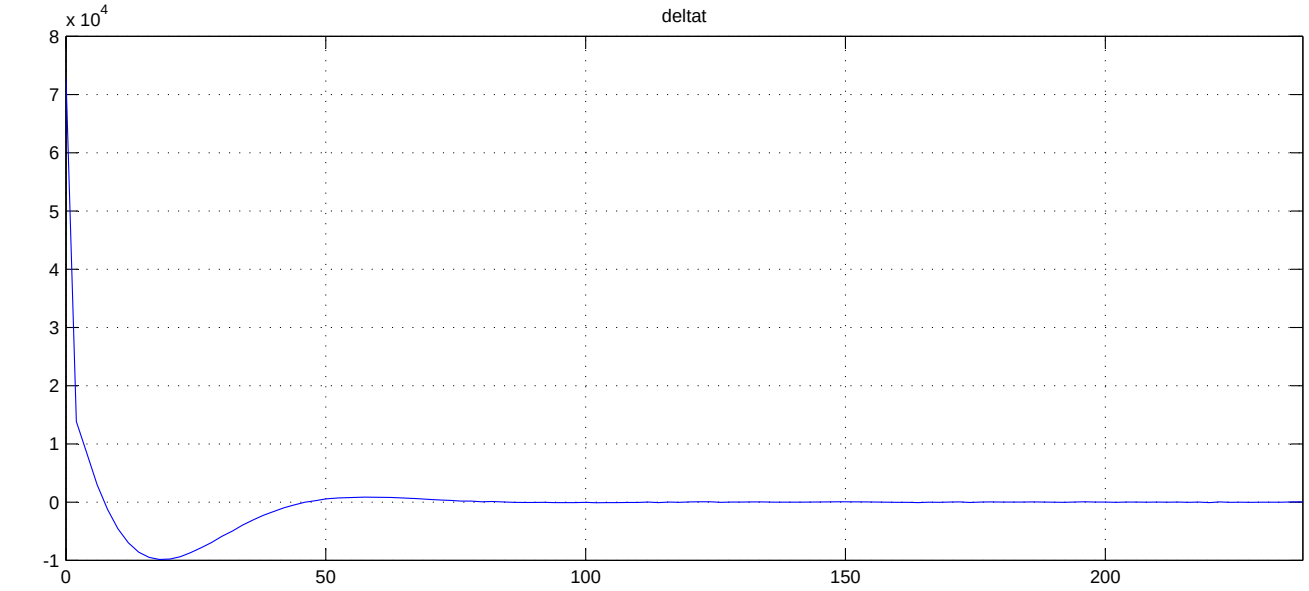

u
Kp
1/Tn
1/Ti
1
s
1
Tp.S+1
deltat
tlocal
tremote
1

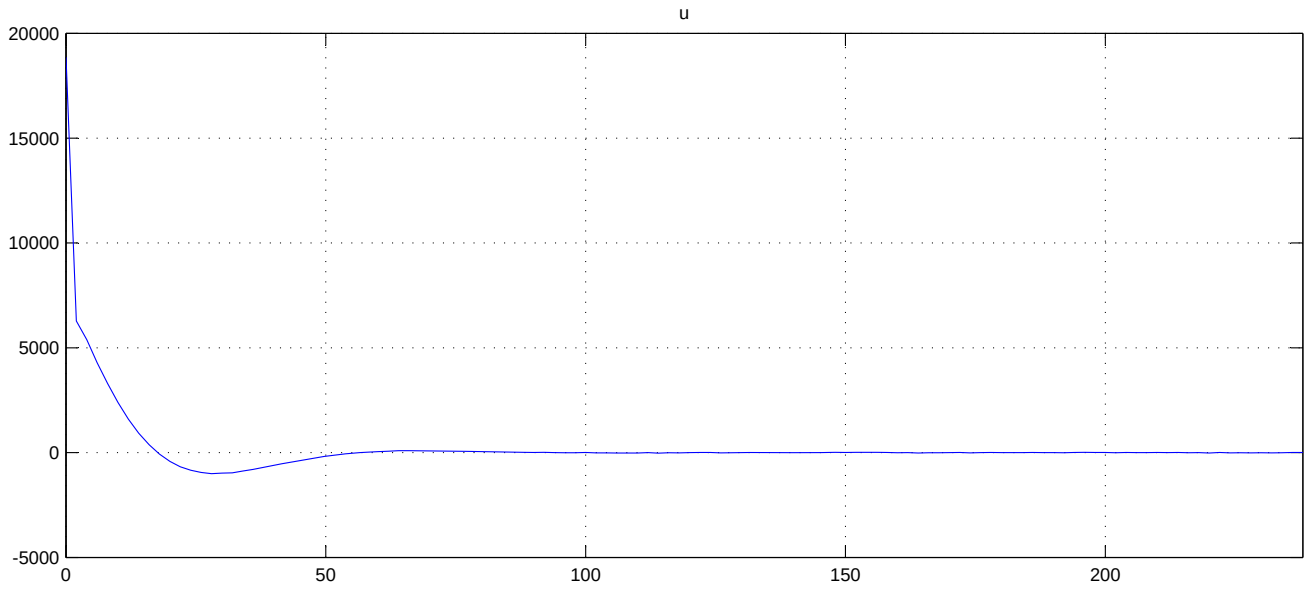


Time offset: 0

