Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

Diplom 2015

Samuel Stucky

Synchronisation von Steuerungen via Ethernet

Dozent/in
François Corthay

Experte/Expertin
Simon Delaley, Imperix SA

Datum der Abgabe des Schlussberichts
10.07.2015

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.BB
haj/31/03/2014

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2014/15**	No TD / *Nr. DA* **it/2015/26**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☒ Industrie **imperix SA** ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Samuel Stucky** Professeur / *Dozent* **François Corthay**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* *(données complètes)* **Simon Delaley** imperix SA | Route de Vissigen 25 | 1950 Sion	

Titre / *Titel*

Synchronisation von Seuerungen via Ethernet

Description / *Beschreibung*

Die Firma imperix erstellt elektronische Rack-Elemente, welche gemeinsam eine Power-Spannung erzeugen. Die Elementaren Spannungen werden durch PWM Signale erzeugt. Diese PWM Steuerungen müssen miteinander synchronisiert werden. Die Firma hat beschlossen, diese Synchronisation über Ethernet durchzuführen.

Die verschiedenen Elektronik-Karten werden ein gemeinsames Taktsignal bekommen, welches dem Ethernet-Clocksignal entspricht. Somit müssen die Karten sich alle auf derselben Periode synchronisieren. Die wird mit Hilfe vom Standard Precision Time Protocol (PTP) durchgeführt.

Objectifs / *Ziele*

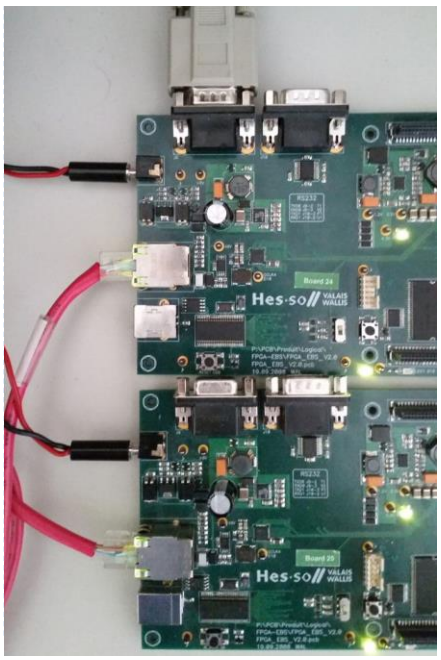
— Die FPGA-Hardware Architektur einer existierenden PTP Diplomarbeit zu analysieren und die Schaltung auf dieser Anwendung anzupassen

— Den Regler Teil durch eine serielle Schnittstelle auf einen PC zu verschieben und diesen mit einem Python-Code durchzuführen.

Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation *Leiter der Vertiefungsrichtung:*..... [1] Etudiant / Student :	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 11.05.2015 Remise du rapport / *Abgabe des Schlussberichts:* 10.07.2015, 12:00 Expositions / *Ausstellungen der Diplomarbeiten:* 26 – 28.08.2015 Défense orale / *Mündliche Verfechtung:* Semaine | Woche 36

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

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



Synchronisation von Steuerungen via Ethernet

Diplomand/in Samuel Stucky

Diplomarbeit
| 2015 |

Studiengang
Systemtechnik

Anwendungsbereich
Infotronics

Verantwortliche/r Dozent/in
François Corthay
francois.corthay@hevs.ch

Partner
Imperix SA

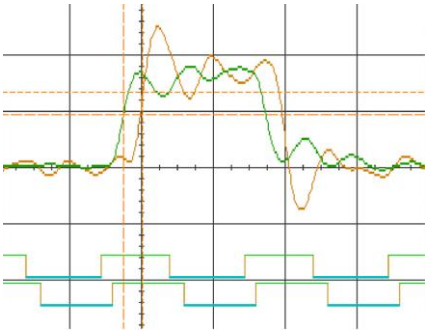
Ziel des Projekts

Ziel dieser Diplomarbeit ist es für die Firma Imperix eine Synchronisation zwischen mehreren Systemen über ein Ethernet Netzwerk zu ermöglichen, wobei das Precision Time Protokoll sowie der Sync-E Standard verwendet werden sollen.

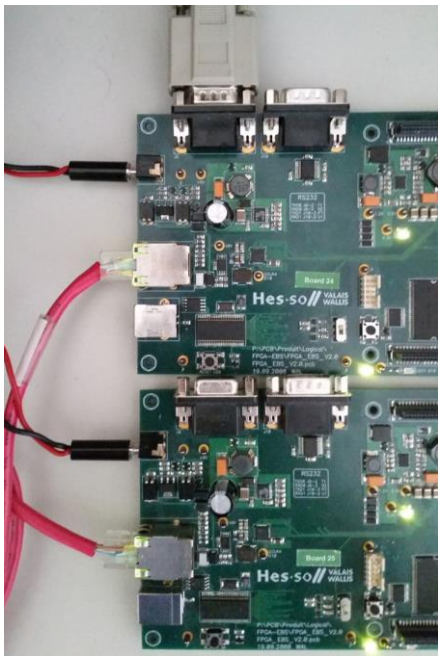
Methoden | Experimente | Resultate

Bei der Verbindung zweier Systeme via Ethernet wird vom Sender ein Clocksignal genutzt um Daten zu übermitteln. Dieses Signal kann auf der Empfängerseite rekonstruiert werden. Somit ist ein Signal mit gleicher Frequenz auf verschiedenen Systemen vorhanden und kann als Zeitgeber der Systeme dienen. Mittels des Precision Time Protokoll wird die zeitliche Verschiebung zwischen den beiden Systemen ermittelt und anschliessend korrigiert, sodass die Systeme miteinander synchronisiert werden. Die Berechnung des Korrekturwerts wurde vorerst mittels einer seriellen Verbindung auf einem Computer durchgeführt. Die Verwendung eines Computers ermöglichte zusätzlich das Aufzeichnen von Zeitdifferenzen, sodass die Genauigkeit und Stabilität des Systems besser analysiert werden konnten. In einem zweiten Schritt wurde die Berechnung lokal auf der elektronischen Karte ausgeführt.

Zur Analyse des Implementierten Designs wurden zwei elektronische Karten programmiert. Diese erstellen anhand ihrer synchronisierten Zeiten jeweils ein Pulssignal, welches einmal pro Sekunde auftritt. Die Zeitdifferenz der Systeme beträgt dabei nach der Synchronisation wenige Nanosekunden.



Messung der Zeitdifferenz zwischen einem Master und Slave System mittels eines Pulssignals.



Synchronisation des contrôleurs par Ethernet

Diplômant/e Samuel Stucky

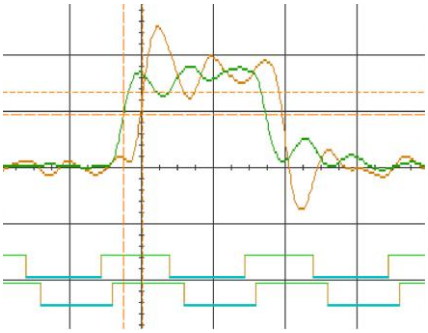
Objectif du projet

L'objectif de ce travail de diplôme est de permettre pour l'entreprise Imperix la synchronisation entre plusieurs systèmes via un réseau Ethernet en utilisant le protocole PTP et le standard Sync-E.

Méthodes | Expériences | Résultats

Pour envoyer des données d'un système à un autre par Ethernet, l'émetteur utilise un signal d'horloge. Ce signal peut être reconstruit du côté du récepteur. De cette manière, un signal de même fréquence est disponible sur les deux systèmes. Celui-ci peut servir comme intervalle de temps interne. Le décalage temporel entre les deux systèmes est déterminé au moyen du protocole PTP. Il est ensuite corrigé de manière à ce que les systèmes soient synchrones. Le calcul de la valeur de correction a été initialement réalisé au moyen d'une connexion série à un ordinateur. L'utilisation d'un ordinateur a en outre permis l'enregistrement des différences de temps, de sorte que la précision et la stabilité du système purent être mieux analysées. Dans une deuxième étape, le calcul a été effectué localement sur la carte électronique.

Pour analyser l'implémentation du protocole, deux cartes électroniques ont été programmées. Ces cartes créent chacune une impulsion qui se produit une fois par seconde en utilisant leur temps synchronisé. La différence de temps entre les systèmes après la synchronisation s'élève à quelques nanosecondes.



Une mesure de la différence de temps entre un système maître et un système esclave au moyen d'un signal d'impulsion.

Travail de diplôme
| édition 2015 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
François Corthay
francois.corthay@hevs.ch

Partenaire
Imperix SA

Inhaltsverzeichnis

Abkürzungen

PTP: **P**recision **T**ime **P**rotocol. Ein durch IEEE 1588 standardisiertes Protokoll, welches die Synchronisation über Ethernet von mehreren Systemen durch ein Master Taktgeber ermöglicht.

FPGA: **F**ield-**P**rogrammable **G**ate **A**rray. Ein integrierter Schaltkreis mit welchem eine logische Schaltung programmiert werden kann.

VHDL: **V**HSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage. Eine Hardwarebeschreibungssprache, welche es ermöglicht, digitale Systeme textbasiert zu beschreiben.

PWM: **P**ulse-**w**idth **m**odulation. Eine Modulationsart, bei welcher die Breite eines Signals mit bestimmter Frequenz eine technische Grösse übermittelt.

MII: **M**edia **I**ndependent **I**nterface. Eine Schnittstelle, welche im FastEthernet Bereich als Verbindung des MAC Layers mit dem PHY verwendet wird.

MIIM: **M**edia **I**ndependent **I**nterface **M**anagement. Ein serieller Bus, welcher zum Lesen und Schreiben der Control und Status Registern des MMI verwendet wird.

MLT3 : **M**ulti**l**evel **T**ransmission Encoding. In der Nachrichtentechnik verwendeter Leitungscode mit drei Spannungspegeln.

MAC: **M**edia **A**ccess **C**ontrol. Erweiterung des OSI-Models. Diese Schicht umfasst Netzwerkprotokolle und Regeln zur Nutzung des physikalischen Layers.

PHY : von **Phy**sical Layer, Schaltkreises zur Kodierung bzw. Dekodierung von Daten zwischen digitalen System

RAM: **R**andom-**A**ccess **M**emory. Ein Direktzugriffsspeicher, dessen Speicherzellen über eine eigene Adresse direkt angesprochen werden können.

FIFO: **F**irst **I**n **F**irst **O**ut (engl. für Wer zuerst kommt, mahlt zuerst). Speicherelement, bei welchem zuerst eintreffende Werte als erste wieder gelesen werden.

UDP: **U**ser **D**atagram **P**rotocol. Ein Netzwerkprotokoll, welches durch die Einführung von Ports das Zuweisen von gewissen Daten an bestimmte Anwendungen ermöglicht.

IP: **I**nternet **P**rotocol. Häufig verwendetes, verbindungsloses Netzwerkprotokoll, welches die Grundlage des Internets darstellt.

JTAG: **J**oint **T**est **A**ction **G**roup. Ein Verfahren um integrierte Schaltkreise zu Testen und Debuggen.

Einleitung

Die Firma Imperix entwickelt modulare Rack-Elemente welche eine Power-Spannung erzeugen. Diese Power-Spannung wird durch mehrere PWM-Signale erstellt. Die einzelnen Signale müssen dabei synchronisiert werden. Wie von der Firma vorgegeben, soll diese Synchronisation über Ethernet erfolgen.

Zur Realisierung dieser Synchronisation wird das Precision Time Protocol (PTP) genutzt. Es handelt sich um ein im IEEE 1588 Standard definiertes Protokoll, welches die Synchronisierung verschiedener Clocksignale durch ein Computernetzwerk ermöglicht. Eine Hardware Implementation dieses Protokolls erzielt eine Genauigkeit im Bereich von wenigen Nanosekunden.

Eine vorhandene Implementation dieses Projektes wird übernommen und abgeändert, um den Anforderungen dieser Problemstellung standzuhalten. Es handelt sich dabei um eine Diplomarbeit von Herrn Johan Droz (Droz, 2012).

Durch das Verwenden desselben Clocksignals auf mehreren Systemen wird die Genauigkeit der Synchronisation erhöht, sowie der Regelkreis deutlich vereinfacht.

Pflichtenheft

- Die FPGA-Hardware Architektur einer existierenden PTP Diplomarbeit zu analysieren und die Schaltung auf diese Anwendung anzupassen.

Die im Jahre 2012 erstellte Diplomarbeit von Herrn Johan Droz (Droz, 2012) beinhaltet eine Implementation des PTP Protokolls. Diese Implementation soll nun insoweit abgeändert werden, dass sie innerhalb eines bestehenden Designs der HES-SO // Valais – Wallis verwendet werden kann. Ausserdem soll teile der Implementation in den Ethernet Clock Bereich verschoben werden. Somit verwenden alle zu synchronisierenden Systeme dasselbe Ethernet Clock Signal, was eine Erleichterung der Synchronisation bedeutet.

- Den Regler Teil durch eine serielle Schnittstelle auf einen PC zu verschieben und diesen mit einem Python-Code durchzuführen.

Das Precision Time Protokoll ermöglicht die Berechnung eines Offsets der Zeit zwischen zwei Systemen. Um eine Korrektur der Zeit vorzunehmen wird ein Regler benötigt, welcher den Korrekturwert berechnet. Dieser Regler soll mittels eines Python-Codes auf einem Computer erstellt werden. Die Regelung mittels eines Python-Codes vereinfacht die Analyse des Reglers deutlich und ermöglicht das Aufzeichnen und Darstellen von erhaltenen Offset Werten.

Ressourcen

Software

Zur Programmierung der FPGAs wurde die Software HDL Designer von Mentor Graphics verwendet. Diese Software ermöglicht die Erstellung von FPGA Designs anhand von VHDL Code.

Zur Simulation der Schaltkreise wurde die Funktionseinheit ModelSim von Mentor Graphics verwendet, welche es ermöglicht, erstellte VHDL Designs zu testen.

Als Synthese und Analyse Tool wurde die Software ISE Project Navigator des Herstellers XILINX verwendet. Es ermöglicht sowohl die Kompilierung des Designs, wie auch die Konfigurierung der FPGA.

Die Entwicklungsumgebung PyCharm ermöglicht das Programmieren und Debuggen in Python. Die Bibliothek PySerial bietet dabei einen einfachen Zugriff auf serielle Schnittstellen.

Hardware

Zur Analyse der Ethernet Clocksignalen stand eine FPGA Karten zur Verfügung, welche innerhalb der HES-SO // Valais - Wallis erstellt und benutzt wird.

FPGA EBS V2.0 Karte

Die FPGA EBS Karte wird je nach Ausführung durch einen Xilinx XC3S500E oder XC3S1200E gesteuert. Ein Ethernet Anschluss ist vorhanden, wessen Physical-Layer mit einem ksz8041nl PHY des Herstellers Micrel realisiert wird. Ein JTAG Anschluss ist zur Programmierung der FPGA vorhanden.

Cisco SG100D-08 Switch

Der Gigabit Ethernet Switch von Cisco ist ein 8-Port Netzwerk Switch. Er ermöglicht das Verbinden von mehreren Ethernet fähigen Geräten und leitet deren Ethernet Pakete an des korrekte Empfängergerät weiter.

Precision Time Protocol

Das Precision Time Protocol (PTP) (Skoog, 2010) ist ein Netzwerkprotokoll, welches zur Synchronisierung von Clocksignalen verschiedener Systeme innerhalb eines Computernetzwerks benutzt wird. Dazu wurden spezielle Pakete zur Übertragung der Zeitinformationen im IEEE 1588 Standard festgelegt. In demselben Standard wird ebenfalls der genaue Ablauf der Synchronisation beschrieben.

Mit einer Hardware Implementation des Precision Time Protokolls wird eine Genauigkeit von einigen Nanosekunden erzielt. Die Nutzung eines Ethernet Netzwerks bietet dabei den entscheidenden Vorteil, dass in vielen Fällen keine zusätzliche Infrastruktur nötig ist, um die Synchronisation durchzuführen. Das vorhandene Netzwerk wird dabei nur geringfügig stärker belastet, so dass der bestehende Paketaustausch kaum beeinträchtigt wird.

Funktionsweise

Die Synchronisation der Clocksignale erfolgt jeweils zwischen einem Master und einem Slave. Dabei gilt das Clocksignal des Masters als Referenz. Das Clocksignal des Slaves soll nach Ablauf der Synchronisation demjenigen des Masters entsprechen.

One-Step

Die Synchronisation beginnt mit einer Sync Nachricht des Masters, welche den momentanen Zeitstempel des Masters enthält (Siehe Abbildung 1, τ_1). Zur Zeit des Empfangs dieser Nachricht speichert der Slave ein weiterer Zeitstempel (τ_2). Da die Nachricht jedoch eine gewisse Zeit benötigt um vom Master System zum Slave System zu gelangen ist es nicht ausreichend, den Wert τ_2 mit dem Wert τ_1 zu ersetzen. Die Nachrichtenlaufzeit wird durch das Senden der Nachricht Delay_Req vom Slave zum Master ermittelt. Hierbei wird sowohl ein Zeitstempel (τ_3) zur Zeit des Sendens als auch ein weiterer (τ_4) zur Zeit des Empfangens erstellt. Der Master antwortet dem Slave mit einem weiteren Paket (Delay_Resp), welches den Wert des letzten Zeitstempels (τ_4) enthält. Somit kennt dar Slave nun sowohl die Differenz der beiden Uhren als auch die Nachrichtenlaufzeit. Aus diesen Werten kann nun die Offset Zeit wie folgt berechnet werden:

$$Slave\ Time\ Offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}$$

Formel 1: Offsetberechnung

Diese Berechnung ist jedoch nur unter der Annahme gültig, dass die Zeit, welche die Nachricht benötigt um vom Master zum Slave zu gelangen, gleich gross ist wie die jene, welche für den Rückweg benötigt wird.

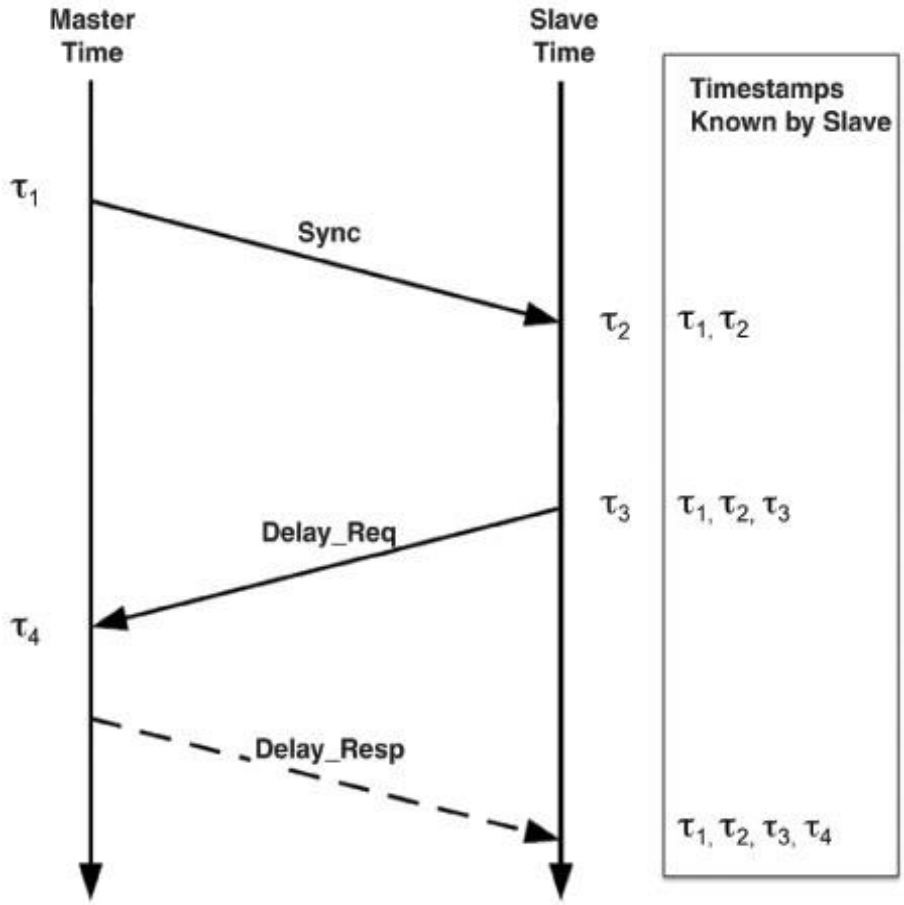


Abbildung 1: PTP One-Step

Two-Step

Im Gegensatz zur One-Step Implementation, wird bei der Two-Step Implementation eine zusätzliche Folow up (siehe Abbildung 2) Nachricht gesendet. Diese enthält den Zeitpunkt, zu welchem die Sync Nachricht gesendet wurde. Somit ist es nicht mehr nötig, diesen Timestamp direkt mit der dazugehörigen Nachricht mitzusenden. Dies erleichtert die Implementation des Protokolls deutlich, da nun jedes Paket im Vorfeld erstellt werden kann und nicht mehr während des Sendevorgangs angepasst werden muss, um den Timestamp einzufügen. Aus diesem Grund wurde im Bereich dieser Arbeit die Two-Step Implementation gewählt.

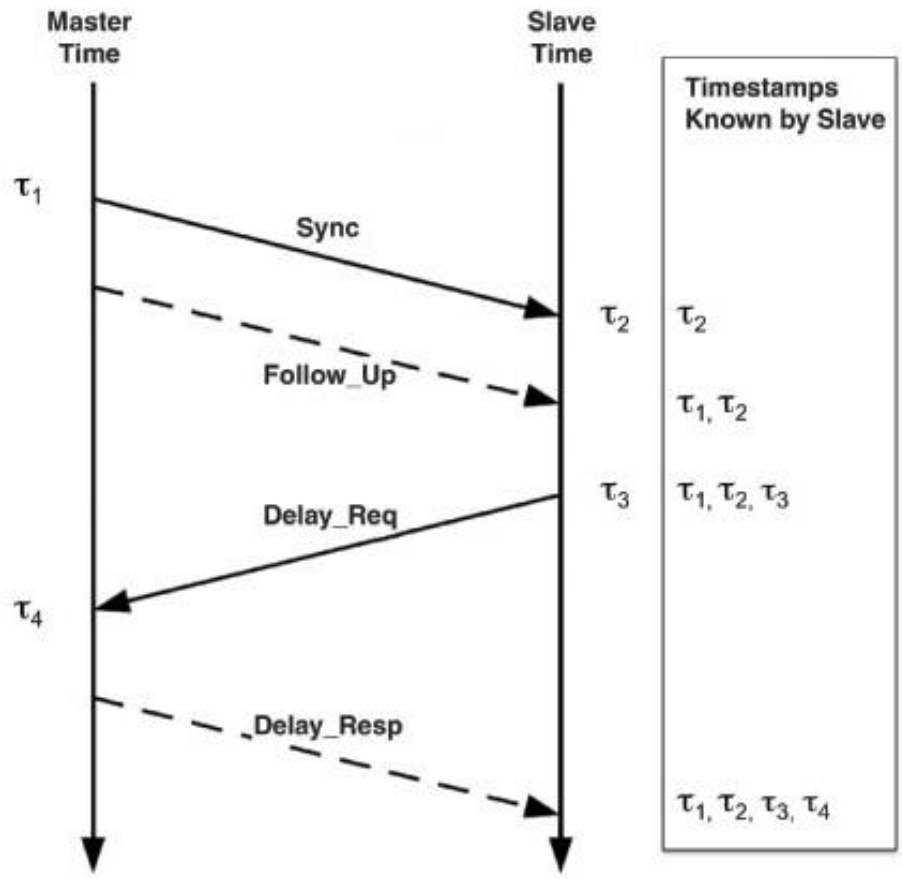


Abbildung 2: PTP Two-Step

PTP Frame

PTP Pakete können sowohl innerhalb eines Ethernet Frames wie auch in einem UDP Frame übertragen werden. Dazu wurden im PTP Standard IEEE1588 der Ethertype 0x88F7 sowie die UDP Portnummern 319 und 320 festgelegt. Innerhalb dieser Arbeit werden die PTP Pakete mittels des User Datagram Protokoll (UDP) übertragen. Ein Aufbau entsprechend dem OSI Referenzmodell eines PTP Frames ist in Abbildung 3 ersichtlich.

Wie in Abbildung 3 ersichtlich ist, bestehen die UDP Daten aus einem PTP Header und den PTP Daten. Die Struktur des Headers bzw. der Daten unterscheiden sich je nach PTP Nachricht, welche gesendet werden soll. Diese Struktur wurde im Standard IEEE1588 festgelegt.

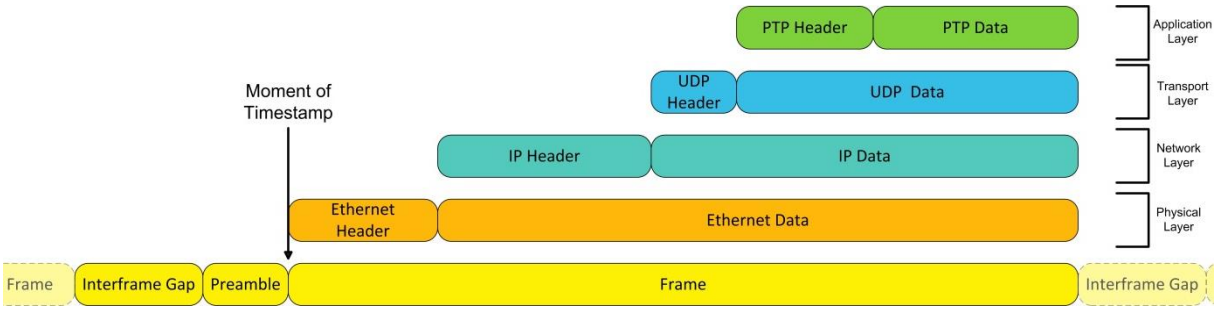


Abbildung 3: PTP Frame

Der Header des PTP Frames beinhaltet allgemeine Informationen des Frames. Von besonderer Bedeutung ist hier der message Type, welcher angibt um welche Nachrichtentyp es sich handelt, die PTP Version (in Bereich dieser Arbeit V2.0) und die sequence Id, welche die zusammengehörigen Nachrichten eines Synchronisationszyklus identifiziert.

Bits								Bytes	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

Tabelle 1: PTP Header

Die Daten der PTP Frames enthalten die Timestamps welche zum berechnen des Offsets benötigt werden. In den untenstehenden Tabellen sind die einzelnen Datenfelder der Frames ersichtlich.

Sowohl die Sync wie auch die Delay_Req enthalten einen originTimestamp (siehe Tabelle 2). Dieser wird nur beim Verwenden einer Transparent Clock (siehe Kapitel Synchronisation mehrerer Systeme) genutzt, nicht jedoch zur Berechnung des Offsets.

Bits								Bytes	Offset
7	6	5	4	3	2	1	0		
PTP Header (cf. Tabelle 1)								34	0
originTimestamp Seconds								6	34
originTimestamp Nanoseconds								4	40

Tabelle 2: Sync / Delay_Req Nachricht

Die Follow_Up Nachricht enthält einen präzisen Timestamp (siehe Tabelle 3) des Zeitpunkts, an dem die Sync Nachricht gesendet wurde.

Bits								Bytes	Offset
7	6	5	4	3	2	1	0		
PTP Header (cf. Tabelle 1)								34	0
preciseOriginTimestamp Seconds								6	34
preciseOriginTimestamp Nanoseconds								4	40

Tabelle 3: Follow_Up Nachricht

Delay_Resp Nachrichten enthalten zusätzlich zum Timestamp ein zusätzliches Feld requestingPortIdentity (siehe Tabelle 4), welcher innerhalb dieser Arbeit nicht benutzt und somit standartmässig auf 0 gesetzt wird.

Bits								Bytes	Offset
7	6	5	4	3	2	1	0		
PTP Header (cf. Tabelle 1)								34	0
receiveTimestamp Seconds								6	34
receiveTimestamp Nanoseconds								4	40
requestingPortIdentity								10	44

Tabelle 4: Delay_Resp Nachricht

PTP Timestamp

Der Timestamp wird benutzt um den genauen Zeitpunkt des Sendens oder Empfangens einer Nachricht abzuspeichern. Der genaue Zeitpunkt, zu welchem der Timestamp erfasst wird ist in Abbildung 3 ersichtlich.

Der Timestamp besteht aus zwei Werten. Der erste Wert ist 48bit gross und enthält die Anzahl Sekunden, der zweite ist 32bit gross und enthält die Anzahl Nanosekunden. Der Wert der Nanosekunden muss dabei immer zwischen 0 und 10^9 bleiben.

Synchronisation mehrerer Systeme

Sollen mehrere Systeme synchronisiert werden, muss ein PTP Switch verwendet werden. Hier stehen zwei Möglichkeiten zur Verfügung: eine Boundary Clock oder eine Transparent Clock. Die Verwendung eines Standard Switches ist nicht möglich, da die Zeit während der sich ein PTP Frame innerhalb des Switches befindet nicht konstant ist und somit die Synchronisation beeinflusst.

Boundary Clock

Die Boundary Clock verfügt über einen Slave Port welcher durch das Precision Time Protokoll die Zeit des Switches mit dem Grandmaster Clock synchronisiert. Die weiteren Ports dienen als Master für mehrere Slave Systeme. Der Switch erstellt dabei nach dem Durchführen der eigenen Synchronisation für jeden Master Port neue Sync sowie FollowUp Nachrichten welche zur Synchronisation der Slave Systeme dienen. Der gesamte Aufbau mit einer Boundary Clock ist in der untenstehenden Abbildung ersichtlich.

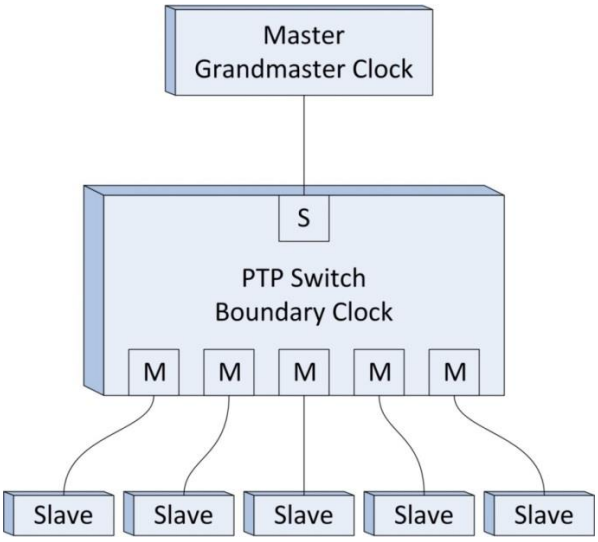


Abbildung 4: Aufbau mit Boundary Clock

Transparent Clock

Eine weitere Möglichkeit die Synchronisation auf mehrere Systeme zu verteilen bietet die Transparent Clock. Hierbei werden Korrektur Zeiten eingeführt welche bestimmen, wie lange sich ein PTP Frame innerhalb des Switches befindet. Dazu werden beim Empfangen, sowie beim Weiterleiten der Frames Timestamps erstellt. Während des Sendevorgangs wird der Timestamp des PTP Frames um den Korrektur Wert angepasst. Somit erscheint der Switch von aussen betrachtet als transparent.

Um ein korrektes Verhalten dieser Implementation mit einer Transparent Clock zu ermöglichen, muss jedoch die Master Applikation leicht angepasst werden, insofern sie ermöglichen muss, mehrere DelayReq Frames nacheinander zu bearbeiten. Dazu darf die Zustandsmaschine nach dem Behandeln der ersten DelayReq Nachricht nicht in einen idle Zustand wechseln sondern muss auf weitere mögliche Antworten warten, sodass sie mehrere Systeme gleichzeitig synchronisieren kann (siehe Kapitel Master Applikation).

Media Independent Interface

Beim Media Independent Interface (MII) (Wikipedia, Media Independent Interface, 2015) handelt es sich um eine Schnittstelle, welche vorwiegend im FastEthernet Bereich Verwendung findet. Sie dient dabei als Verbindung zwischen dem MAC Layer und dem PHY Layer. In Bereich dieser Arbeit verbindet er somit den PHY mit der FPGA.

Das MII besteht aus jeweils einem Tackt Signal und vier Datenlinien pro Richtung (siehe Abbildung 5). Die Datenlinien ändern sich dabei synchron zum Tackt Signal. Da die Übertragungsgeschwindigkeit bei FastEthernet 100Mbit/s beträgt, ist die Taktrate des Tackt Signals durch die vier Datenlinien vier Mal kleiner, also 25MHz.

Um die Daten vom MAC Layer zum PHY zu senden, werden die vier Datenbits synchron zum empfangenen Tackt Signal geschrieben. In Senderichtung ist ausserdem ein Enable Signal vorhanden, sowie in manchen Fällen ein Error Signal. Dabei wird das Enable Bit während des Sendens eines Pakets auf ‘1‘ gesetzt und auf ‘0‘ während des IDLE Zustands. Das Error Signal dient zur Indikation eines Fehlers, es wird jedoch in der verwendeten Implementation nicht genutzt.

In Empfangsrichtung sind ebenfalls ein Tackt- und vier Datensignale vorhanden. Hierbei werden die Daten bei der Ankunft des Tackt Signals gelesen. Ausserdem gibt es eine DataValid Signal zur Indikation, das die Daten gültig sind, sowie ein Error Signal beim Auftreten eines Fehlers. Beim Eintreffen einer Kollision wird dies durch die Linie Collision (COM) signalisiert. Das Signal Carier Sense ist nur im half-duplex Modus von Bedeutung, weshalb hier nicht weiter darauf eingegangen wird.

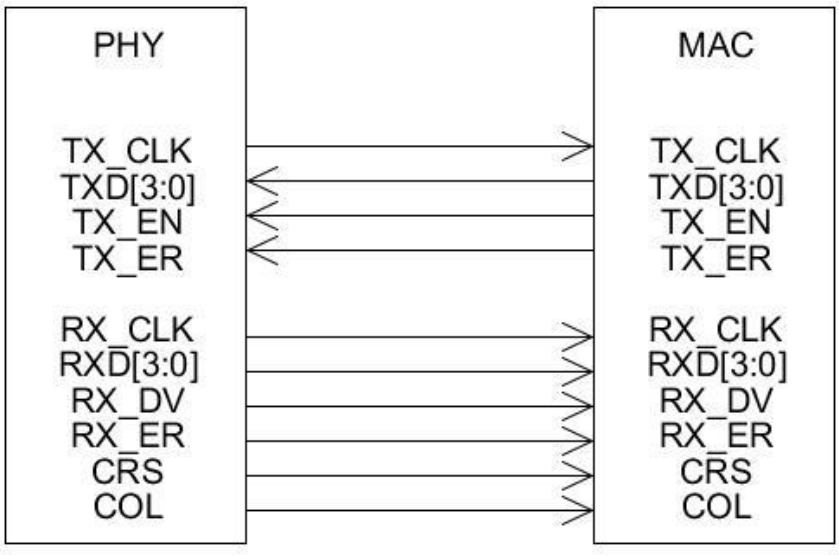


Abbildung 5 : Media Independent Interface

Media Independent Interface Management Bus

Um die Register des PHYs zu konfigurieren wird der vorhandene Media Independent Interface Management (MIIM) (Wikipedia, Management Data Input/Output, 2015) Bus verwendet. Es handelt sich hierbei um einen seriellen Bus, bestehend aus zwei Signalen: ein Tackt Signal, welches vom MAC Layer erstellt wird und zur Synchronisation der Datenbits dient, sowie ein bidirektionales Signal, welches die Daten überträgt. Das bidirektionale Signal wird dabei vom PHY ausschliesslich dann benutzt, wenn er dem MAC Layer auf eine read Register Anfrage antworten muss. Somit ist es möglich mehrere Slaves (PHYs) durch einen Master (MAC) zu kontrollieren. Auf der vorhandenen FPGA Karte mit zwei PHYs wurden jedoch zwei separate Linien realisiert, somit wurde die Option, mehrere PHYS gleichzeitig zu steuern, nicht verwendet.

Der Ablauf um einen bestimmten Wert in ein Register zu schreiben, wird in der untenstehenden Abbildung dargestellt. Der Beginn der Datenfolge wird mit einer “01“ Bitfolge signalisiert. Anschliessen wird mit einer zweiten “01“ Bitfolge der Schreibbefehl gesendet. Die folgenden Bit Werte enthalten die PHY Adresse sowie die Register Adresse, wessen Werte geändert werden sollen. Die Datenbits werden durch eine “10“ Bitfolge von den vorhergehenden Bitwerten getrennt.

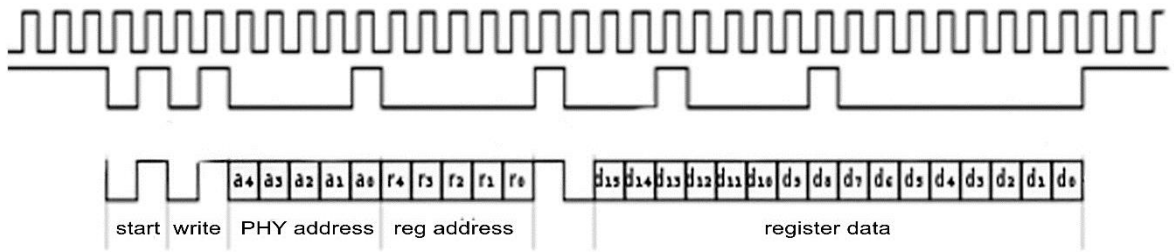


Abbildung 6 : Media Independent Interface Management Bus

Clock Bereiche

Analyse des Ethernet Taktsignals

Kodierung und Dekodierung

In diesem Kapitel wird die Funktion des PHYs ksz8041nl (Micrel, 2008) der Karte FPGA EBS analysiert. Hierbei liegt der Fokus auf der 100Base-TX Funktionalität des PHYs.

Um zu verstehen, wie das Ethernet Clocksignal von einem PHY zum andern übertragen wird, muss die Kodierung bzw. Dekodierung des Signals betrachtet werden. Der Ablauf ist in der untenstehenden Abbildung ersichtlich. Eine besondere Rolle spielen hier der 4B/5B sowie der MLT3 Encoder (siehe Markierung in Abbildung 7).

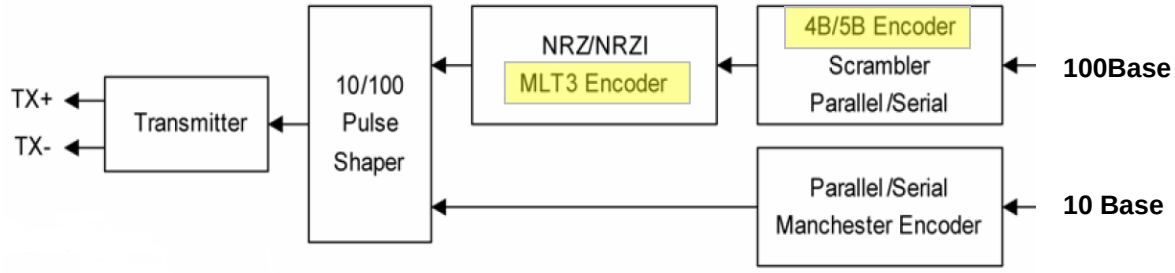


Abbildung 7 : Kodierung der Daten (Micrel, 2008)

4B/5B Encoder

Durch den 4B5B-Code (Wikipedia, 4B5B, 2013) werden vier Nutzdatenbits auf fünf Codebits abbildet. Dies ermöglicht die zusätzliche Übertragung von Verbindungszuständen wie beispielsweise Start, End, Idle, Quiet. Ausserdem werden die Datenbits so definiert, dass lange Folgen von gleichwertigen Bitwerten während der Übertragung von Daten vermieden werden können. Dies weist in Kombination mit dem MLT3 Encoder einen entscheidenden Vorteil auf.

MLT3 Encoder

Der MLT3 Encoder (Wikipedia, MLT-3 encoding, 2015) wandelt das digitale Signal in eine dreistufige Spannung um. Hierbei wird die Spannungsstufe jeweils beim Auftreten des Bit Werts ‘1‘ gewechselt. Beim Wert ‘0‘ bleibt die Spannung auf dem vorherigen Niveau. Der Ablauf ist in der untenstehenden Abbildung zu sehen.

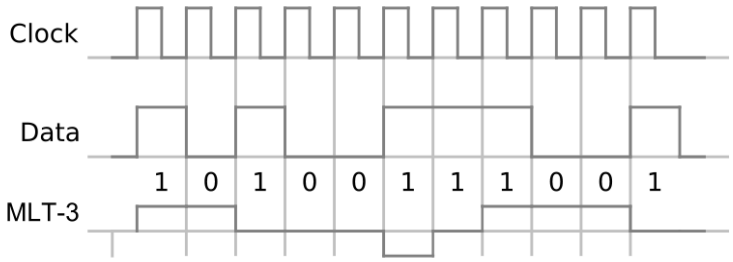


Abbildung 8 : MLT3 Encoder (Wikipedia, MLT-3 encoding, 2015)

Durch die Kombination dieser zwei Encoder wird ein ständiger Wechsel des Signalpegels gewährleistet. Dies zum einen durch die neue Kodierung der Datenbits im 4B5B-Code und zum andern durch die Einführung des Quiet Status welcher einer Folge von fünf '1' Bit Werten entspricht. Diese Signalfolge wird zwischen dem Senden von zwei Datenpaketen wiederholt gesendet und führt im MLT3 Encoder zu einem ständigen Wechsel der Ausgangsspannung. Dieser Ständige Wechsel bildet die Grundlage zur Rückgewinnung des Clocksignals anhand des Spannungspegels auf der Übertragungsleitung.

Tackt Rückgewinnung des Decoders

Um die Tackt Rückgewinnung anhand des Clocksignals zu illustrieren wurde beim Erhalt eines neuen Ethernet Pakets ein Trigger Signal erstellt. Dieses wird in den untenstehenden Abbildungen mit dem Label FRAME_25 gekennzeichnet.

Wie in der Abbildung 9 ersichtlich ist, wirkt sich die Ankunft eines neuen Ethernet Pakets nicht auf die Synchronisation des RX Clock Signals aus, da dieses zwischen den einzelnen Paketen anhand des Verbindungszustands Idle (ständiger Wechsel des Spannungspegels) rekonstruiert werden kann.

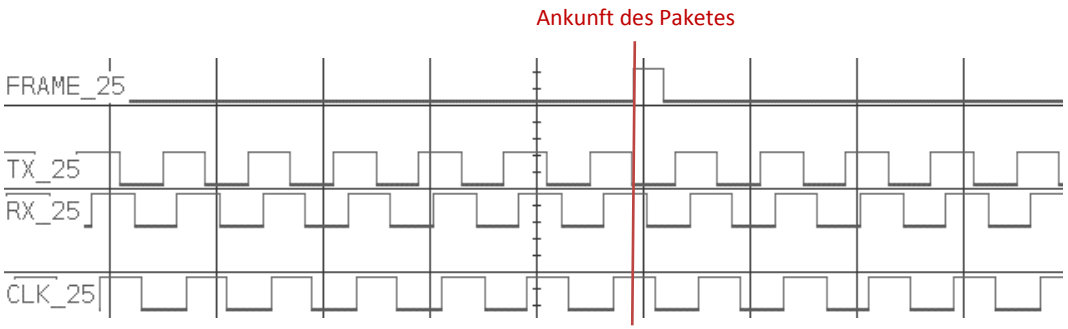


Abbildung 9 : Ankunft eines Ethernet Pakets

Eine Anpassung des RX Clock Signals kann jedoch beim Trennen der physikalischen Verbindung beobachtet werden, da es somit unmöglich wird, ein Tacktsignal zu rekonstruieren. In diesem Fall dient der eigene Taktgeber, welcher ebenfalls zur Erstellung des TX Clocks benötigt wird als Quelle der RX Clocks. Dieser Übergang ist in der Abbildung 10 ersichtlich.

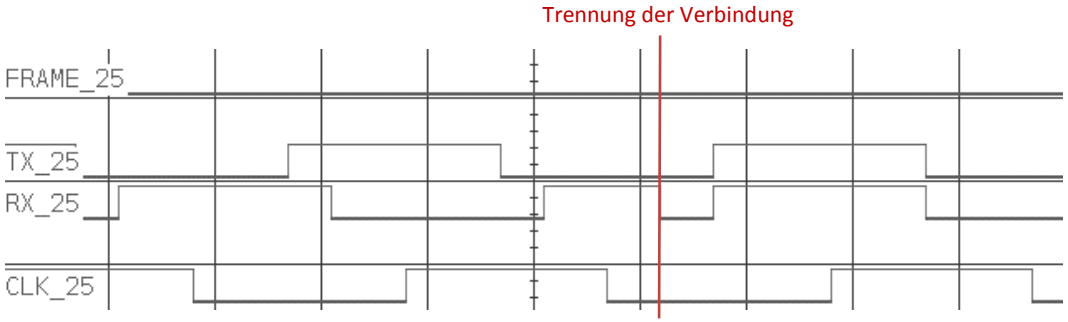


Abbildung 10 : Trennung des Ethernet Kabels

Somit kann festgehalten werden, dass eine Rekonstruktion des Signals beim Vorhandensein einer Verbindung durchgehend möglich ist. Dies ermöglicht die Verwendung das Ethernet Clock Signals als Referenz Clock innerhalb mehrerer Systemen.

Synchronisation zwischen Clock Bereichen

Die verschiedenen Clock Bereiche des Systems bestehen einerseits aus dem lokalen Clocksignal der FPGA Karte, andererseits aus den beiden Ethernet-Clocksignalen des MII Busses. Durch das Auftreten verschiedener Clocksignale innerhalb eines Designs können Probleme auftreten bei dem Übertragen von Signalen in einen anderen Clock Bereich.

Metastabilität

Um metastabile Zustände (Altera, 2009) eines Registers zu vermeiden, werden eine Setup-Zeit t_{SU} sowie die Hold-Zeit t_H vom Hersteller des Bausteins angegeben (siehe Abbildung 11: Clock Signal). Werden diese Zeiten eingehalten, nimmt der Ausgang des Registers spätestens nach der Clock-To-Output Zeit t_{CO} seinen neuen Wert an.

In synchronen Systemen kann diese Vorgabe ohne Probleme eingehalten werden, da die verschiedenen Signale alle durch denselben Clock erstellt werden und somit die Zeitvorgaben einhalten.

In asynchronen Systemen, kann diese Voraussetzung nicht eingehalten werden, da die Signale nicht von einem einzigen Clocksignal erstellt werden. Somit wird nicht sichergestellt, dass ein

Eingangssignal des Registers während t_{SU} und t_H stabil bleibt. Dies kann zu einem metastabilen Zustand führen (siehe Abbildung 11). Dabei nimmt der Ausgang des Registers vorerst einen Zwischenwert an, welcher sich zwischen einer logischen 0 und einer logischen 1 befindet. Ein stabiler Zustand wird erst nach Ablauf der vorgegebenen Clock-To-Output Zeit t_{CO} erreicht. In diesem Fall kann nicht bestimmt werden, ob dieser Zustand dem alten oder dem neuen Eingangswert entspricht.