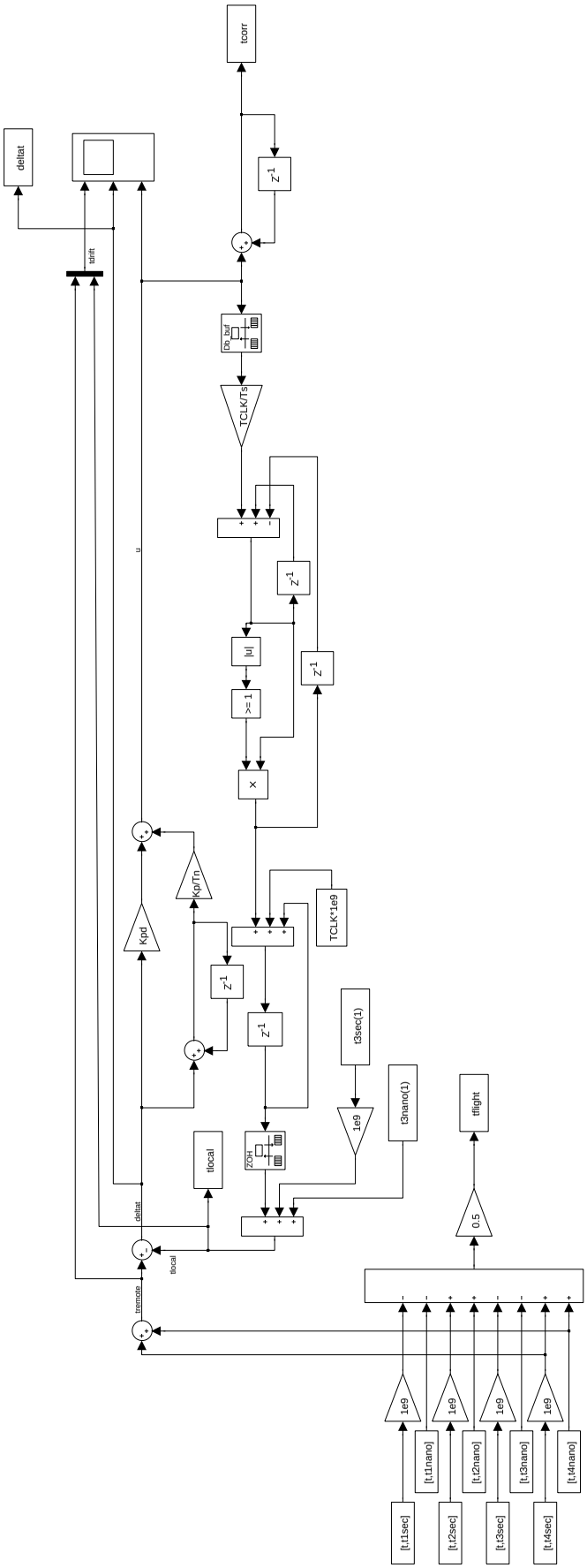

tdrift
tdrift
tlocal
u
Kp/Tn
Kpd
z-1
z-1
tlocal
deltat
tlocal
tflight
0.5
tremote
[t,t1nano]
[t,t2nano]
[t,t3nano]
[t,t4nano]
1