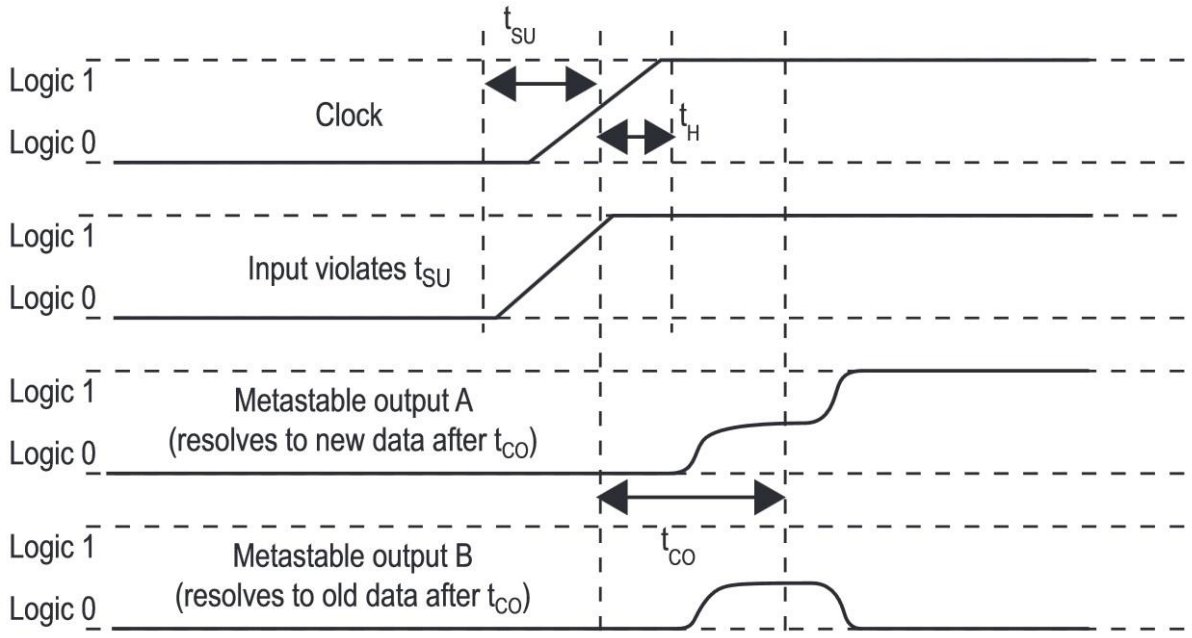


Abbildung 11: Metastabilität

Um solche metastabilen Zustände zu vermeiden müssen Signale beim Übertritt in einen neuen Clock Bereich synchronisiert werden. Im Bereich dieser Arbeit treten dabei hauptsächlich zwei Signaltypen auf, welche synchronisiert werden müssen.

Bus Signale

Bei der Synchronisation von Bus Signalen muss beachtet werden, welcher der beiden Clock Bereiche die höhere Frequenz aufweist. Innerhalb dieses Projekts treten nur Bus Signale auf, welche eine sehr niedrige Frequenz aufweisen und in einen Empfänger Clock Bereich synchronisiert werden, welcher eine höhere Frequenz aufweist. Dies erleichtert die Synchronisation deutlich, da es zur Synchronisation dieser Signale reicht, ein zusätzliches Register innerhalb des Empfänger Clock Bereichs einzufügen. Dies vermindert das Auftreten von Metastabilitäten deutlich, da die Signale somit erst nach Ablauf einer vollen Clockperiode gelesen werden und das Signal bis zu diesem Zeitpunkt wieder einem stabilen Zustand entspricht. Ebenfalls werden Änderungen der Daten des Bus Signals mittels eines Pulses indiziert. Somit werden die Werte des Busses immer erst einige Clockperioden nach der eigentlichen Änderung gelesen. Dies verhindert, dass Werte gelesen werden, welche durch Metastabilität nicht dem eigentlichen Wert des Busses entspricht.

Beim Synchronisieren von Signalen in einen Clock Bereich mit tieferer Frequenz wäre dies nicht ausreichend, da kurze Zustandsänderungen innerhalb des langsameren Clock Bereichs nicht detektiert werden können.

Logic Pulse

Bei der Synchronisation eines einzelnen Pulses treten ähnliche Probleme auf. Soll nämlich der Puls in einen Bereich mit einer höheren Frequenz synchronisiert werden, kann es vorkommen, dass dieser mehrfach erkannt wird, und somit im neuen Frequenzbereich mehreren nachfolgenden Pulsen entspricht (siehe Abbildung 12). Im Gegensatz hierzu wird ein Puls womöglich gar nicht erst detektiert, wenn er in einen tieferen Frequenzbereich übertragen wird (siehe Abbildung 13). Somit ist es nicht möglich die Synchronisation mit einem Flipflop durchzuführen.

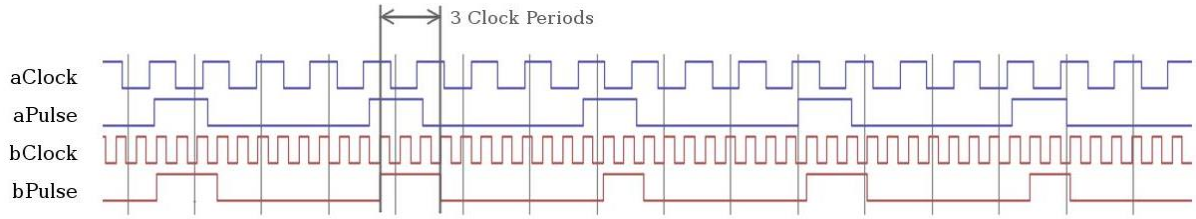


Abbildung 12: Synchronisation mit FlipFlop - Frequenz a < Frequenz b

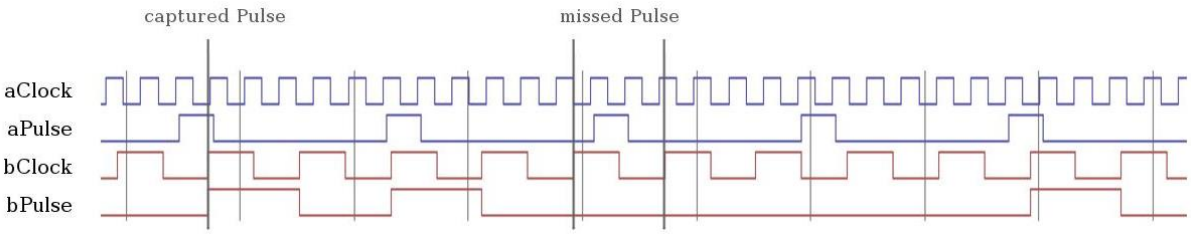


Abbildung 13: Synchronisation mit FlipFlop - Frequenz a > Frequenz b

Um dieses Problem zu eliminieren, wird anstelle der Pulssignale ein Signal über die Clockgrenze synchronisiert, welches beim Eintreffen eines Pulses seinen Zustand wechselt. Um sicherzustellen, dass dieses erzeugte Signal korrekt synchronisiert wird, dürfen Pulssignale höchstens mit einer halb so hohen Frequenz auftreten, wie die Clockfrequenz der Empfängerseite. Somit kann sichergestellt werden, dass das Signal mindestens während zwei Clockperioden vom Empfänger detektiert werden kann. Die Synchronisation dieses Signals kann mit einem einfachen Flipflop durchgeführt werden. Anschliessend müssen die einzelnen Pulse durch ein Detektieren der Flanken aus dem synchronisierten Signal wiederhergestellt werden (siehe Abbildung 14 und Abbildung 15).

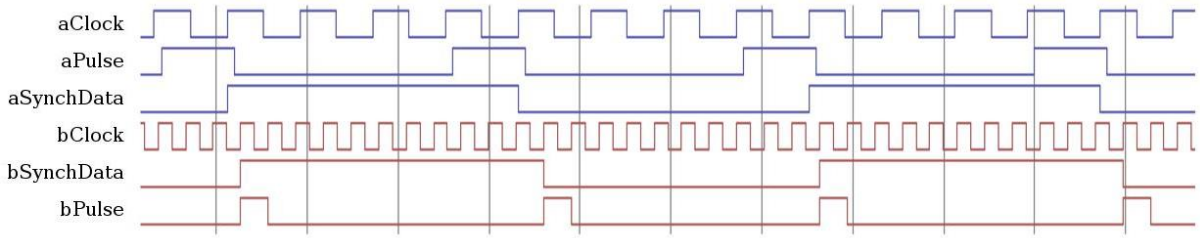


Abbildung 14: Synchronisation mit Zwischensignal - Frequenz a < Frequenz b

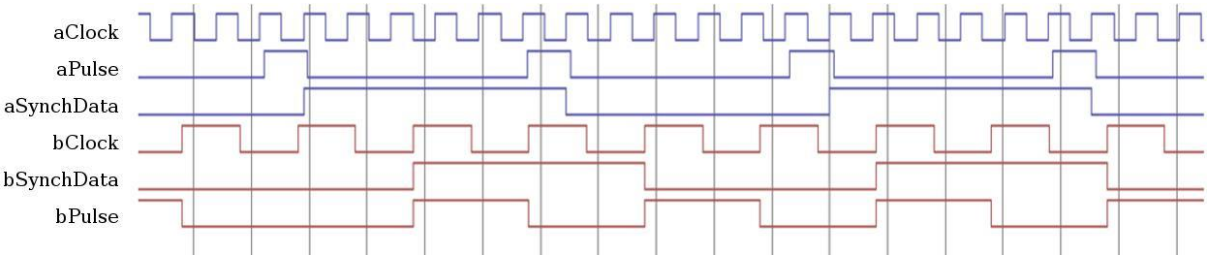


Abbildung 15: Synchronisation mit Zwischensignal - Frequenz a > Frequenz b

Die Synchronisation eines Pulses stellt im Zusammenhang mit dem Precision Time Protocol ein weiteres Problem dar, da die Dauer der Synchronisation nicht einen konstanten Wert aufweist.

In der untenstehenden Abbildung ist eine Simulation einer Synchronisation zwischen zwei Clocksignalen mit beinahe derselben Frequenz dargestellt. Diese Situation tritt beispielsweise auf, sobald zwei Clocksignale mit derselben Frequenz vorhanden sind, welche jedoch von unterschiedlichen Quarzen erstellt werden (z.B. Ethernet RX- und TX- Clocksignale). Je nach Phasenverschiebung der beiden Signale dauert die Synchronisation zwischen einer und zwei Clockperioden. Diese beiden Extremfälle treten auf, wenn die steigende Flanke des Empfängerclocks sehr kurz auf die des Senders folgt, beziehungsweise dieser nur sehr kurz vorausgeht.

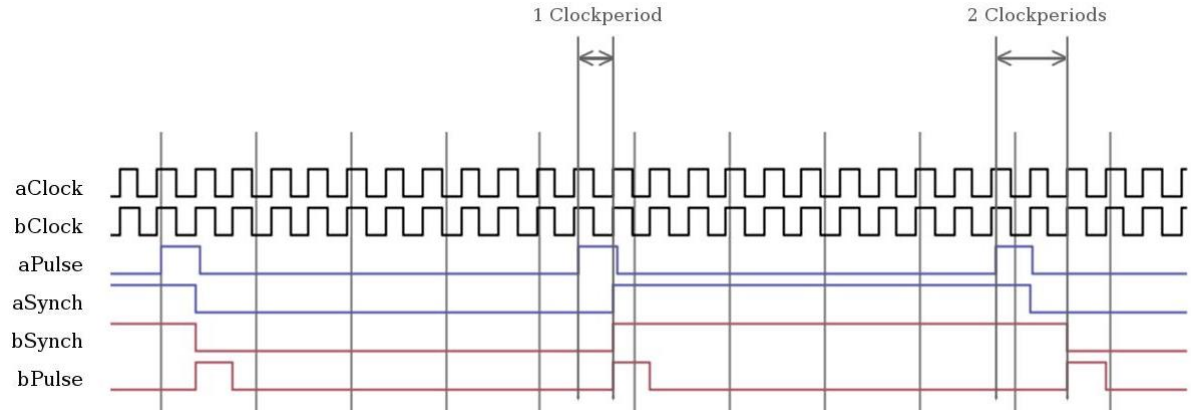


Abbildung 16: Dauer der Synchronisation

Dieser Effekt zeigt sich innerhalb dieses Projekts folgendermassen: Ein Puls zur Indikation des Starts eines PTP Frames, welcher im Rx- oder Tx-Clock Bereich erstellt wird und in einen anderen Clock Bereich synchronisiert werden muss, kann somit eine Verzögerung zwischen einer und zwei Clockperioden aufweisen. Dies entspricht im Bereich eines Fast Ethernet Netzwerks 40ns bis 80ns. Da sich der Wert des lokalen Zählers nur alle 40ns ändert, können somit Timeouts einen nicht konstanten Wert aufweisen, welcher sich um 40ns unterscheidet. Dies hat zur Folge, dass bei der Berechnung ein Fehler von 20ns Berechnet wird (siehe Formel 2), obwohl die PTP Clock Signale des Masters und des Slaves synchron verlaufen.

$$Slave\ Time\ Offset = \frac{((t_2 + 40ns)\ - t_1) - (t_4 - t_3)}{2} = 20ns$$
$$falls\ t_2 - t_1 = t_4 - t_3\ (synchrones\ System)$$

Formel 2: Berechnung mit einem falschen Timeout Wert

Dieses Verhalten muss beim Erstellen des Korrekturwertes in Betracht gezogen werden, sodass es nicht zu unnötigen Korrekturen führt.

Software Pipelining

Bei der Berechnung von sehr hohen Werten sind grosse logische Schaltungen nötig, welche eine Grosszahl an Gatter-Verzögerungen aufweisen. Wird durch das Aneinanderreihen von vielen kombinatorischen Schaltungen eine gesamte Gate-Verzögerung erreicht, welche grösser oder gleich einer Clockperiode ist, kann ein korrektes Verhalten der Schaltung nicht mehr garantiert werden. Die Berechneten Werte haben möglicherweise noch nicht ihren Endwert angenommen und es kann zu metastabilen Zuständen kommen.

Um dies zu verhindern wird bei solch grossen Berechnungen häufig das Software Pipelining angewandt. Hierbei werden die einzelnen Schaltungsabschnitte der Berechnung durch Register voneinander getrennt. Somit wird die Gate-Verzögerung der einzelnen Abschnitte deutlich verringert, wodurch die mögliche Clockfrequenz erhöht werden kann.

Zur Berechnung des PTP Offsets wird mit grossen Werten gerechnet. Um ein mögliches Fehlverhalten auszuschliessen wurde die Berechnung in mehrere Schritte geteilt und somit eine einfache Form des Software Pipelinings angewandt. Während der Clockperiode t_{clk} (siehe Abbildung 17) muss somit nicht die gesamte Berechnung ausgeführt werden, sondern nur Teile davon (siehe Abbildung 18).

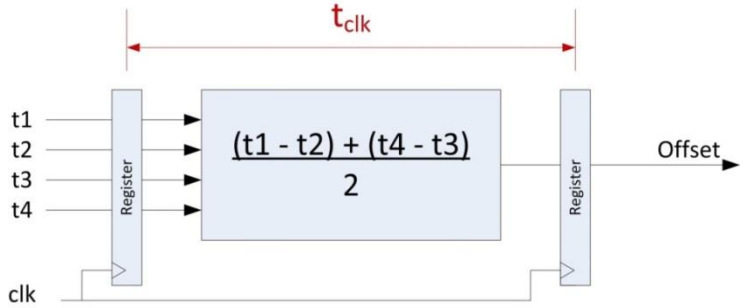


Abbildung 17: Berechnung ohne Pipeline

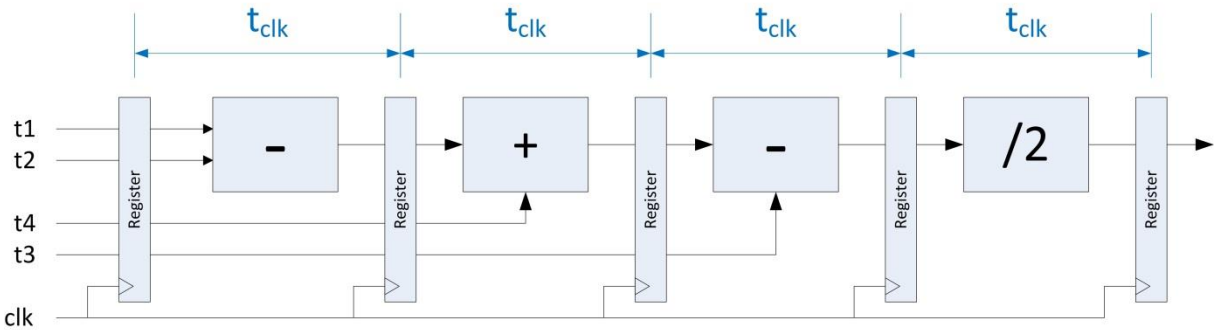


Abbildung 18: Berechnung mit Pipeline

Das Teilen der Berechnung kann innerhalb einer Zustandsmaschine durch das Einfügen von zusätzlichen Zuständen einfach realisiert werden, da der Zustandswechsel frühestens nach einer Clockperiode t_{clk} auftritt.

Implementation

Als Ausgangspunkt zur Implementierung des Precision Time Protokolls wurde eine Diplomarbeit von Herrn Johan Droz (Droz, 2012) verwendet und an die Anforderungen dieses Projektes angepasst.

Die Kommunikation über Ethernet wurde mittels eines bestehenden Designs der HES-SO // Valais - Wallis erstellt. Dieses Design ermöglicht das Senden und Empfangen von UDP Paketen.

Grundstruktur

Die PTP Implementation besteht aus mehreren Teilen (siehe Abbildung 19).

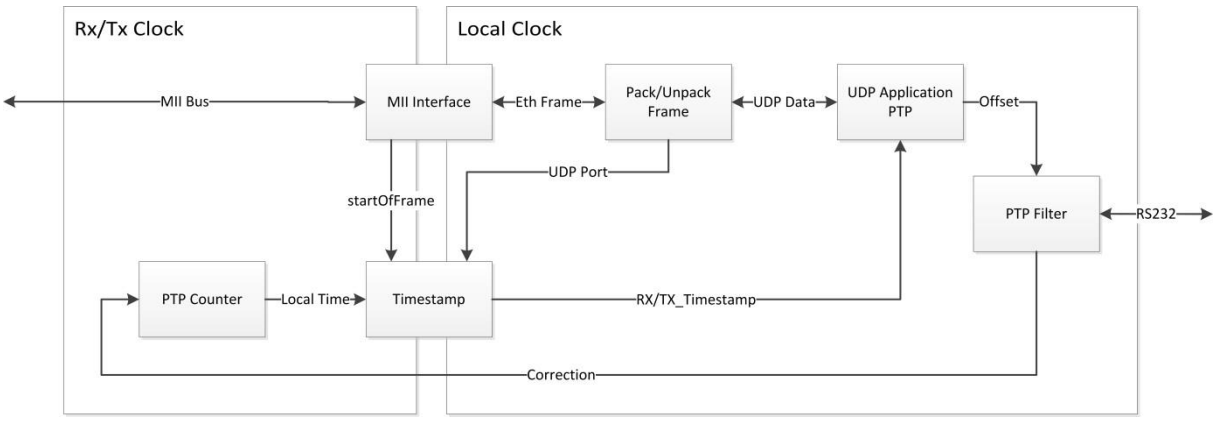


Abbildung 19: Aufbau PTP Implementation

Zur Kommunikation mittels des UDP Protokolls werden Blöcke benötigt, welche die UDP Frames mittels des Internet Protokolls (IP) in ein Ethernet Frame verschachteln. Ausserdem ist ein MII Interface nötig zur Kommunikation mit dem Physical Layer.

Die PTP UDP Applikation stellt den wichtigsten Teil der PTP Implementation dar, da hier die einzelnen PTP Frames erstellt werden und ankommende PTP Frames gelesen und analysiert werden. Ausserdem wird in diesem Teil das PTP Offset berechnet.

Der Wert dieses Offsets wird anschliessend an einen Filter gesendet, welcher über eine serielle Schnittstellt mit einem Computer kommuniziert. Hier wird ein Korrekturwert berechnet und anschliessend an den Filter zurückgesendet. Dieser leitet diesen Wert weiter an den PTP Zähler.

Der Zähler erstellt die lokale Zeit des Systems durch das Inkrementieren eines Werts um eine konstante Anzahl Nanosekunden pro Clock Puls. Um die Zeit an andere Systeme anpassen zu können

wird mittels eines Enabel Signals der Korrekturwert zur momentanen Zeit hinzugefügt. Diese resultierende Zeit wird anschliessend genutzt um die Timestamps zu erstellen. Sie dient ausserdem als Referenz zur Erstellung eines Clocksignals welches auf mehreren Slave Systemen synchron läuft.

Die Timestamps werden in einem weiteren Teil erstellt. Hier wird jeweils die momentane Zeit gespeichert, zu welcher ein Frame gesendet beziehungsweise empfangen wird.

Top Level

Die Hauptfunktion des Toplevels ist die Synchronisation verschiedener Eingangssignale auf den gewünschten Clock. Dies verhindert Metastabilitäten (siehe Kapitel Metastabilität) und ist nötig, da die Eingangssignale asynchron gegenüber dem Clocksignal verlaufen. So wird ein Reset Signal auf verschiedene Clock Bereiche synchronisiert, damit es innerhalb dieser ohne Fehlfunktion benutzt werden kann. Ausserdem werden die eingehenden RX MII Signale auf den RX Clock synchronisiert.

Das Design des Top Levels ist in **Anhang 1** ersichtlich.

Ethernet Kommunikation

Digitale Schaltung zur Konfiguration

Um den PHY mit den richtigen Werten zu konfigurieren wurde ein Block erstellt, welcher die gewünschten Werte der zu ändernden Register enthält. Diese werden anschliessend an ein FIFO weitergeleitet. Dabei werden die Registeradressen, die Datenwerte, sowie ein Read/ Write Bit übergeben. Mit einem zusätzlichen Bit Write wird dem FIFO mitgeteilt, dass die Daten stabil sind und gelesen werden sollen (siehe untenstehende Abbildung). Der erstellte VHDL Code ist in **Anhang 22** ersichtlich.

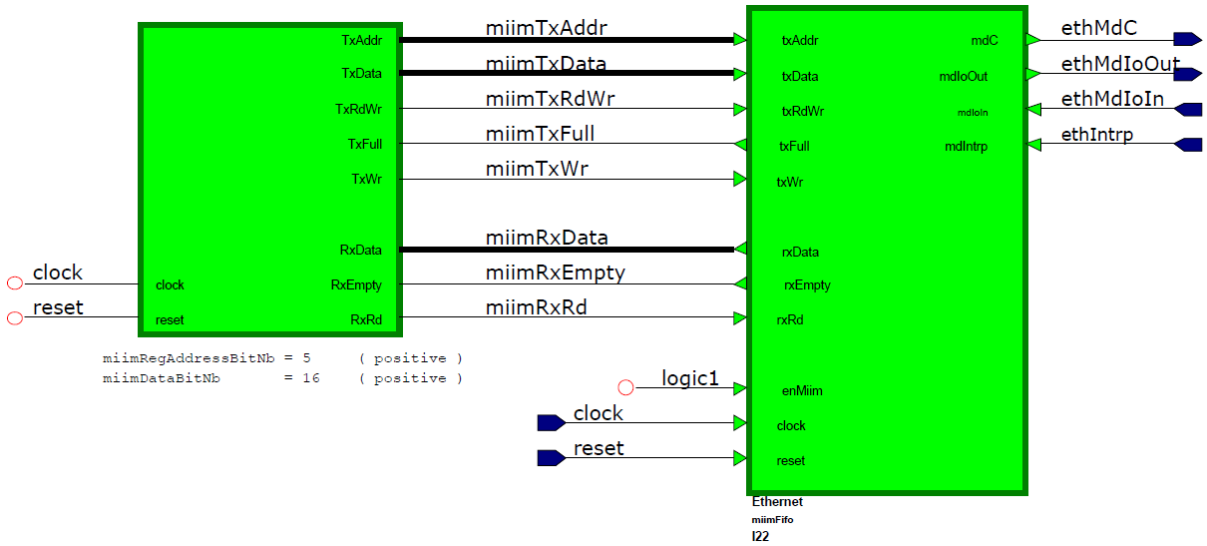


Abbildung 20: PHY Initialisierung

In einem ersten Prozess wird dazu ein Zählersignal erstellt. Dieses dient später als Referenzpunkt der Registeradresse. Dazu wird jedoch der Zähler durch zwei geteilt, sodass die Register Adresse sich jeweils nach zwei Takt Perioden ändert. Dies erlaubt es, in einer ersten Periode die Daten zu schreiben und in der zweiten Periode ein readEnable zu setzen. Somit wird sichergestellt, dass die Daten beim Lesen stabil sind. Diese Datenlinien werden in einem zweiten Prozess gesetzt. Dabei wird anhand der momentanen Registeradresse der richtige Datenwert gewählt und an den Ausgang geleitet.

Das FIFO dient der Zwischenspeicherung der Registerwerte, sodass sie unabhängig von der Tackt Geschwindigkeit des MIIM Busses in den Speicher geschrieben werden können. Anschliessend werden sie von einem weiteren Block gelesen. Dieser Block sowie das FIFO waren bereits Bestandteil der vorgegebenen Schaltung und wurden nicht angepasst. Sie befindet innerhalb des miimFIFO Blocks (siehe Abbildung 20 rechts).

Der zweite Block liest die Werte des FIFOs und serialisiert diese mit einer angepassten Geschwindigkeit, welche anhand eines Vorteilers gewählt werden kann. Die Serialisation entspricht dabei der in Abbildung 6 dargestellten Bitfolge. Am Ausgang sind somit die gewünschten Signale des MIIM Busses verfügbar.

Weiter zu beachten gilt es, dass die intern generierten Signale nicht dem gewünschten Tri-State Signal des MIIM Busses entspricht, sondern zwei separate Signale, eine Eingang und ein Ausgang, aufweist. Hierzu wird deshalb zusätzlich eine Konvertierung der Signale benötigt (siehe **Anhang 1** ethMdio Signal).

Die wichtigsten gewählten Registerwerte sind in der untenstehenden Tabelle ersichtlich.

Address	Name	Description	Value
0.13	Speed Select	1 = 100Mbps 0 = 10Mbps	1
0.12	Auto-Negotiation Enable	1 = Enable auto-negotiation process 0 = Disable auto-negotiation process	0
0.8	Duplex Mode	1 = Full-duplex 0 = Half-duplex	1
1f.15	HP_MDIX	0 = Micrel Auto MDI/MDI-X mode 1 = HP Auto MDI/MDI-X mode	1
1f.13	Pairswap Disable	1 = Disable auto MDI/MDI-X 0 = Enable auto MDI/MDI-X	0

Tabelle 5: wichtige Registerwerte

Frame Erstellung

Das folgende Kapitel beschreibt die vorgegebene Schaltung zur Kommunikation über das Media Independent Interface (MII), welche von der HES-SO // Valais - Wallis erstellt wurde. Aus platztechnischen Gründen wird hier nur eine grobe Übersicht der Schaltung gezeigt. Auf die Details der Implementation wird nicht eingegangen. Eine Übersicht der Schaltung ist in der untenstehenden Abbildung ersichtlich.

Die zwei Blöcke miiToRam und udpFifo im dienen dem Datenaustausch zwischen dem MII und der eigentlichen Applikation. Hierbei wird vorerst im Block miiToRam aus den eintreffenden Daten des MII ein Frame erstellt und in einen Speicher geschrieben. An dieser Stelle wird der Zeitpunkt des Timeouts bestimmt und mittels eines Signales rxStartOfFrame an den zuständigen PTP Block weitergeleitet.

Im folgenden Block udpFifo werden die Daten des Frames anschliessend extrahiert und an den Block udpApplication weiter gegeben. Dieser Block beinhaltet verschiedene UDP Funktionalitäten wie Echo, Quote of the Day oder Bonjour (siehe **Anhang 3**). An dieser Stelle wurde ebenfalls die PTP Applikation eingefügt, welche die PTP Pakete liest und deren Antwort erstellt.

Der Rückweg der Daten erfolgt in umgekehrter Reihenfolge durch die vorher beschriebenen drei Blöcke. So wird im udpApplication Block die Antwort auf das erhaltene Paket erstellt und an den Block udpFifo geleitet wo sie als Frame zwischengespeichert wird. Im Block miiToRam werden die Daten dieses Pakets anschliessend Stück für Stück über das MII weitergeleitet. Auch hier wird wiederum ein Signal erstellt welches den Zeitpunkt des Timeouts txStartOfFrame indiziert.

Ein Block zur Initialisierung des PHYs über den MIIM Bus wurde nachträglich zum bereits bestehenden Design angefügt (**Anhang 2** Seite 2). Dieser wird im Kapitel Digitale Schaltung zur Konfiguration genauer erklärt.

Der Block „MIIM Bus Controller“ dient wiederum der Initialisierung des PHYs. Er besteht aus einem FIFO und einem Block zur Serialisierung der Daten. Dies wird ebenfalls im Kapitel Digitale Schaltung zur Konfiguration genauer beschrieben.

Abbildung 21: Frame Erstellung

PTP Applikation

Die PTP Applikation entspricht einer UDP Applikation, welche durch die Portnummer 319 angesteuert wird. Die Applikation besteht aus einer Master oder Slave Applikation und aus zwei Blöcken, welche zum Erstellen und Auseinandernehmen von PTP Frames dienen (siehe Abbildung 22 und **Anhang 4**). Diese mussten angepasst werden, sodass sie innerhalb des vorgegebenen Designs der HES-SO // Valais – Wallis zur Verteilung der erhaltenen UDP Daten an die einzelnen Applikationen genutzt werden kann. Sie beinhalten jeweils einen Adresszähler, welcher nach jedem Lesen beziehungsweise Schreiben von Daten inkrementiert wird. Abhängig von diesem Wert können Daten den entsprechenden Registern zugewiesen werden und umgekehrt.

Um die Funktion von mehreren Applikationen zu garantieren, wird jeweils nur eine Applikation selektiert. Dazu muss die einkommende Portnummer mit der Portnummer der Applikation verglichen werden. Sind sie identisch wird dies durch das Signal rxSelectedInstance angezeigt.

Beim Senden von Daten muss ebenfalls beachtet werden, dass nicht mehrere Applikationen gleichzeitig schreiben wollen. Dazu wird zuerst eine Anfrage gestellt und nur dann gesendet, wenn die Anfrage eine positive Antwort zur Folge hat. Dies wird mit den Signalen writeRequest und writeGranted ermöglicht. Der Schiedsrichter befindet sich dabei innerhalb des Vorgegebenen Designs der HES-SO // Valais – Wallis.

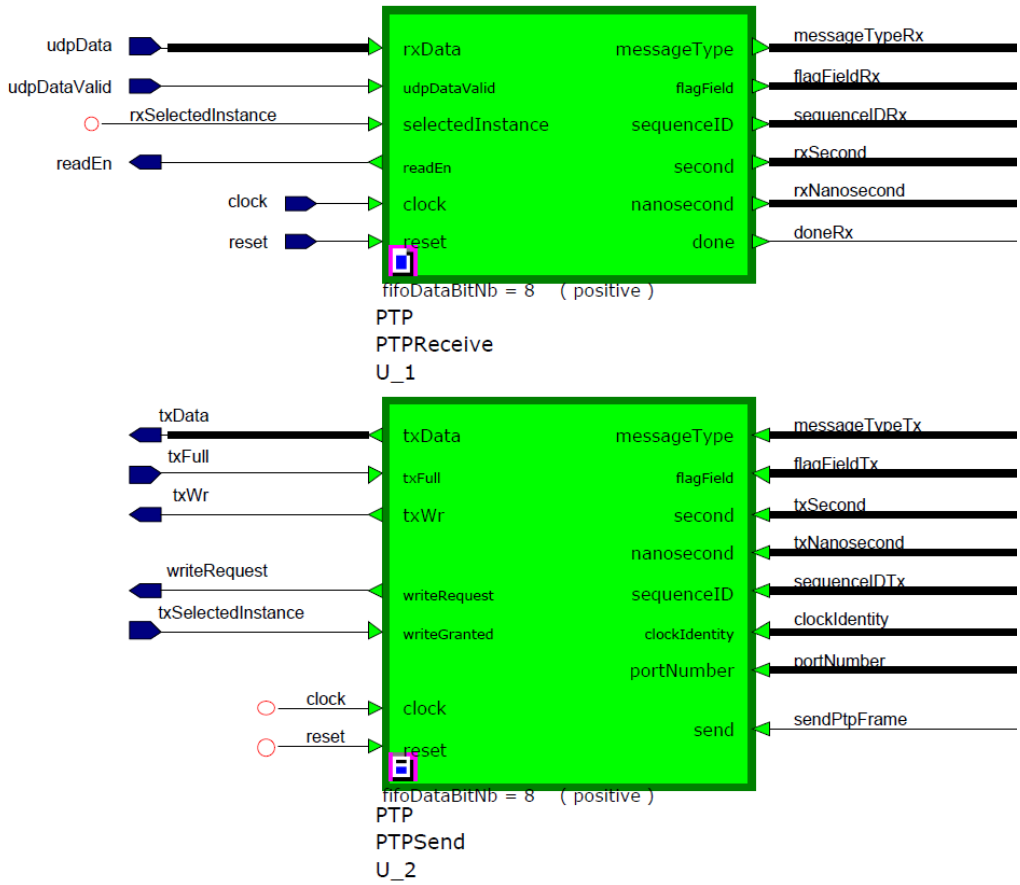


Abbildung 22: PTP Frame Pack/Unpack

Neben diesen zwei Blocken befinden sich ebenfalls die zwei Master und Slave Applikationen. Dabei wird jeweils nur derjenige der beiden Blöcke erstellt, welcher dem System entspricht. So enthält ein Slave System nur die Slave Applikation und ein Master System entsprechend nur die Master Applikation

Master Applikation

Die Master Applikation konnte vollständig von Herrn Johan Droz übernommen werden. Die Grundfunktion der Applikation ist in Abbildung 23 ersichtlich. Die einzelnen Zustände entsprechen dabei dem Paketaustausch des PTPs. Dabei soll der Zustand erst gewechselt werden, sobald ein PTP Frame tatsächlich gesendet wurde oder ein PTP Frame empfangen wurde. Diese Sendeerkennung wurde in einem externen Block ermöglicht. Weitere zwei externe Blöcke erstellen ein Timeout Signal und ein startSynchronisation Signal. Das Timeout Signal dient dazu, den Synchronisationsvorgang nach einer bestimmten Zeit zurückzusetzen, falls die DelayReq Nachricht nicht oder zu spät eintrifft. Das startSynchronisation Signal startet den Synchronisationszyklus. Die Übersicht der Master Applikation sowie die Zustandsmaschinen der einzelnen Blöcke sind in **Anhang 5 bis 8**, sowie **Anhang 12** ersichtlich.

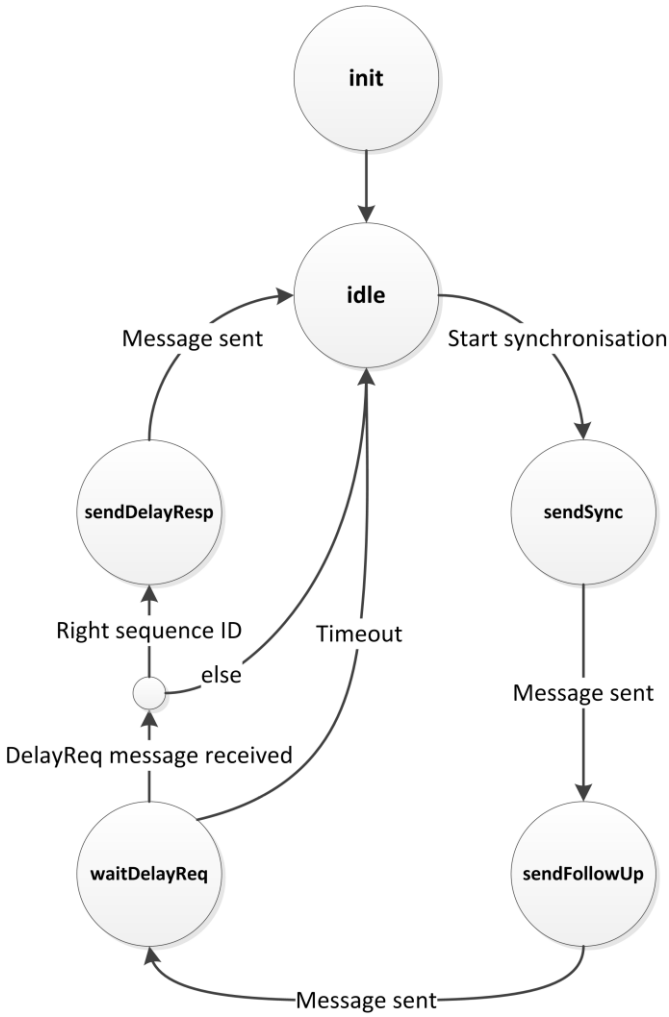


Abbildung 23: Master Applikation (Droz, 2012)

Slave Applikation

Zusätzlich zu den Zuständen welche den einzelnen Frames des PTPs entsprechen, enthält die Zustandsmaschine des Slaves weitere Zustände zur Berechnung des Offsets und zur Anpassung der lokalen Zeit (siehe Abbildung 24). Dabei wurde grundsätzlich das Design von Herrn Johan Droz übernommen, jedoch wurden mehrere Berechnungszustände hinzugefügt, um ein Software Pipelining zu ermöglichen (siehe Kapitel Software Pipelining).

Während dem Senden und Erhalten der Nachrichten wird jeweils ein Timestamp zwischengespeichert. Diese werden anschliessend zur Berechnung des Offsets benötigt.

Auch hier wurden extern Signale erstellt zur Erkennung des gesendeten Frames sowie zum Ermöglichen eines Timeouts und somit einer Zurücksetzung der Zustandsmaschine.

Die Übersicht der Slave Applikation, der VHDL Code der Haupt Zustandsmaschinen, sowie die Zustandsmaschinen der externen Blöcke sind in **Anhang 9 bis 12** ersichtlich.

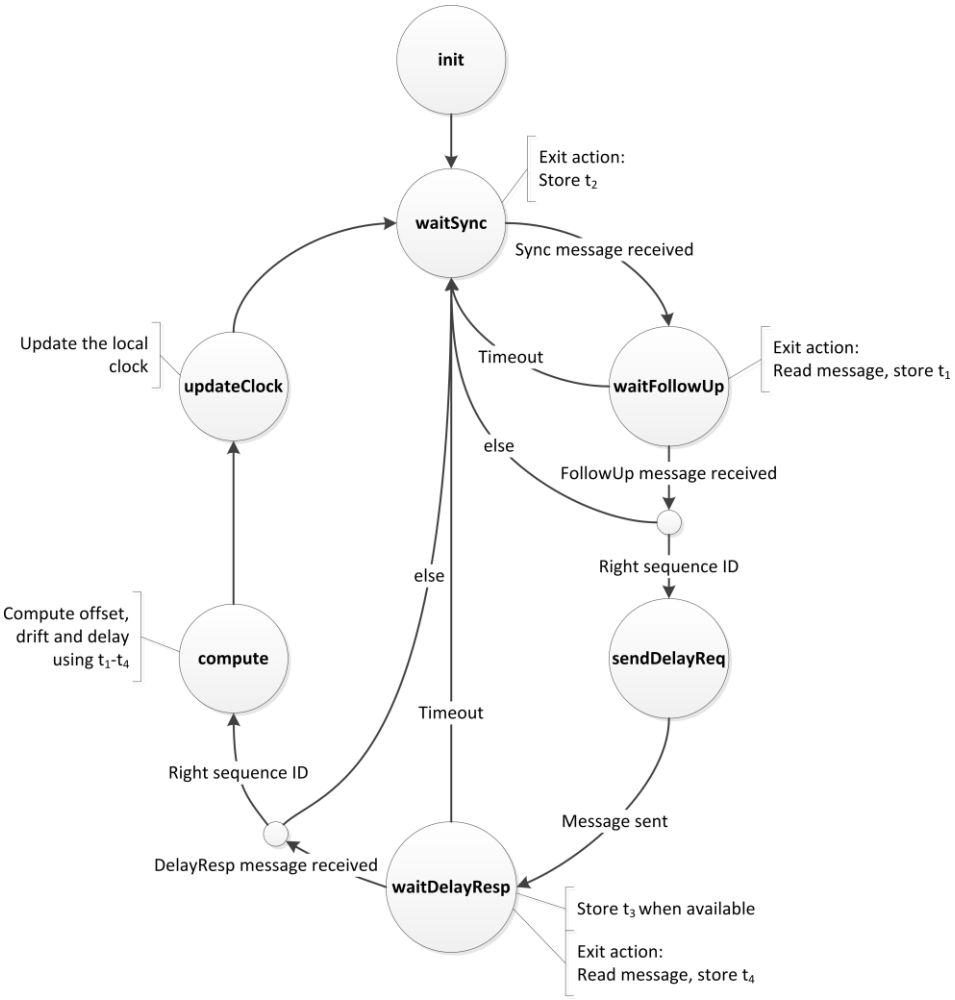


Abbildung 24: Slave Applikation (Droz, 2012)

PTP Zähler

Der PTP Zähler sowie die in den folgenden Kapiteln beschriebenen Blöcke befinden sich in einem separaten Block, welcher ausserhalb der Ethernet Kommunikation erstellt wurde. Die Übersicht dieses Blockes ist in **Anhang 13** ersichtlich.

Der PTP Zähler arbeitet Im Ethernet Clock Bereich. Abhängig davon ob es sich um ein Master oder Slave System handelt, muss das RX- (Slave) oder das TX-Clocksignal (Master) genutzt werden. Somit wird sichergestellt dass die verbundenen Master und Slave Systeme dasselbe Clock Signal zur Erstellung der lokalen Zeit nutzen und damit nicht auseinander driften. Hierzu wurde innerhalb der Signalsynchronisation ein neues Clock Signal erstellt welches je nach System einen der beiden RX oder TX Clocksignalen zugeordnet (siehe Kapitel Synchronisation).

Der Zähler erstellt die lokale Zeit des Systems. Diese besteht analog zum Timestamp aus einem 48bit Sekunden- und einem 32bit Nanosekundenwert. Dabei liegt der Nanosekundenwert ständig zwischen 0 und 10^9. Dieser Wert der Nanosekunden wird bei jeder steigenden Flanke des Clocksignals um die Clockperiode erhöht.

Ein Korrekturwert, sowie das dazugehörige enable Signal liegen als Eingang an diesem Block. Solange dieses enable Signal auf ‘1‘ liegt, wird der Wert der lokalen Zeit um den Korrekturwert angepasst. Durch die Möglichkeit der Korrektur kann das Slave System seine Zeit an das Master System anpassen.

Um zu verhindern dass der Nanosekundenwert ausserhalb der erwünschten Werte liegt, muss ständig überprüft werden ob er durch die Korrektur oder das Inkrementieren des Wertes aus dem gewünschten Bereich gelangt ist. Ist dies der Fall muss der Wert durch Addition beziehungsweise Subtraktion von 10^9 in den vorgegebenen Bereich gebracht werden. Ausserdem muss der Wert der Sekunden wegen dieser Änderung des Nanosekundenwerts durch Addition oder Subtraktion von einer Sekunde angepasst werden. Dieser Überlauf des Nanosekundenwertes dient zusätzlich als Pulse per Second Signal, welches pro Sekunde jeweils während einer Clockperiode auf ‘1‘ gesetzt wird.

Zu testzwecken wurde ein Signal hinzugefügt, welches es erlaubt den Zähler zurückzusetzen.

Der Code dieses Zählers ist in **Anhang 14** ersichtlich.

PTP Timestamp

Das Erstellen eines Timestamps wird durch ein einfaches Register ermöglicht, welches durch ein enable Signal einen neuen Wert annimmt. Dieses enable Signal entspricht dem Impuls, welcher innerhalb der Ethernet Kommunikation erstellt wurde. Der gespeicherte Wert entspricht der lokalen Zeit, welche vom PTP Zähler erstellt wird. Der PTP Timestamp Block befindet sich im Ethernet Clock Bereich, da sowohl das enable Signal als auch das lokale Zeitsignal im Ethernet Clock Bereich erstellt werden. Der Timestamp muss jedoch anschliessend in den Clock Bereich des lokalen FPGA Clocks synchronisiert werden (siehe Kapitel syncBlöcke). Wie im Kapitel Synchronisation zwischen Clock Bereichen stellt dies jedoch kein Problem dar, da sich die PTP Timestamps nur selten ändern, und somit die Empfänger Clockfrequenz in jedem Fall deutlich höher liegt.

Verbesserungen

Da bei einem Test unter ausgelastetem Ethernet Netzwerk (siehe Kapitel Ausgelastetes Netzwerk) eine Fehlfunktion erkannt wurde musste dieser Teil der Arbeit nachträglich abgeändert werden. Es wurde erkannt das Timestamps nicht nur von PTP Frames erstellt werden sondern von allen eintreffenden und gesendeten Frames. Dies führt zu Problemen, da die Frames einen Buffer durchlaufen und somit das momentan bearbeitete Frame nicht zwangsläufig zum momentanen Timestamp gehören muss.

Um diesen Effekt zu verhindern, muss die UDP Portnummer beim Erstellen eines Timestamps betrachtet werden. So kann sichergestellt werden, dass nur PTP Timestamps erstellt werden.

Da beim Empfangen eines Frames der UDP Port erst nach Abschluss des eigentlichen Timestamps ermittelt wird, muss dieser Timestamp zwischengespeichert werden und erst dann gegebenenfalls weitergeleitet werden, sobald der UDP Port bekannt ist. Hierzu wurde ein Puls Signal erstellt welches beim Ermitteln der Portnummer auf ‘1‘ gesetzt wird (siehe Abbildung 25).

Beim Senden eines PTP Paketes ist die Portnummer bereits vor dem Erstellen des Timestamps bekannt. Jedoch werden die erstellten Frames welche die Portnummer erhalten in einen Buffer gefüllt. Somit kann es vorkommen dass die Portnummer wechselt bevor das Frame gesendet wird(siehe Abbildung 26. Um trotzdem den dem Frame entsprechenden Port zu erhalten wurde hierfür ein FIFO erstellt. In dieses FIFO wird jeweils beim Erstellen eines neuen UDP Frames dessen Portnummer hinzugefügt. Bei jedem Speichern eines Timestamps während dessen Sendevorgang kann nun dessen entsprechende Portnummer aus dem FIFO gelesen werden. Der UDP Port wird somit wie auch das Frame in einem Buffer zwischengespeichert. Dieses FIFO ist in Abbildung 27 ersichtlich.