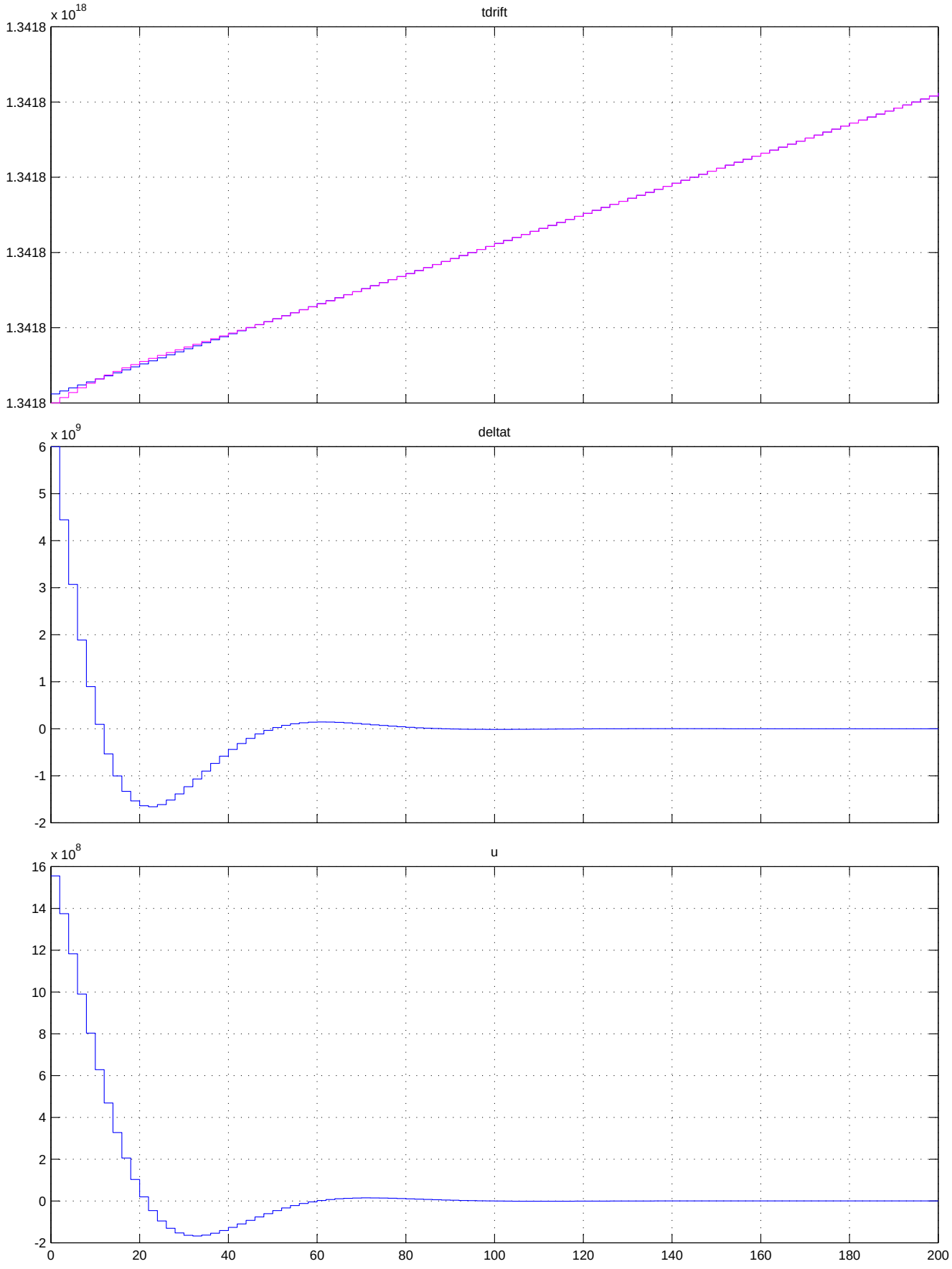






Time offset: 0


tcorr
deltat
Z-1
ddrift
Db_buf
TCLK/Ts
u
|u|
>=1
x
Kp/Tn
Kpd
TCLK*1e9
tlocal
ZOH
t3sec(1)
1e9
t3nano(1)
tflight
0.5
tremote
[t.t1sec]
[t.t2sec]
[t.t3sec]
[t.t4sec]
[t.t1nano]
[t.t2nano]
[t.t3nano]
[t.t4nano]



Time offset: 0

Table des figures

Liste des tableaux

Liste des programmes