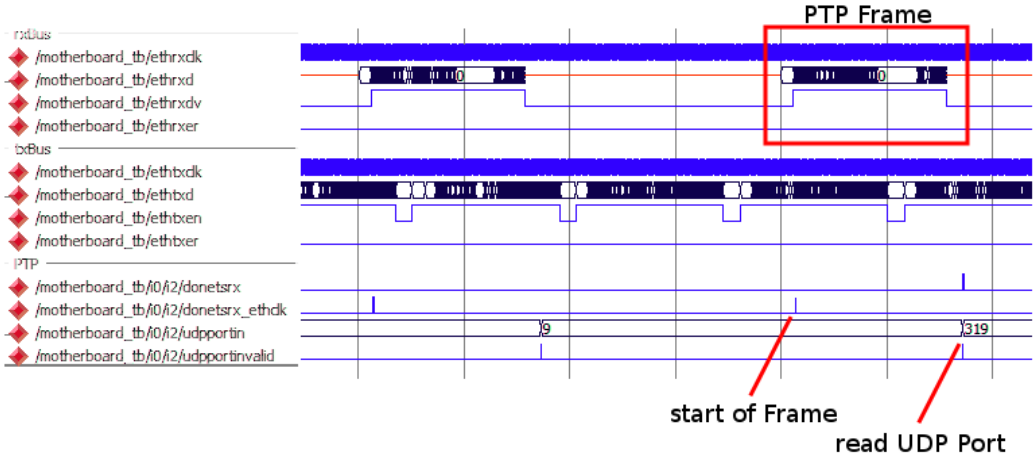


Abbildung 25: UDP Port beim Empfangen

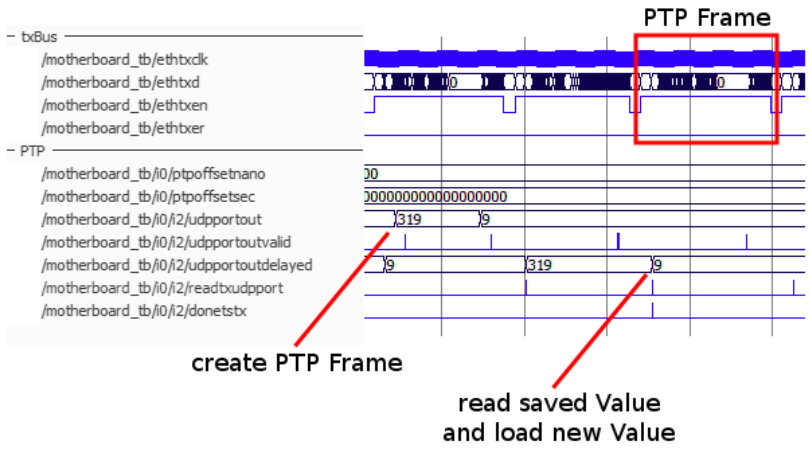


Abbildung 26: UDP Port beim Senden

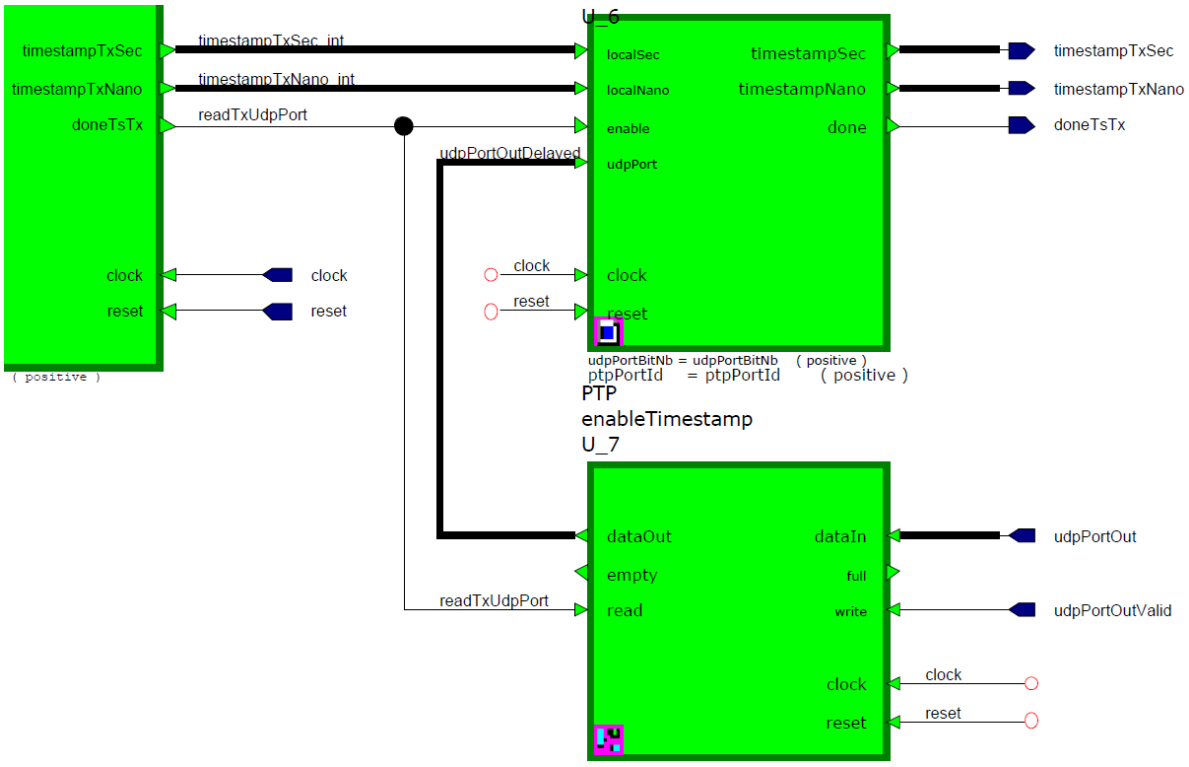


Abbildung 27: UDP Port FIFO

PTP Filter

Der Filter Block musste komplett neu erstellt werden, da eine serielle Verbindung mit einem Computer erstellt werden sollte. Diese dient dazu, die berechneten Offset Werte an einen Computer zu senden, welcher die Werte aufzeichnet und daraus ein Korrekturwert berechnet. Dieser Wert kann anschliessend an die FPGA zurückgesendet werden und zur Anpassung der Zeit verwendet werden.

In einem zweiten Teil wurde der serielle Filter durch einen lokalen Filter ersetzt, da sich herausstellte das der Filter sehr einfach zu implementieren ist.

Der Clock des Filters wird nur innerhalb des Slave Systems erstellt, da nur in diesem Fall ein Offset Wert berechnet werden soll und die Zeit angepasst wird.

Serielle Schnittstelle

Die serielle Übertragung wird mittels einer RS232 Schnittstelle ermöglicht. Hierzu konnte ein vorhandener Block der HES-SO // Valais – Wallis übernommen werde. Dieser kann wie ein FIFO geschrieben und gelesen werden. Die Daten werden dabei synchronisiert und mit der gewünschten Baudrate übermittelt.

Die Offset Werte, wie auch die Korrekturwerte werden im ASCII Format übertragen. Dies ermöglicht ein einfaches Lesen der Daten am Computer. Dazu wurden zwei Blöcke erstellt, welche die Binären Daten in ASCII Form umwandeln. Ebenfalls wurde eine Zustandsmaschine erstellt, welche die 32bit und 48bit Werte in Blöcke teilen, welche über die serielle Verbindung gesendet werden können. Eine weitere Zustandsmaschine rekonstruiert aus den erhaltenen Datenbytes die Korrekturwerte.

Die einzelnen Werte für Sekunden und Nanosekunden werden dabei durch ein Semikolon geteilt. Die Übertragung eines Wertes wird jeweils durch das Senden von carrier return (<cr>) und line feed (<lf>) abgeschlossen.

Die übersicht dieser Blöcke, sowie die beiden Zustandsmaschinen sind in **Anhang 15 bis 17** ersichtlich.
Der erstellte Python code ist in **Anhang 18** ersichtlich.

Lokale Korrektur

Aus den Tests ging hervor, dass der Korrektur Wert direkt dem invertierten Offset entspricht. Somit wurde ein einfacher lokaler Filter erstellt, welcher eine Korrektur erstellt, sobald der Offset Wert nicht innerhalb 0 und 20ns liegt. Innerhalb dieses Bereiches wird wegen Synchronisationsproblemen keine Korrektur vorgenommen (siehe Kapitel Synchronisation zwischen Clock Bereichen). Der Code zu diesem Block ist in **Anhang 19** ersichtlich.

Synchronisation

Wie im Kapitel Synchronisation zwischen Clock Bereichen ersichtlich wird, müssen Signale, welche in einen anderen Clock Bereichen verlaufen, synchronisiert werden. Hierzu wurden zwei Blöcke erstellt. Der eine Block erstellt dabei die Synchronisation zwischen dem RX und dem TX Clock Bereich. Der zweite führt abhängig davon ob es sich um ein Master oder Slave System handelt eine Synchronisation zwischen dem TX beziehungsweise RX und dem lokalen Clock Bereich aus.

Zur übersichtlichen Darstellung wurde ein weiteres Clock Signal (ethClock) erstellt, welches je nach System dem RX oder TX Clocksignal entspricht. Dies geschieht im ersten Block und ist in **Anhang 20** ersichtlich. Zusätzlich werden hier die Signale, welche den Zeitpunkt des Timestamps bestimmen synchronisiert. Hierbei muss nur eines der beiden Signale synchronisiert werden, da sich das zweite bereits im richtigen Clockbereich befindet. Das Signal txStartOfFrame gibt den Zeitpunkt eines gesendeten Frames an und ist somit im TX Clock Bereich, deshalb muss es in einem Master System nicht synchronisiert werden. Analog dazu verhält es sich mit dem rxStartOfFrame Signal beim Empfangen eines Frames und dem Slave System. Die Synchronisation dieser Pulse kann wie im Kapitel Logic Pulse beschrieben wurde synchronisiert werden. Dazu wurde ein Block benutzt, welcher bereits von der HES-SO // Valais – Wallis erstellt wurde.

Die Synchronisation zwischen dem neu erstellten ethClock und dem lokalen Clocksignal wird in dem weiteren Block erstellt. Dieser ist in **Anhang 21** ersichtlich. Hierbei wurden Pulse wiederum mit derselben Methode synchronisiert. Verschiedene Bus Signale konnten wie in Kapitel Bus Signale beschrieben mit einem Flipflop synchronisiert werden. Dazu wurden zwei Synchronisationsblöcke

erstellt, welche diese Synchronisation sowohl für Signed wie auch für Unsigned Werte mit generischer Grösse durchführen.

Tests

Grundfunktion

Simulation

Um die Grundfunktion der PTP Implementation zu testen wurde vorerst eine Simulation erstellt, dabei konnte ein Design der HES-SO // Valais – Wallis verwendet werden, welches es erlaubt das Empfangen von Frames zu simulieren, sowie gesendete Frames aufzuzeichnen.

Ein zur Simulation erstelltes Frame ist in der untenstehenden Abbildung ersichtlich.

```
at     30 us
info     Synch Message
frame    E4 AF A1 39 02 02        # destination MAC address
         E4 AF A1 39 02 01        # source MAC address
         08 00                    # IP Ethertype
            45 00 00 48
            00 00 40 00
            10 11 0B 51           # TTL, protocol:UDP, checksum
            C0 A8 6F 01           # source IP address
            C0 A8 6F 02           # destination IP address
               01 3F 01 3F        # source and dest. UDP ports
               00 34 0A FD        # length, checksum
                  80 02 00 2c 00 00 02 00 00 00
                  00 00 00 00 00 00 00 00 00 00
                  00 00 00 00 00 00 00 00 00 00
                  00 82 00 7f 00 00 50 77
         5D 80 26 E2 D5 80
```

Abbildung 28: PTP Test Frame

Der Paketaustausch funktionierte nach Anpassen der MAC und IP Adressen fehlerlos (siehe untenstehende Abbildung). Die Berechnung der Offsetwerte wurde dabei jedoch noch nicht getestet.

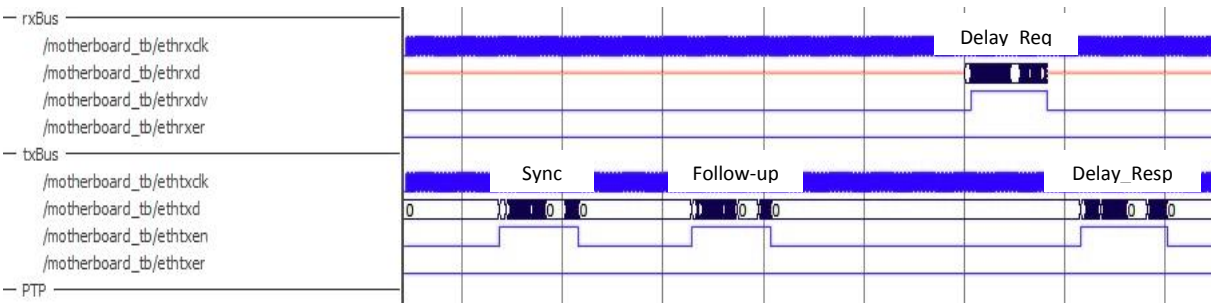


Abbildung 29: PTP Frame Austausch

Test der Berechnung

Vor dem ersten Test der Berechnung des Offsetwerts wurde die serielle Verbindung getestet, da diese Verbindung das Lesen des Offsetwertes deutlich erleichtert. Dieser Test verlief ohne Probleme.

Ein erstes Programmieren eines Master und Slave Systems zeigte, dass die Offsets oftmals sehr hohe Werte aufweisen. Der Grund dafür war einerseits eine fehlerhafte Synchronisation der Signale und andererseits eine fehlerhafter Berechnungsvorgang des Offsets.

Da die Timestamps Signale vorerst innerhalb des lokalen Clock Bereiches erstellt wurden, musste die lokale Zeit synchronisiert werden. Dies führte zu Problemen, da sich deren Wert bei jeder Clock Periode ändert. Nach Verschiebung der Timeouts in den Ethernet Clock Bereich konnte eine Verbesserung des Verhaltens erkennt werden.

Ein zweites Problem stellte die Berechnung des Offsetwertes dar. Diese Fehlfunktion wurde durch das Synthese Programm erkannt, da die Gatterverzögerung der Berechnung einen sehr hohen Wert annahm. Durch ein Software Pipelining konnte dieses Fehlverhalten beseitigt werden (siehe Kapitel Software Pipelining).

Nach der Behebung dieser Fehler wurden jeweils konstante Werte des Offsets erhalten. Dies deutet darauf hin, dass eine einmalige Korrektur der Zeit genügt, um ein synchrones Verhalten der Systeme zu erhalten.

Nach der Implementierung des Python-Codes konnte diese Korrektur vorgenommen werden. Die Werte des Offsets lagen nun jeweils bei 0ns oder 20ns, das reale Offset zwischen beiden Systemen war jedoch konstant bei etwa 5ns (siehe Kapitel Master-Slave Differenz). Eine Erklärung für das Auftreten einer Fehlberechnung von 20ns stellt die Synchronisation dar (siehe Kapitel Synchronisation zwischen Clock Bereichen). Aus diesem Grund wurde anschliessend dieser 20ns Wert innerhalb des Filters ignoriert, da davon ausgegangen werden kann, dass das reale Offset diesen Fehlerwert nicht aufweist.

Ausgelastetes Netzwerk

Um die Funktionalität des Precision Time Protokolls innerhalb einem stark ausgelasteten Ethernet Netzwerks zu testen, wurden sogenannte Discard Frames gesendet. Es handelt sich dabei um UDP Pakete mit Port 9, welche von dem Empfänger verworfen werden.

Um ein ausgelastetes Netzwerk zu simulieren wurde ein Zähler erstellt, welcher die FPGA Clockfrequenz teilt und den Impuls zum Senden eines Frames erstellt. Durch die Anzahl Bit des Overflowzählers, sowie der Datengrösse des Discard Frames konnte bestimmt werden, wie hoch die Auslastung des Netzwerks ist. Dabei muss zusätzlich zu den einzelnen Headern und den Daten des Discard Frames ebenfalls das Interframe Gap und die Preamble (siehe Abbildung 30) betrachtet werden, da das Netzwerk zu dieser Zeit ebenfalls belegt ist.

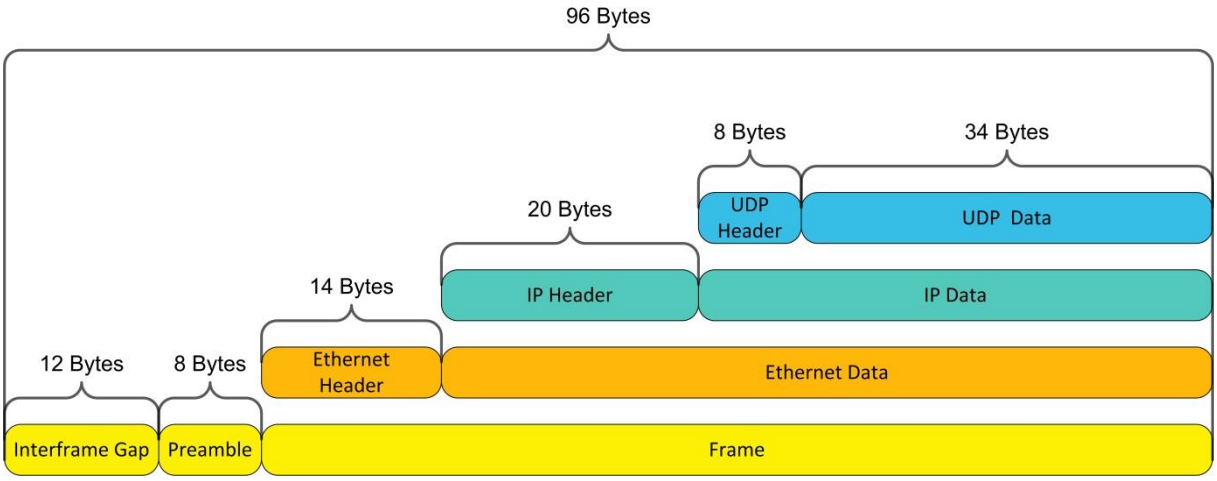


Abbildung 30: Aufbau Discard Frame

Anhand der FPGA Clockfrequenz und des Frequenzteilers wird bestimmt wie viele Frames pro Sekunde gesendet werden. Anhand dieses Wertes und der Anzahl an Bits pro Frame kann die Auslastung des Ethernet Netzwerks bestimmt werden. Die detaillierte Berechnung ist in Abbildung 31 ersichtlich.

```
InterframeGap = 12;
Preamble = 8;
EthHeader = 14;
IpHeader = 20;
UDPHeader = 8;
UDPData = 34;
BytesPerFrame = InterframeGap + Preamble + EthHeader +
IpHeader + UDPHeader + UDPData

ClkFreq = 66e6;
ClkDividerBitNb = 9;
ClkDivider = 2^ClkDividerBitNb;

FramesPerSec = ClkFreq/ClkDivider
BitsPerSec = FramesPerSec * BytesPerFrame * 8;

EthBitPerSec = 100e6;
EthOccupancy = BitsPerSec/EthBitPerSec * 100


  EthOccupancy =

                    99%
```

Abbildung 31: Berechnung der Ethernet Auslastung

Ein erster Test mit diesem Aufbau brachte zum Vorschein, dass bei starker Auslastung des Netzwerks nicht nur Timestamps der PTP Frames sondern auch von anderen Frames erstellt wurden. Dies führt

zu einem Fehlverhalten, da der aktuellste Timestamp nicht zwangsläufig zu einem PTP Paket gehören muss. Es ist möglich, dass zwischen den einzelnen PTP Paketen andere Pakete empfangen und gesendet werden.

Nach dieser Abänderung des Designs (siehe Verbesserungen in Kapitel PTP Timestamp) wurde die Schaltung erneut getestet. Die Ergebnisse der Synchronisation innerhalb eines ausgelasteten Netzwerkt wiesen dabei keine Unterschiede zum normalen Betrieb auf. Somit verlief der Test erfolgreich.

Messung der Zeitdifferenz

Master-Slave Differenz

Um die reale Zeitverschiebung zwischen den zwei synchronen Systemen zu messen, wurde ein Pulssignal des PTP Zählers auf den Ausgang der FPGA gelegt. Somit konnten diese Signale des Masters sowie des Slaves auf einem Oszilloskop miteinander verglichen werden. In der untenstehenden Abbildung ist der gemessene Zeitunterschied zwischen den beiden Pulen pro Sekunde zu sehen. Der Wert dieser Messung bewegt sich um 5ns ± 1ns und ist abhängig von den Ethernet Clock Signalen. Die Phasenverschiebung der zwei Clocksignale entsteht durch die Rekonstruktion des Signales innerhalb des PHYs und kann bei verschiedenen Bauteilen unterschiedlich auftreten. Die gemessenen Werte sind somit nur für den PHY ksz8041nl gültig.

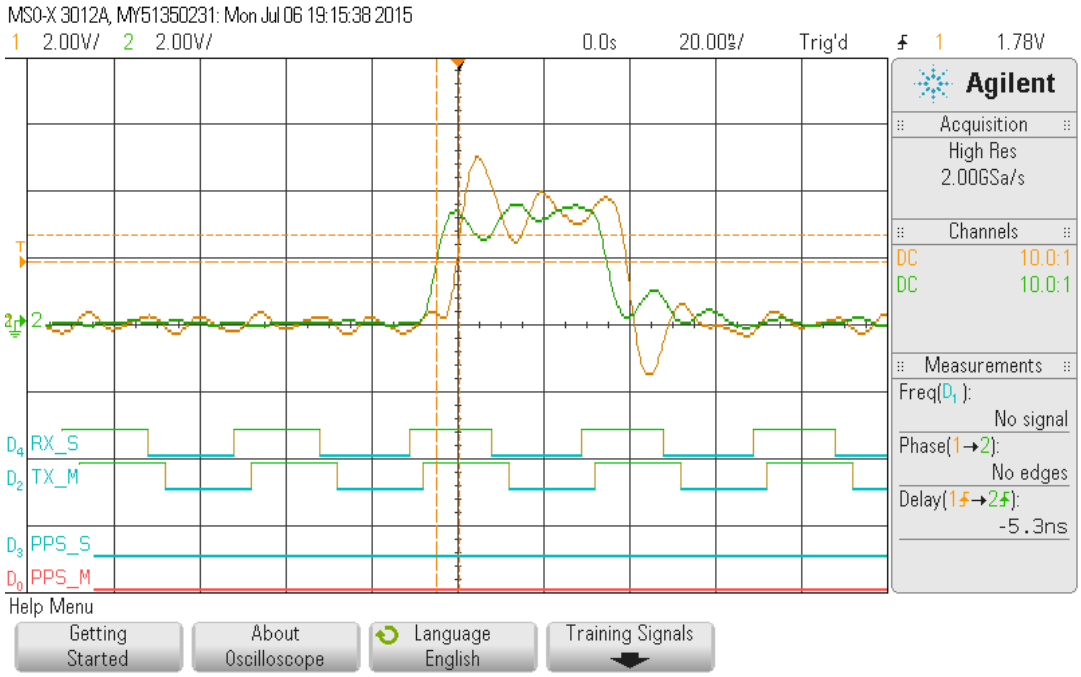


Abbildung 32: Zeitunterschied Master-Slave

Slave-Slave Differenz

Die Zeitdifferenz zwischen Master und Slave hat keinen Einfluss auf die Differenz zwischen zweier Slave Systemen bei der Benutzung einer Boundary Clock (siehe Kapitel Boundary Clock), da hier die verwendeten Clock Signale alle von demselben Clocksignal rekonstruiert werden. Jedoch bewirkt die Master-Slave Zeitdifferenz eine Verzögerung der Boundary Clock in Bezug zur Grandmaster Clock. Somit werden die Slave Systeme ebenfalls eine Verzögerung gegenüber des Grandmaster Clocks aufweisen.

Um die Zeitverschiebung zwischen zwei Systemen zu überprüfen wurde ein Ethernet Switch verwendet, welcher dasselbe Clocksignal zur Erstellung alles Port Ethernet Clock Signale benutzt. Die Zeitmessung ist in der untenstehenden Abbildung ersichtlich. Die gemessenen Differenzen bewegen sich zwischen ±1ns. Somit kann festgestellt werden, dass die Rekonstruktion der Clocksignale innerhalb mehreren ksz8041nl PHYs sich nicht um mehr als 1ns unterscheidet.

Eine konkrete Messung der Pulse pro Sekunde war nicht möglich, da kein PTP Switch vorhanden war und eine Synchronisation mehrerer Systeme nicht durgeführt werden konnte. Da dieser Puls pro Sekunde direkt abhängig vom rekonstruierten Clocksignal ist, kann angenommen werden, dass die Genauigkeit der Pulse derjenigen der Clocksignalen entspricht.

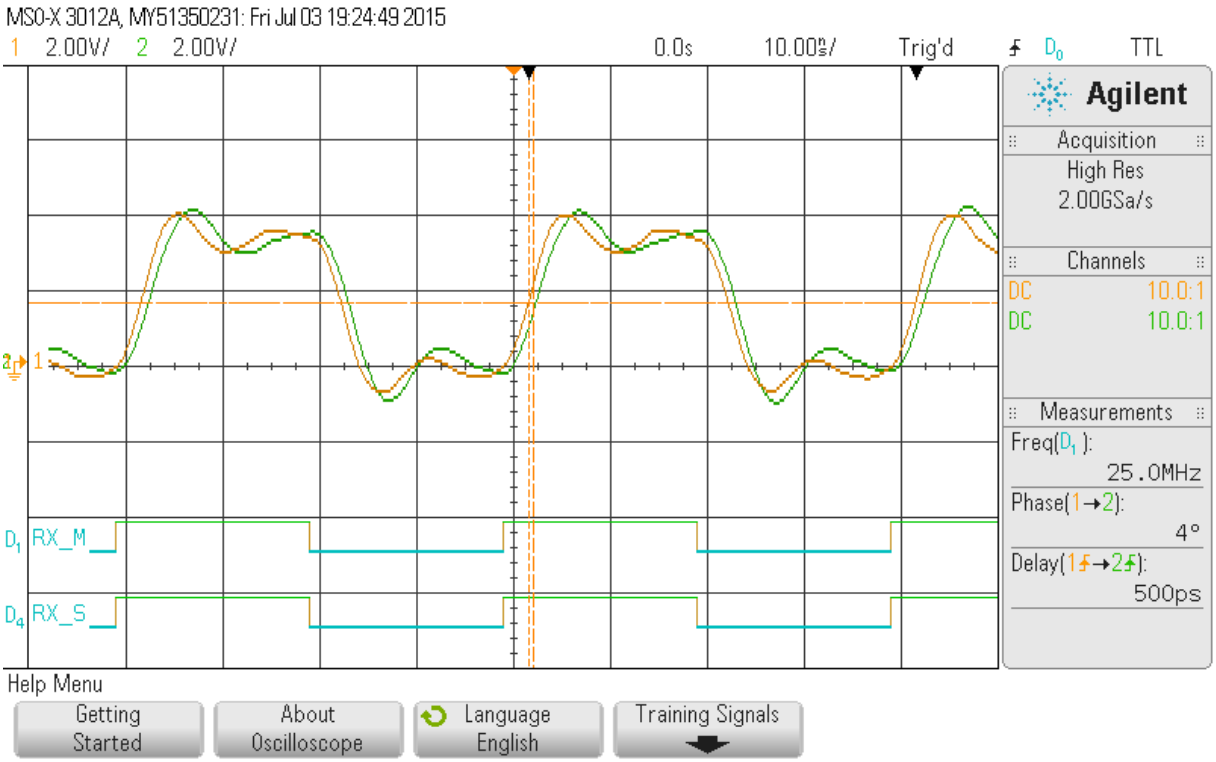


Abbildung 33: Zeitunterschied Slave-Slave

Anmerkung

Wird beim PHY ksz8041nl das automatische Erkennen der Sende- und Empfangsleitungen (Auto MDI-X) ausgeschaltet führt dies dazu, dass die Phasenverschiebung zwischen dem rekonstruierten Ethernet Clocksignal und des eigentlichen Signals nicht mehr konstant bei etwa 5ns liegt, sondern sich nach jeder Trennung des Kabels ändert. Somit wird die Genauigkeit dieser Implementation des Precision Time Protokolls stark beeinträchtigt, da davon ausgegangen werden muss, dass die Signale eine Zeitdifferenz von bis zu einer Clockperiode (40ns) aufweisen können. Aus fehlenden Informationen zum PHY konnte dieses Verhalten nicht genauer analysiert werden.

Um sicherzustellen dass die Implementation mit einem anderen PHY korrekt funktioniert, sollten vorerst deren Clocksignale und insbesondere die Rekonstruktion dieser Signale überprüft werden.

Schlussfolgerung

Die gesetzten Ziele dieser Arbeit konnten vollständig erreicht werden.

Eine Synchronisation zwischen zwei Systemen wurde mittels des Precision Time Protokolls erstellt. Dabei wurde eine Implementation der Diplomarbeit von Herrn Johan Droz (Droz, 2012) genutzt und an die gewünschte Anwendung angepasst. Zur Kommunikation wurde ein bestehendes Design der HES-SO // Valais – Wallis übernommen. Dieses wurde ebenfalls leicht abgeändert, so dass es die PTP Implementation besser unterstützt.

Damit eine durchgehende Korrektur der Slave Systeme nicht mehr nötig ist, wurden einige PTP Funktionen in den Ethernet Clock Bereich verschoben, sodass mehrere Systeme dasselbe Clocksignal als Referenz zur Erstellung der lokalen Zeit benutzen. Durch das Vorhandensein dieses Clocksignals auf mehreren Systemen ist nun lediglich eine einmalige Korrektur der Zeit jedes Systems nötig, Korrekturen während einer bestehenden Verbindung werden durch die identische Frequenz auf den verschiedenen Systemen überflüssig.

Der Regler, welcher Korrekturwerte aus den berechneten Offsetwerten erstellt, wurde sowohl lokal, als auch mittels eines Python-Codes auf einem Computer erstellt. Dazu wurde eine serielle Verbindung zwischen Computer und FPGA erstellt. Der Python-Code ermöglicht zusätzlich das Aufzeichnen von Offset Werten, sodass die Genauigkeit und Stabilität des Systems besser analysiert werden konnten.

Bei der Umsetzung dieses Projektes auf einem Gigabit Ethernet Netzwerk müssen die verschiedenen PHYs der Systeme vorerst genauer analysiert werden, da dieses Projekt auf dem PHY ksz8041nl von Micrel basiert, welcher nicht Gigabit Ethernet fähig ist. Ein anderes Verhalten der Ethernet Clock Rekonstruktion kann die erstellte Implementation des Protokolls deutlich ändern und die Genauigkeit beeinflussen.

Ein grosser Vorteil dieser Implementation liegt bei der Einfachheit des Reglers durch die Verwendung desselben Clocksignals auf mehreren Systemen. Eine Korrektur ist somit nur einmalig bei der Verbindung mit dem Netzwerk nötig, im Gegensatz zu anderen Implementationen, bei welchen eine ständige Korrektur ausgeführt werden muss.

Bibliographie

Altera, C. (2009). Abgerufen am 2015 von https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01082-quartus-ii-metastability.pdf

Cummings, C. E. (2008). *Clock Domain Crossing (CDC) Design & Verification Techniques Using System Verilog.* Boston: SNUG.

Droz, J. (2012). *Base de temps pour réseaux distribués.* Hes-so Wallis.

Micrel. (2008). *KSZ8041NL.* Von http://www.micrel.com/_PDF/Ethernet/datasheets/ksz8041nl.pdf abgerufen

Skoog, P. (2010). *Improving Synchronization With 1588 Transparent Clocks*. Von http://www4.evaluationengineering.com/articles/201001/improving-synchronization-with-1588-transparent-clocks.php abgerufen

TI. (2015). *DP83640.* Von http://www.ti.com/lit/ds/symlink/dp83640.pdf abgerufen

Wikipedia. (2013). *4B5B*. Von http://en.wikipedia.org/wiki/4B5B abgerufen

Wikipedia. (2015). *Management Data Input/Output*. Von http://en.wikipedia.org/wiki/Management_Data_Input/Output abgerufen

Wikipedia. (2015). *Media Independent Interface*. Von http://en.wikipedia.org/wiki/Media-independent_interface abgerufen

Wikipedia. (2015). *MLT-3 encoding*. Von http://en.wikipedia.org/wiki/MLT-3_encoding abgerufen

Anhänge

Alle Anhänge sind in digitaler Form auf der beigelegten CD ersichtlich.

Anhang 1: Top Level
Anhang 2: Motherboard
Anhang 3: UDP Applikation
Anhang 4: UDP Appl PTP
Anhang 5: Application Master
Anhang 6: Master FSM
Anhang 7: Master Timeout Controll
Anhang 8: Synch Control
Anhang 9: Application Slave
Anhang 10: Slave Txt FSM
Anhang 11: Slave Timeout Control
Anhang 12: Busy Network Control
Anhang 13: PTP
Anhang 14: PTP Clock Counter
Anhang 15: Uart Filter
Anhang 16: Send Offset
Anhang 17: Receive Correction
Anhang 18: Python Code
Anhang 19: PTP Filter Lokal
Anhang 20: Synch Rx Tx Clk
Anhang 21: Synch Eth Main Clk
Anhang 22: Initialisation Code

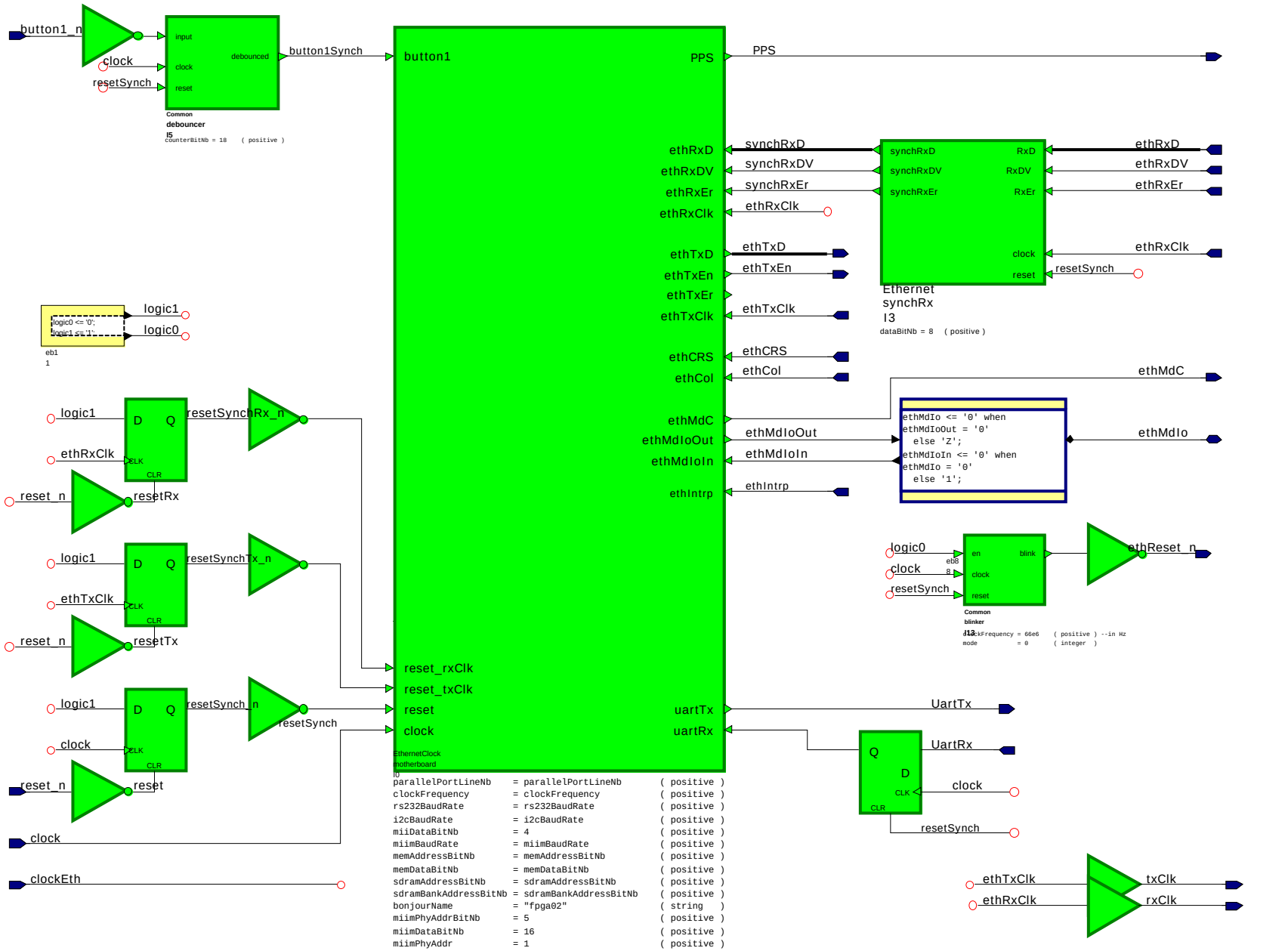
Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;
LIBRARY Common;
USE Common.CommonLib.all;
```

Declarations

Pre User:

```
    constant clockFrequency: positive := 66E6;
    constant rs232BaudRate: positive := 9600;
    constant i2cBaudRate: positive := 50E3;
    constant miimBaudRate: positive := 2500E3;


    constant miimRegAddressBitNb : positive := 5;
    constant miimDataBitNb : positive := 16;

    constant sdramAddressBitNb: positive := 12;
    constant sdramBankAddressBitNb: positive := 2;
```



<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>	<enter comments here>	
Path:	Board/PTP_SLAVE/struct		
Edited:	by samuel.stucky on 06 juil. 2015		

Package List

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 USE ieee.numeric_std.all;

LIBRARY Ethernet;
USE Ethernet.ethernet.all;

LIBRARY PTP;
 USE PTP.config.all;
```

Declarations

Ports:

```
button1                 : std_ulogic
clock                   : std_ulogic
ethCRS                  : std_uLogic
ethCol                  : std_uLogic
ethIntrp                : std_uLogic
ethMdIoIn               : std_logic
ethRxClk                : std_uLogic
ethRxD                  : std_ulogic_vector(miiDataBitNb-1 DOWNTO 0)
ethRxDV                 : std_uLogic
ethRxEr                 : std_uLogic
ethTxClk                : std_uLogic
reset                   : std_ulogic
reset_rxClk             : std_ulogic
reset_txClk             : std_ulogic
uartRx                  : std_uLogic
PPS                     : std_ulogic
ethMdC                  : std_uLogic
ethMdIoOut              : std_logic
ethTxD                  : std_ulogic_vector(miiDataBitNb-1 DOWNTO 0)
ethTxEn                 : std_uLogic
ethTxEr                 : std_uLogic
uartTx                  : std_uLogic
```

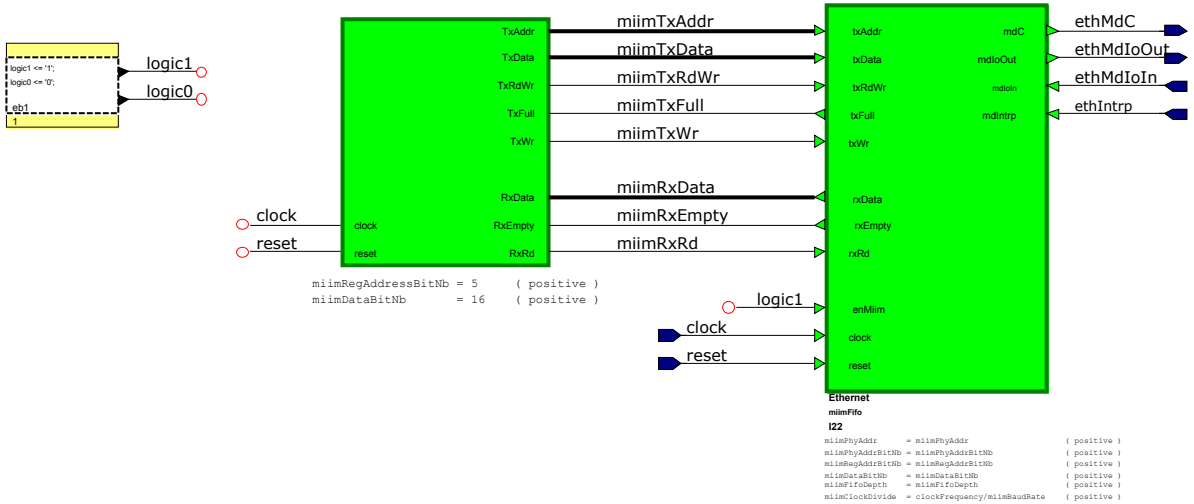
Pre User:

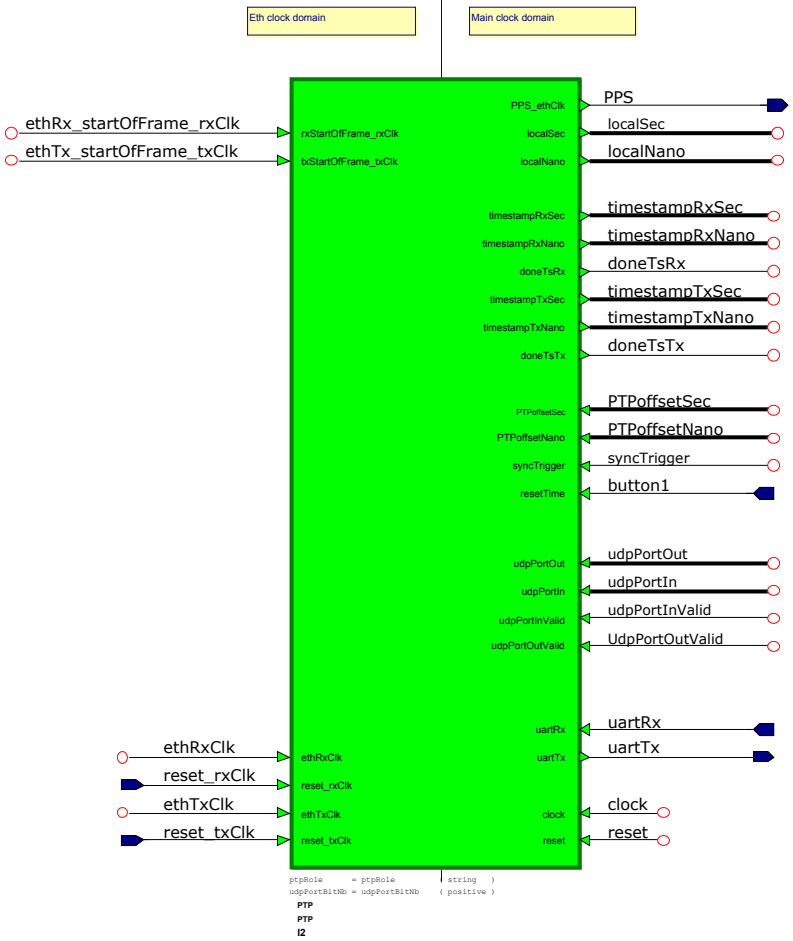
```
constant rs232DataBitNb: positive := 8;
constant rs232FifoDepth: positive := 16;

constant bramAddressBitNb: positive := 8;

constant i2cDataBitNb: positive := 10;
constant i2cFifoDepth: positive := 8;

constant miimRegAddrBitNb : positive := 5;
constant miimFifoDepth : positive := 8;

constant macAddressBitNb : positive := 48;
constant ipAddressBitNb : positive := 32;
constant udpPortBitNb : positive := 16;
constant ethRamAddressBitNb : positive := 10;
constant ethRamDataBitNb : positive := 16;
constant ethFifoDataBitNb : positive := 8;

constant regOut_RegNb : positive := 4;
constant regIn_RegNb : positive := 4;
constant regUDPPortID : positive := 2000;
```

Diagram Signals:

```
SIGNAL PTPoffsetNano            : signed(UInt32BitNb-1 DOWNTO 0)
SIGNAL PTPoffsetSec             : signed(UInt48BitNb-1 DOWNTO 0)
SIGNAL UdpPortOutValid          : std_ulogic
SIGNAL destIpAddr               : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddr              : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destUdpPort              : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL doneTsRx                 : std_ulogic
SIGNAL doneTsTx                 : std_ulogic
SIGNAL ethRxData                : std_ulogic_vector(ethFifoDataBitNb-1 DOWNTO 0)
SIGNAL ethRxEmpty               : std_ulogic
SIGNAL ethRxRd                  : std_ulogic
SIGNAL ethRx_address            : unsigned(ethRamAddressBitNb -1 DOWNTO 0)
SIGNAL ethRx_baseAddress        : unsigned(ethRamAddressBitNb-1 DOWNTO 0)
SIGNAL ethRx_data               : std_ulogic_vector(ethRamDataBitNb-1 DOWNTO 0)
SIGNAL ethRx_startOfFrame       : std_ulogic
SIGNAL ethRx_startOfFrame_rxClk : std_ulogic
SIGNAL ethTxData                : std_ulogic_vector(ethFifoDataBitNb-1 DOWNTO 0)
SIGNAL ethTxFull                : std_ulogic
SIGNAL ethTxWr                  : std_ulogic
SIGNAL ethTx_address            : unsigned(ethRamAddressBitNb-1 DOWNTO 0)
SIGNAL ethTx_baseAddress        : unsigned(ethRamAddressBitNb-1 DOWNTO 0)
SIGNAL ethTx_data               : std_ulogic_vector(ethRamDataBitNb-1 DOWNTO 0)
SIGNAL ethTx_endOfFrame         : std_ulogic
SIGNAL ethTx_startOfFrame_txClk : std_ulogic
SIGNAL ethTx_write              : std_ulogic
SIGNAL ipAddress                : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL isBroadcast              : std_ulogic
SIGNAL isMulticast              : std_ulogic
SIGNAL localNano                : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL localSec                 : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL logic0                   : std_ulogic
SIGNAL logic1                   : std_ulogic
SIGNAL macAddress               : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL miimRxData               : std_ulogic_vector(miimDataBitNb-1 DOWNTO 0)
SIGNAL miimRxEmpty              : std_ulogic
SIGNAL miimRxRd                 : std_ulogic
SIGNAL miimTxAddr               : unsigned(miimPhyAddrBitNb-1 DOWNTO 0)
SIGNAL miimTxData               : std_ulogic_vector(miimDataBitNb-1 DOWNTO 0)
SIGNAL miimTxFull               : std_ulogic
SIGNAL miimTxRdWr               : std_ulogic
SIGNAL miimTxWr                 : std_ulogic
SIGNAL reg_In_data              : reg_type(regIn_RegNb-1 DOWNTO 0)
SIGNAL reg_In_en                : std_ulogic_vector(regIn_RegNb-1 DOWNTO 0)
SIGNAL reg_Out_data             : reg_type(regOut_RegNb-1 DOWNTO 0)
SIGNAL reg_Out_en               : std_ulogic_vector(regOut_RegNb-1 DOWNTO 0)
SIGNAL sourceIPAddr             : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL sourceMacAddr            : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL sourceUdpPort            : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL syncTrigger              : std_ulogic
SIGNAL timestampRxNano          : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampRxSec           : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL timestampTxNano          : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampTxSec           : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL udpChecksumError         : std_ulogic
SIGNAL udpPortIn                : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortInValid           : std_ulogic
SIGNAL udpPortOut               : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```

Post User:


logic1 <= '1';
logic0 <= '0';
eb1
1
logic1
logic0
TxAddr
TxData
TxRdWr
TxFull
TxWr
RxData
RxEmpty
RxRd
clock
reset
miimTxAddr
miimTxData
miimTxRdWr
miimTxFull
miimTxWr
miimRxData
miimRxEmpty
miimRxRd
miimRegAddressBitNb = 5 (positive)
miimDataBitNb = 16 (positive)
txAddr
txData
txRdWr
txFull
txWr
rxData
rxEmpty
rxRd
enMiim
clock
reset
logic1
clock
reset
mdC
mdIoOut
mdIoIn
mdIntrp
ethMdC
ethMdIoOut
ethMdIoIn
ethIntrp
Ethernet
miimFifo
I22
miimPhyAddr = miimPhyAddr (positive)
miimPhyAddrBitNb = miimPhyAddrBitNb (positive)
miimRegAddrBitNb = miimRegAddrBitNb (positive)
miimDataBitNb = miimDataBitNb (positive)
miimFifoDepth = miimFifoDepth (positive)
miimClockDivide = clockFrequency/miimBaudRate (positive)



<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>	<enter comments here>	
Path:	EthernetClock/motherboard/struct		
Edited:	by samuel.stucky on 22 juin 2015		

Package List

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

LIBRARY Ethernet;
USE Ethernet.ethernet.all;

LIBRARY PTP;
USE PTP.config.all;

Declarations

Ports:

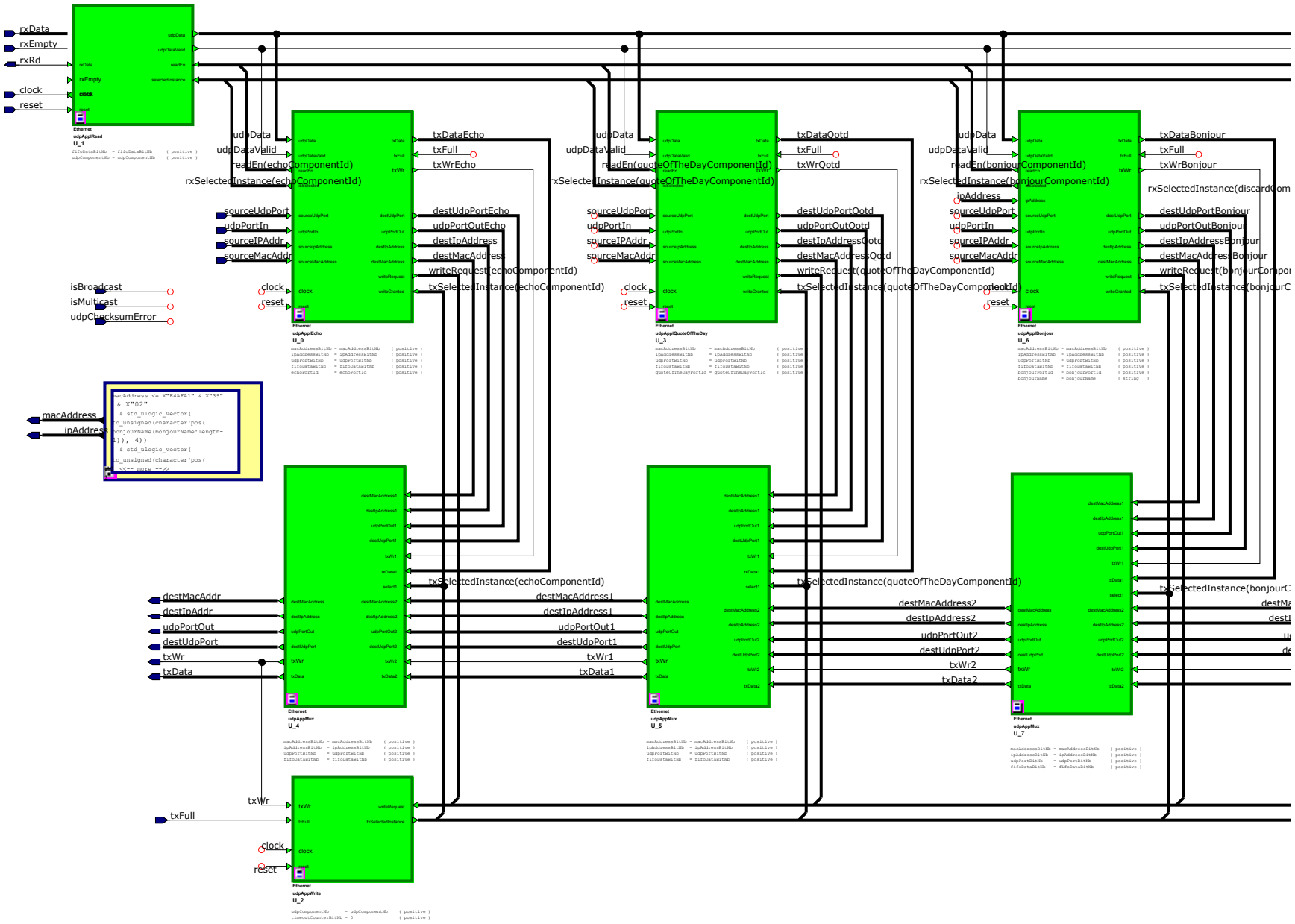
```
PTPlocalNano            : unsigned(UInt32BitNb-1 DOWNTO 0)
PTPlocalSec             : unsigned(UInt48BitNb-1 DOWNTO 0)

clock                   : std_ulogic
ethTxEn                 : std_ulogic
isBroadcast             : std_ulogic
isMulticast             : std_ulogic
reg_Out_data            : reg_type(regOut_RegNb-1 DOWNTO 0)
reg_Out_en              : std_ulogic_vector(regOut_RegNb-1 DOWNTO 0)
reset                   : std_ulogic
rxData                  : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
rxEmpty                 : std_ulogic
sendDiscardFrame        : std_ulogic
sourceIPAddr            : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
sourceMacAddr           : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
sourceUdpPort           : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
timestampRxDone         : std_ulogic
timestampRxNano         : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampRxSec          : unsigned(UInt48BitNb-1 DOWNTO 0)
timestampTxDone         : std_ulogic
timestampTxNano         : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampTxSec          : unsigned(UInt48BitNb-1 DOWNTO 0)
txFull                  : std_ulogic
udpChecksumError        : std_ulogic
udpPortIn               : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
PTPoffsetNano           : signed(UInt32BitNb-1 DOWNTO 0)
PTPoffsetSec            : signed(UInt48BitNb-1 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)
ipAddress               : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
macAddress              : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
reg_In_data             : reg_type(regIn_RegNb-1 DOWNTO 0)
reg_In_en               : std_ulogic_vector(regIn_RegNb-1 DOWNTO 0)
rxRd                    : std_ulogic
syncTrigger             : std_ulogic
txData                  : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
txWr                    : std_ulogic
udpPortOut              : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```

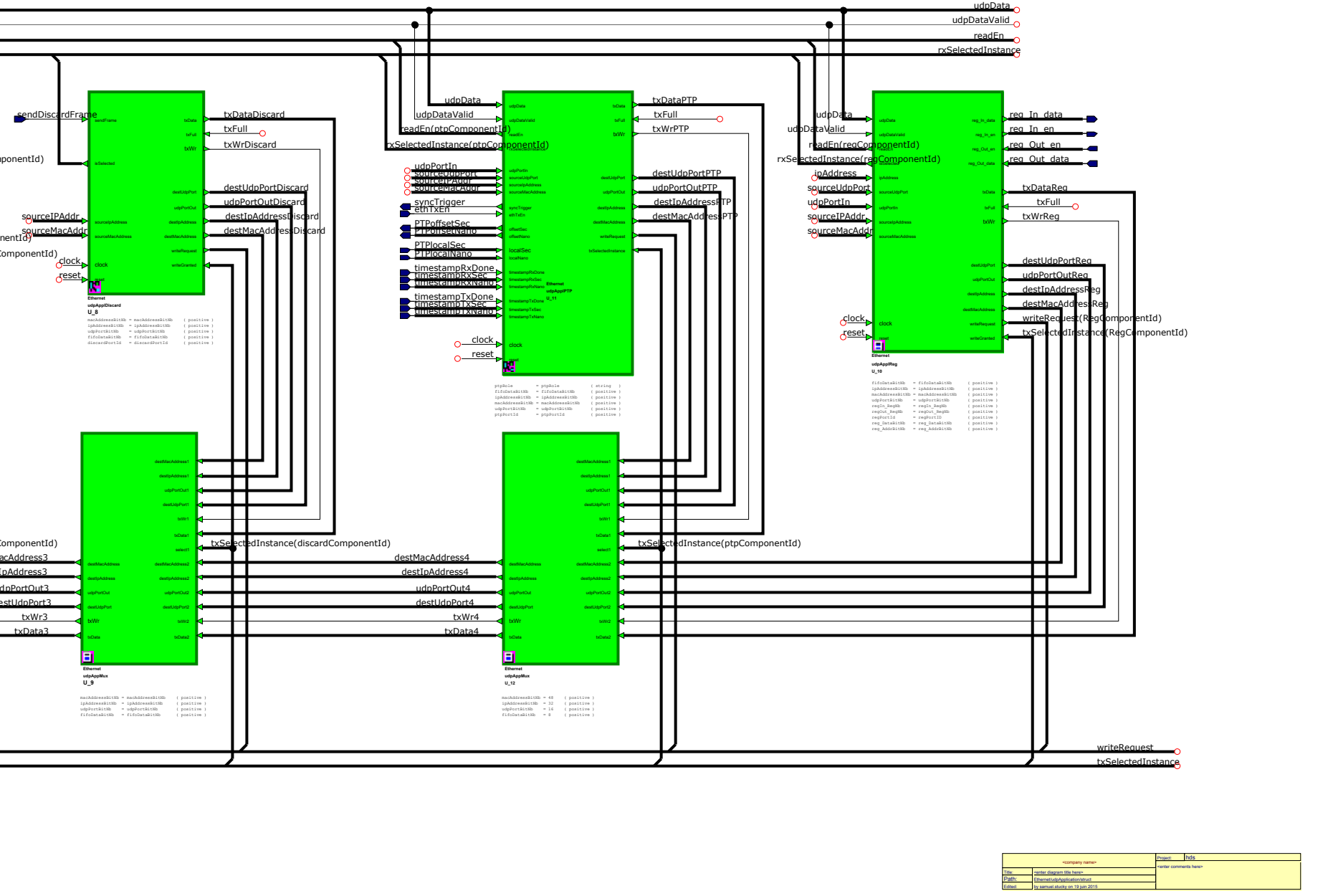
Pre User:

```
constant udpComponentNb : positive := 6;
constant echoComponentId : positive := 1;
constant echoPortId : positive := 07;
constant discardComponentId : positive := 2;
constant discardPortId : positive := 09;
constant quoteOfTheDayComponentId : positive := 3;
constant quoteOfTheDayPortId : positive := 17;
constant bonjourComponentId : positive := 4;
constant bonjourPortId : positive := 5353;
constant regComponentId : positive := 5;
constant regPortId : positive := regUDPPortId;
constant ptpComponentId : positive := 6;
constant ptpPortId : positive := 319;
```

Diagram Signals:

```
SIGNAL destIpAddress          : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddress1         : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddress2         : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddress3         : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddress4         : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddressBonjour   : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddressDiscard   : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddressPTP       : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddressQotd      : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destIpAddressReg       : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddress         : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddress1        : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddress2        : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddress3        : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddress4        : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddressBonjour  : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddressDiscard  : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddressPTP      : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddressQotd     : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddressReg      : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destUdpPort1           : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPort2           : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPort3           : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPort4           : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortBonjour     : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortDiscard     : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortEcho        : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortPTP         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortQotd        : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL destUdpPortReg         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL readEn                 : std_ulogic_vector(1 TO udpComponentNb)
SIGNAL rxSelectedInstance     : std_ulogic_vector(1 TO udpComponentNb)
SIGNAL txData1                : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txData2                : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txData3                : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txData4                : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataBonjour          : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataDiscard          : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataEcho             : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataPTP              : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataQotd             : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txDataReg              : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL txSelectedInstance     : std_ulogic_vector(1 TO udpComponentNb)
SIGNAL txWr1                  : std_ulogic
SIGNAL txWr2                  : std_ulogic
SIGNAL txWr3                  : std_ulogic
SIGNAL txWr4                  : std_ulogic
SIGNAL txWrBonjour            : std_ulogic
SIGNAL txWrDiscard            : std_ulogic
SIGNAL txWrEcho               : std_ulogic
SIGNAL txWrPTP                : std_ulogic
SIGNAL txWrQotd               : std_ulogic
SIGNAL txWrReg                : std_ulogic
SIGNAL udpData                : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
SIGNAL udpDataValid           : std_ulogic
SIGNAL udpPortOut1            : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOut2            : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOut3            : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOut4            : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutBonjour      : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutDiscard      : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutEcho         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutPTP          : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutQotd         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL udpPortOutReg          : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL writeRequest           : std_ulogic_vector(1 TO udpComponentNb)
```





<company name>		Project:	hds
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	Ethernet/udpApplication/struct		
Edited:	by samuel stucky on 19 juin 2015		

Package List

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 USE ieee.numeric_std.all;

LIBRARY Ethernet;
 USE Ethernet.ethernet.all;

LIBRARY PTP;
 USE PTP.config.all;
```

Declarations

Ports:

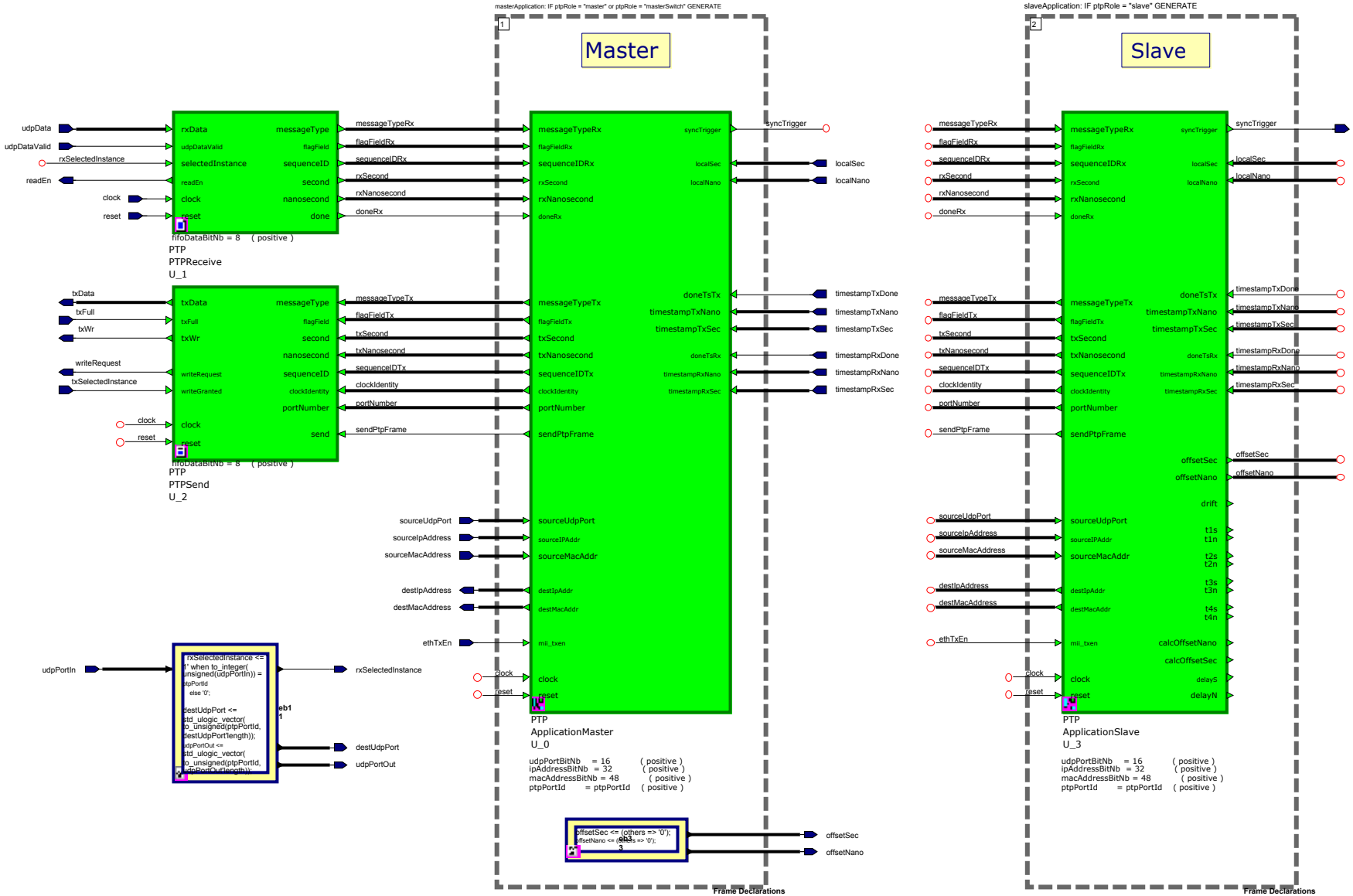
```
    clock              : std_ulogic
    ethTxEn            : std_ulogic
    localNano          : unsigned(UInt32BitNb-1 DOWNTO 0)
    localSec           : unsigned(UInt48BitNb-1 DOWNTO 0)
    reset              : std_ulogic
    sourceIpAddress    : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    sourceMacAddress   : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    sourceUdpPort      : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    timestampRxDone    : std_ulogic
    timestampRxNano    : unsigned(UInt32BitNb-1 DOWNTO 0)
    timestampRxSec     : unsigned(UInt48BitNb-1 DOWNTO 0)
    timestampTxDone    : std_ulogic
    timestampTxNano    : unsigned(UInt32BitNb-1 DOWNTO 0)
    timestampTxSec     : unsigned(UInt48BitNb-1 DOWNTO 0)
    txPull             : std_ulogic
    txSelectedInstance : std_ulogic
    udpData            : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
    udpDataValid       : std_ulogic
    udpPortIn          : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    destIpAddress      : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    destMacAddress     : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    destUdpPort        : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    offsetNano         : signed(UInt32BitNb-1 DOWNTO 0)
    offsetSec          : signed(UInt48BitNb-1 DOWNTO 0)
    readEn             : std_ulogic
    rxSelectedInstance : std_ulogic
    syncTrigger        : std_ulogic
    txData             : std_ulogic_vector(fifoDataBitNb-1 DOWNTO 0)
    txWr               : std_ulogic
    udpPortOut         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    writeRequest       : std_ulogic
```

Pre User:

```
    constant baudRate: real := 115200.0;
    constant clockFrequency: real := 66.0E6;
```

Diagram Signals:

```
    SIGNAL clockIdentity      : std_ulogic_vector(79 DOWNTO 0)
    SIGNAL doneRx             : std_ulogic
    SIGNAL flagFieldRx        : std_ulogic_vector(15 DOWNTO 0)
    SIGNAL flagFieldTx        : std_ulogic_vector(15 DOWNTO 0)
    SIGNAL messageTypeRx      : std_ulogic_vector(3 DOWNTO 0)
    SIGNAL messageTypeTx      : std_ulogic_vector(3 DOWNTO 0)
    SIGNAL portNumber         : std_ulogic_vector(15 DOWNTO 0)
    SIGNAL rxNanosecond       : unsigned(UInt32BitNb-1 DOWNTO 0)
    SIGNAL rxSecond           : unsigned(UInt48BitNb-1 DOWNTO 0)
    SIGNAL sendPtpFrame       : std_ulogic
    SIGNAL sequenceIDRx       : std_ulogic_vector(15 DOWNTO 0)
    SIGNAL sequenceIDTx       : std_ulogic_vector(15 DOWNTO 0)
    SIGNAL txNanosecond       : unsigned(UInt32BitNb-1 DOWNTO 0)
    SIGNAL txSecond           : unsigned(UInt48BitNb-1 DOWNTO 0)
```



<company name>	Project: Ihds
	<enter comments here>
Title: <enter diagram title here>	
Path: Ethernet/udpApplPTP/struct	
Edited: by samuel.stucky on 22 juin 2015	

Package List

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

Declarations

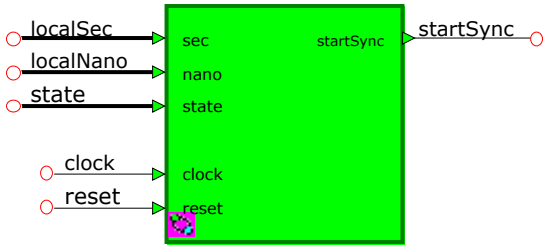
Ports:

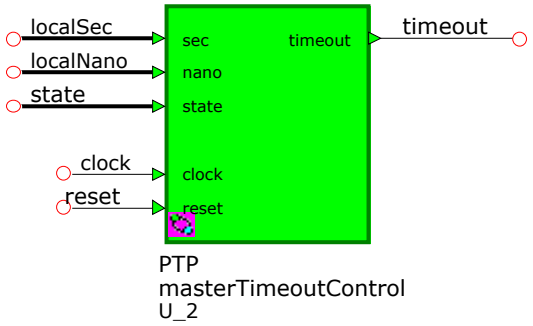
```
    clock           : std_ulogic
    doneRx          : std_ulogic
    doneTsRx        : std_ulogic
    doneTsTx        : std_ulogic
    flagFieldRx     : std_ulogic_vector(15 DOWNTO 0)
    localNano       : unsigned(UInt32BitNb-1 DOWNTO 0)
    localSec        : unsigned(UInt48BitNb-1 DOWNTO 0)
    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)
    destIpAddr      : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    destMacAddr     : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    flagFieldTx     : std_ulogic_vector(15 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)
```

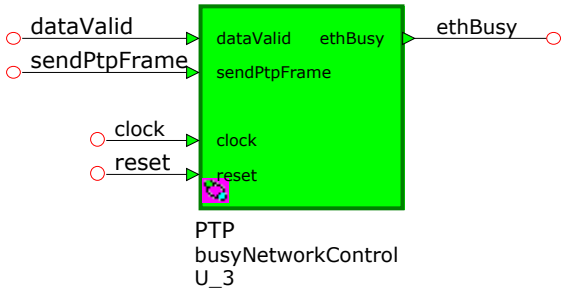
Pre User:

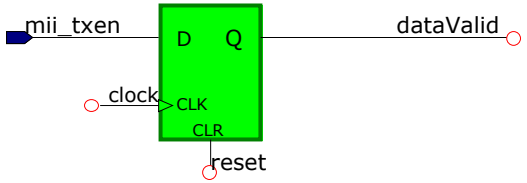
Diagram Signals:

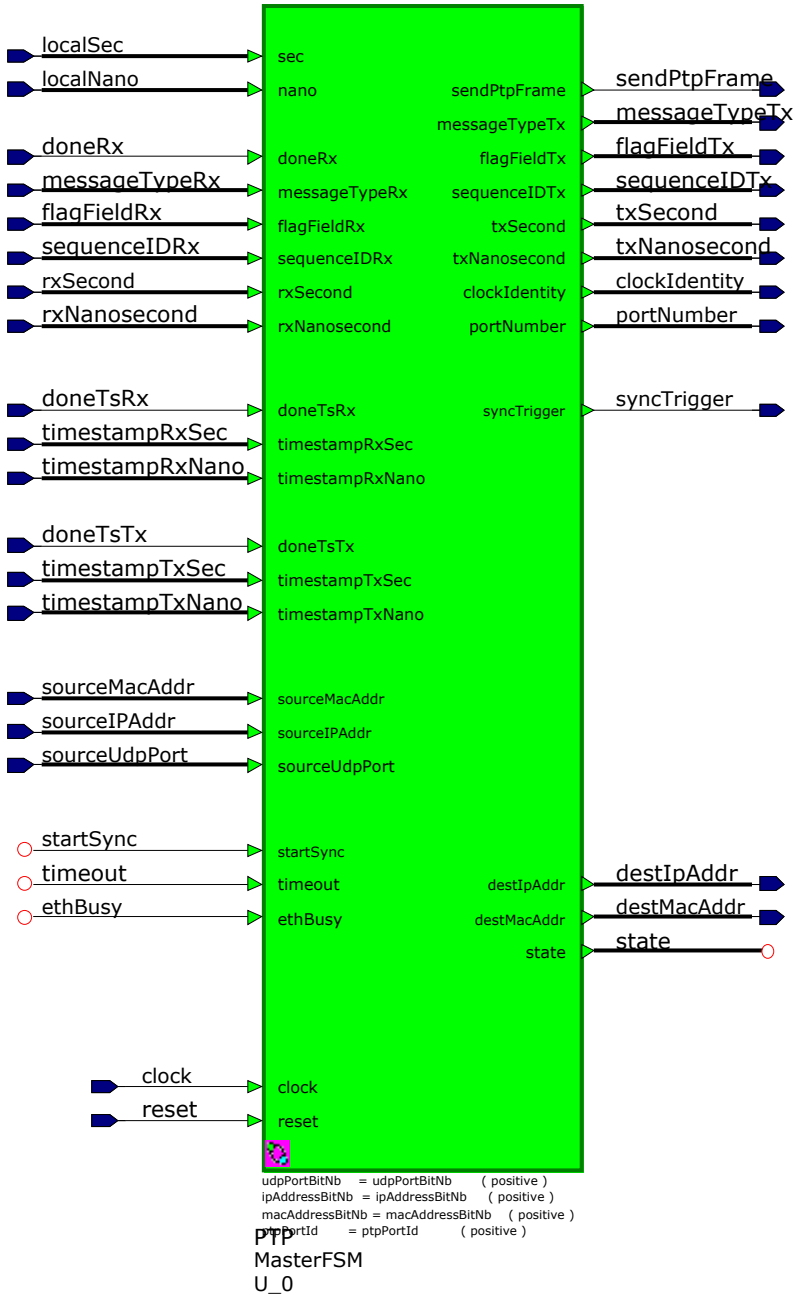
```
    SIGNAL dataValid       : std_uLogic
    SIGNAL ethBusy         : std_ulogic
    SIGNAL startSync       : std_ulogic
    SIGNAL state           : std_ulogic_vector(3 DOWNTO 0)
    SIGNAL timeout         : std_ulogic
```











HES-SO Valais		Project:
Title:	Application Maître	Base de temps pour systèmes distribués
Path:	PTP/ApplicationMaster/struct	
Edited:	by samuel.stucky on 22 juin 2015	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements
sendPtpFrame <= sendPTP;

Architecture Declarations

Signal Status

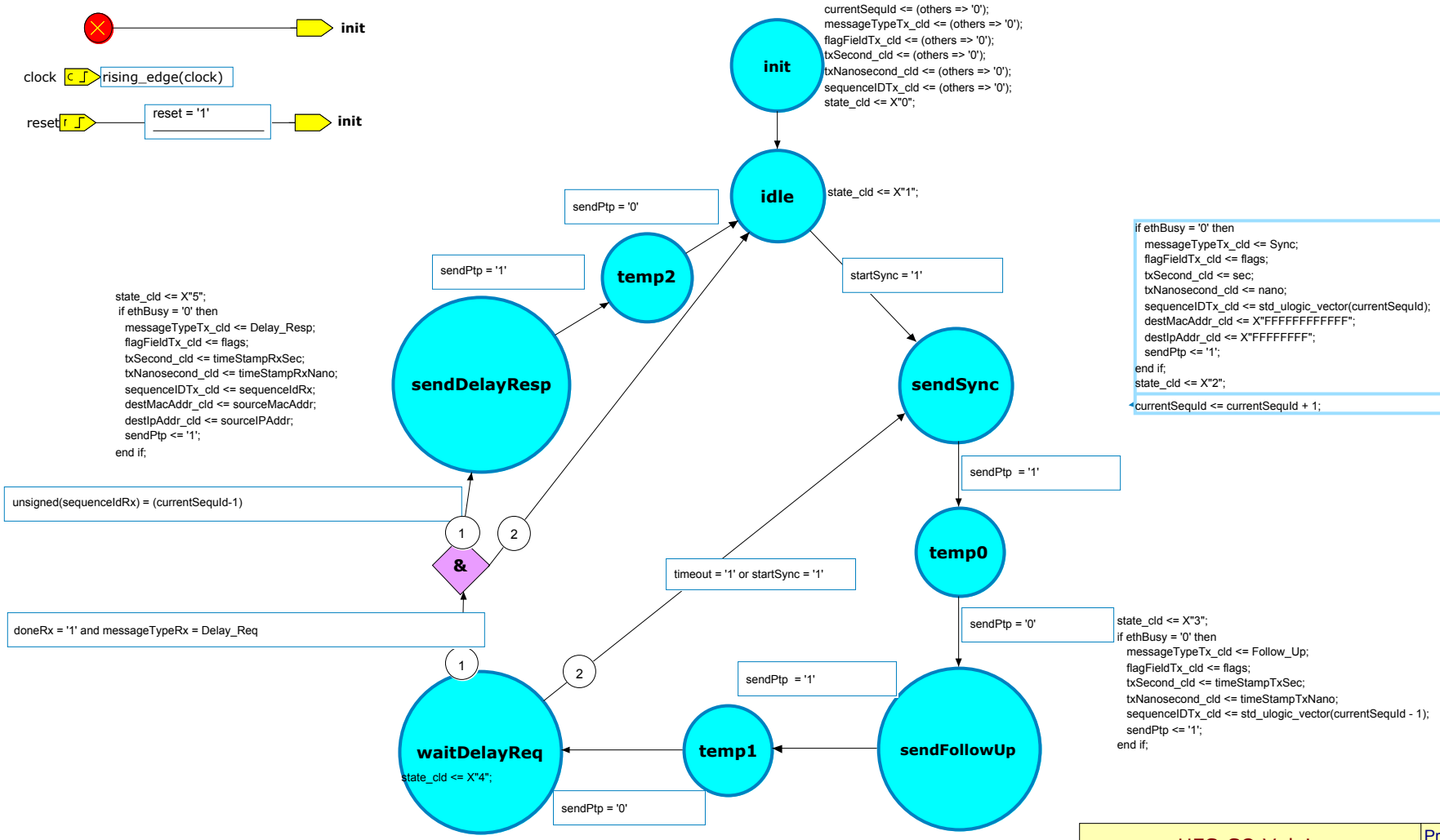
SIGNAL	MODE	DEFAULT	RESET	SCHEME
clockIdentity	OUT	(others => '0')		COMB
state	OUT	(others => '0')	(others => '0')	CLKD
destIpAddr	OUT		X"FFFFFFFF"	CLKD
destMacAddr	OUT		X"FFFFFFFFFFFF"	CLKD
flagFieldTx	OUT		(others => '0')	CLKD
messageTypeTx	OUT		(others => '0')	CLKD
portNumber	OUT	(others => '0')		COMB
sequenceIDTx	OUT		(others => '0')	CLKD
syncTrigger	OUT	'0'	'0'	CLKD
txNanosecond	OUT		(others => '0')	CLKD
txSecond	OUT		(others => '0')	CLKD
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:

Package List

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



HES-SO Valais		Project:
Title:	Application maître FSM	Base de temps pour systèmes distribués
Path:	PTP/MasterFSM/fsm	
Edited:	by samuel.stucky on 22 juin 2015	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

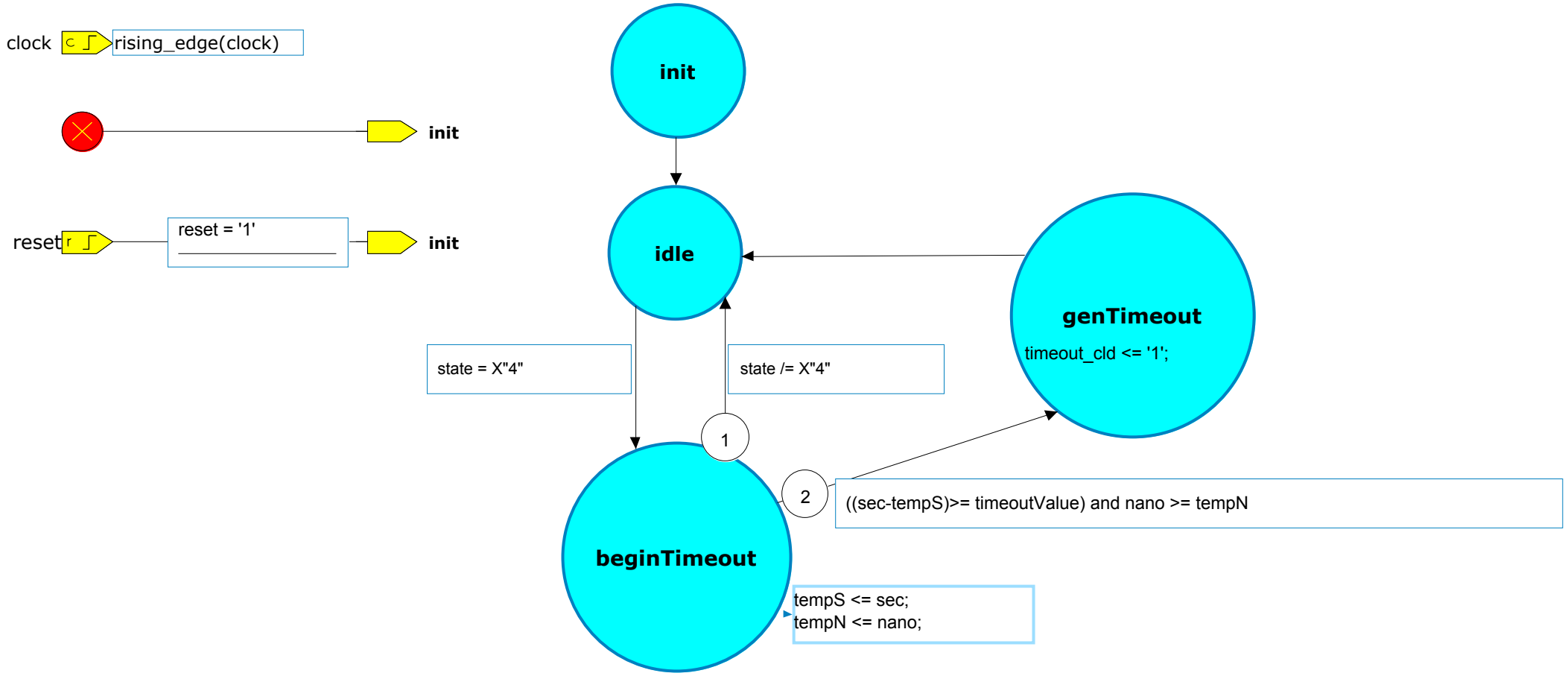
SIGNAL	MODE	DEFAULT	RESET	SCHEME
timeout	OUT	'0'	'0'	CLKD
tempS	LOCAL		(others => '0')	CLKD
tempN	LOCAL		(others => '0')	CLKD
timeoutValue	LOCAL	X"03"	X"03"	CLKD

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 PTP;
  USE PTP.config.all;
```



HES-SO Valais		Project:
Title:	Timeout FSM	Base de temps pour systèmes distribués
Path:	PTP/masterTimeoutControl/fsm	
Edited:	by samuel.stucky on 15 juin 2015	

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 PTP;
  USE PTP.config.all;
```

Signal Status

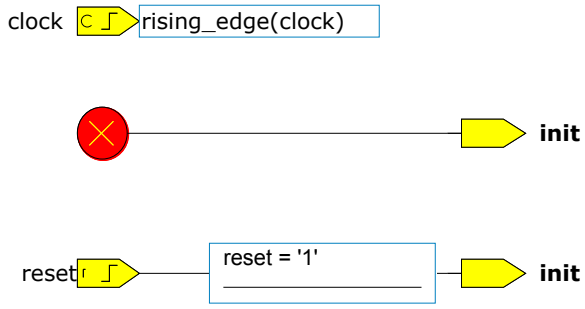
SIGNAL	MODE	DEFAULT	RESET	SCHEME
startSync	OUT	'0'	'0'	CLKD
tempSec	LOCAL		(others => '0')	CLKD
tempNano	LOCAL		(others => '0')	CLKD
syncInterval	LOCAL	X"02"	X"02"	CLKD

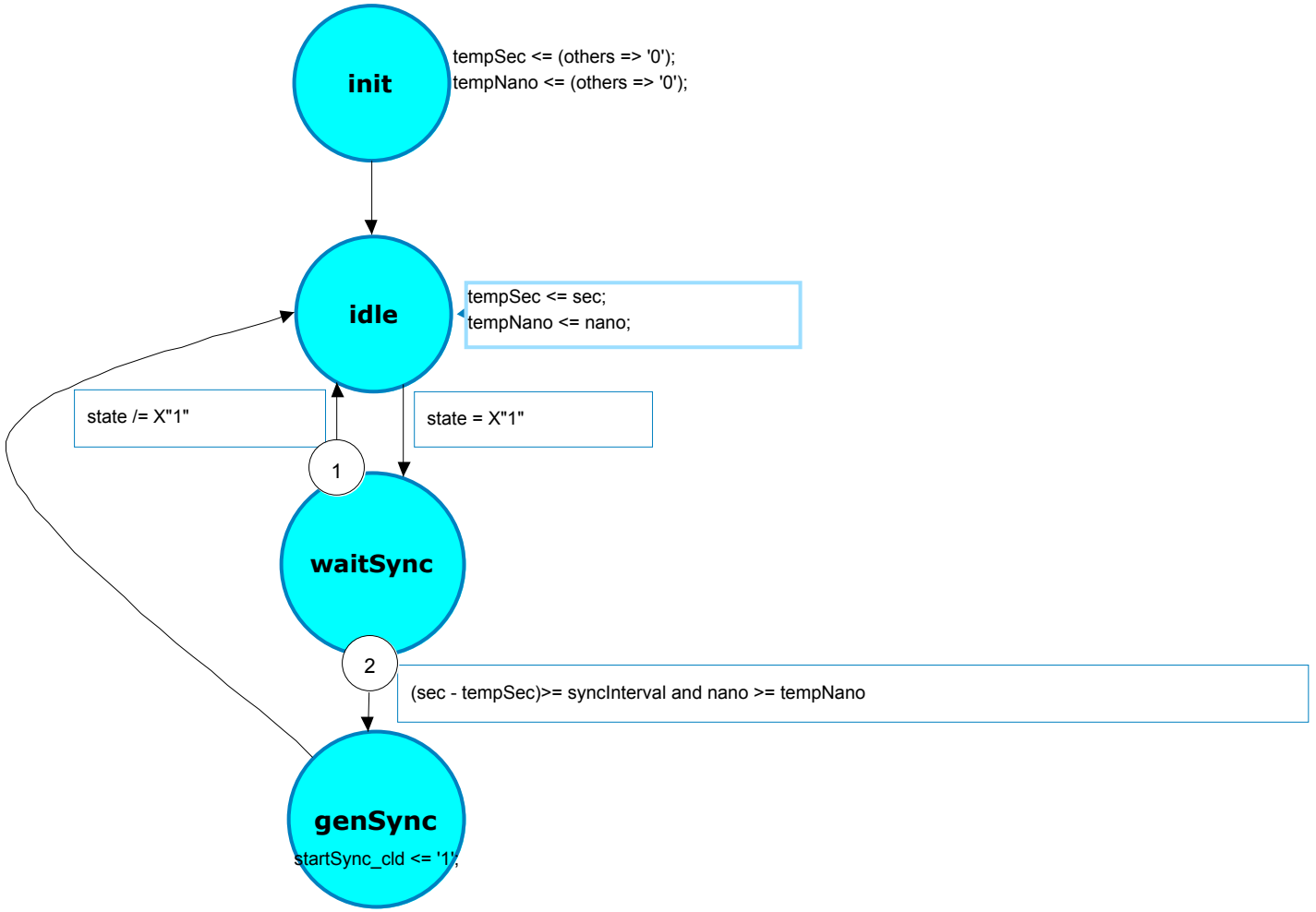
State Register Statements

Process Declarations

Clocked Process:

Output Process:





HES-SO Valais		Project:
Title:	Synchronisation FSM	Base de temps pour systèmes distribués
Path:	PTP/syncControl/fsm	
Edited:	by samuel.stucky on 15 juin 2015	

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 PTP;
  USE PTP.config.all;
```

Signal Status

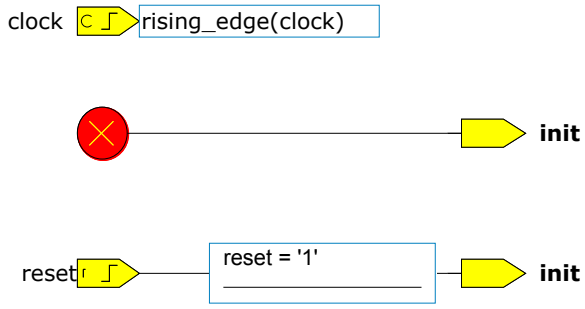
SIGNAL	MODE	DEFAULT	RESET	SCHEME
startSync	OUT	'0'	'0'	CLKD
tempSec	LOCAL		(others => '0')	CLKD
tempNano	LOCAL		(others => '0')	CLKD
syncInterval	LOCAL	X"02"	X"02"	CLKD

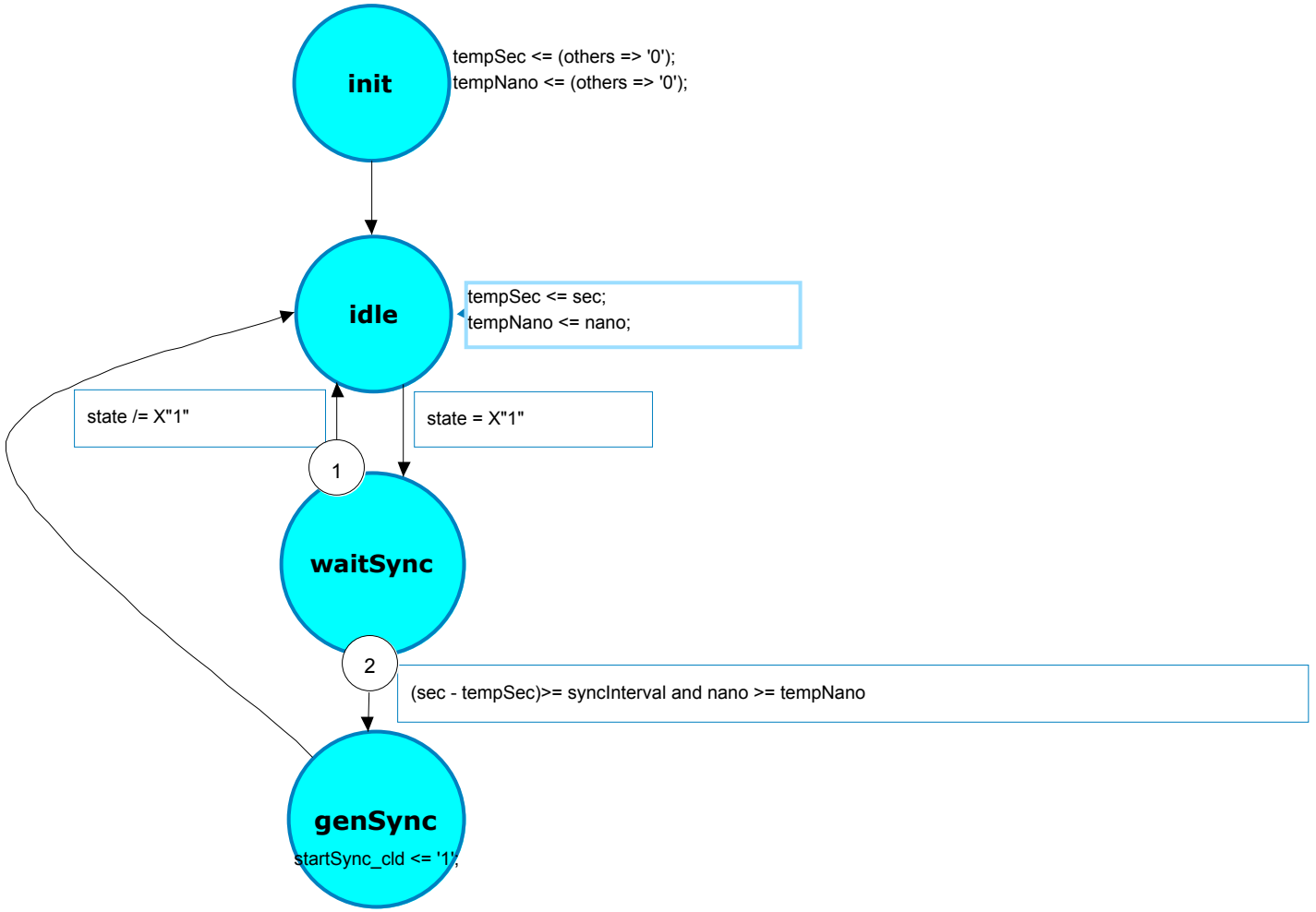
State Register Statements

Process Declarations

Clocked Process:

Output Process:





HES-SO Valais		Project:
Title:	Synchronisation FSM	Base de temps pour systèmes distribués
Path:	PTP/syncControl/fsm	
Edited:	by samuel.stucky on 15 juin 2015	

Package List

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

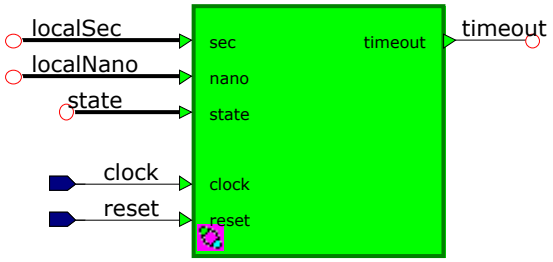
Declarations

Ports:

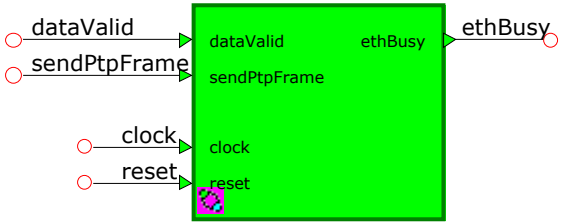
```
clock           : std_ulogic
doneRx          : std_ulogic
doneTsRx        : std_ulogic
doneTsTx        : std_ulogic
flagFieldRx     : std_ulogic_vector(15 DOWNTO 0)
localNano       : unsigned(UInt32BitNb-1 DOWNTO 0)
localSec        : unsigned(UInt48BitNb-1 DOWNTO 0)
messageTypeRx   : std_ulogic_vector(3 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)
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)
drift           : signed(UInt32BitNb-1 DOWNTO 0)
flagFieldTx     : std_ulogic_vector(15 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)
```

Diagram Signals:

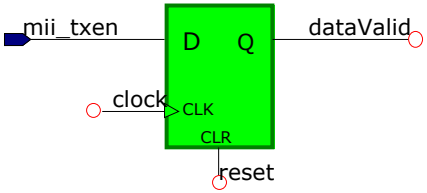
```
SIGNAL dataValid       : std_uLogic
SIGNAL ethBusy         : std_ulogic
SIGNAL state           : std_ulogic_vector(3 DOWNTO 0)
SIGNAL timeout         : std_ulogic
```

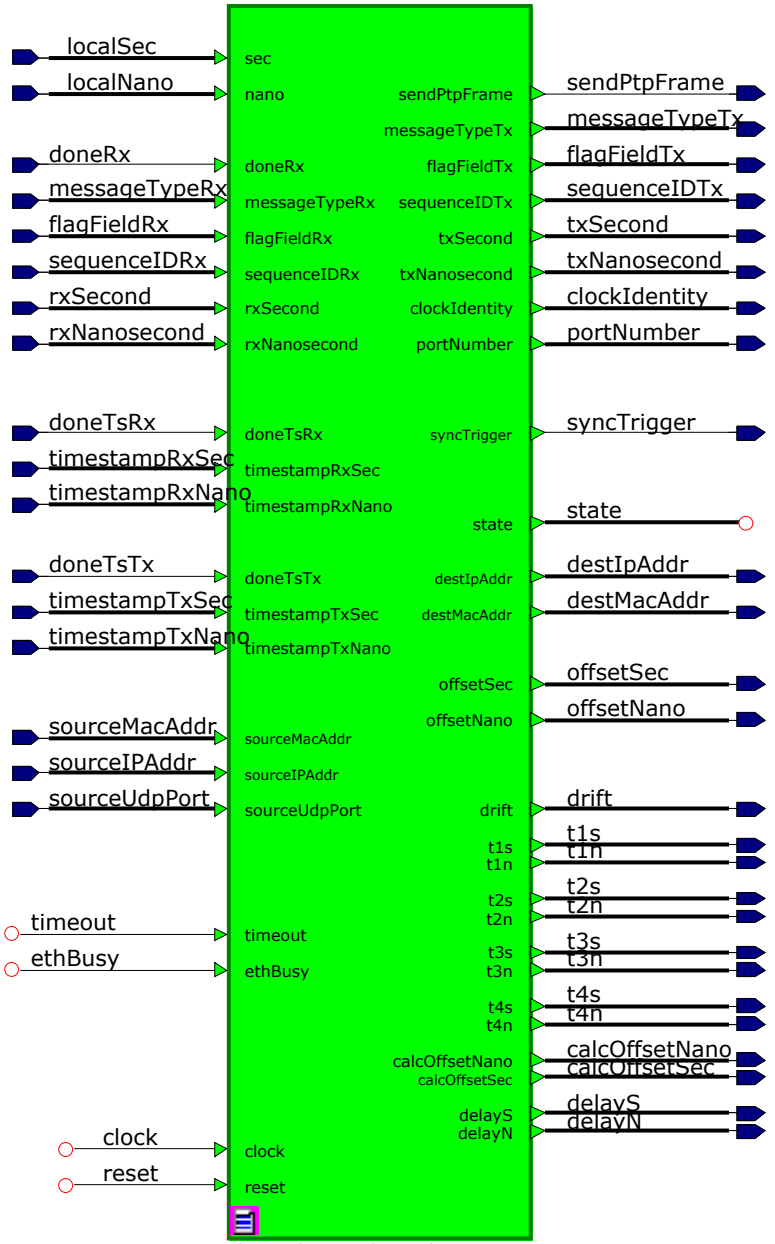


PTP
slaveTimeoutControl
U_2



PTP
busyNetworkControl
U_3





udpPortBitNb = 16 (positive)
ipAddressBitNb = 32 (positive)
macAddressBitNb = 48 (positive)

PTP
SlaveTxtFSM
U_1

HES-SO Valais		Project:
Title:	Application Esclave	Base de temps pour systèmes distribués
Path:	PTP/ApplicationSlave/struct	
Edited:	by samuel.stucky on 19 juin 2015	

```
--
-- VHDL Architecture TDSlave.SlaveTxtFSM.rtl
--
-- Created:
--          by - drozj1.UNKNOWN (WE4136)
--          at - 12:48:11 01.07.2012
--
-- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
--
-- Cette machine d'état a été générée a partir du bloc SlaveFSM
-- du projet TDSlave. Le texte généré a été copié dans ce composant
-- de manière a pouvoir modifier le code généré.

ARCHITECTURE rtl OF SlaveTxtFSM IS
   -- Architecture Declarations
   SIGNAL currentSequId : unsigned(SequenceIdRx'range);
   SIGNAL flags : std_ulogic_vector(flagFieldRx'range);
   SIGNAL oldT1 : Timestamp;
   SIGNAL oldT2 : Timestamp;
   SIGNAL sendPtp : std_ulogic;
   SIGNAL t1 : Timestamp;
   SIGNAL t2 : Timestamp;
   SIGNAL t3 : Timestamp;
   SIGNAL t4 : Timestamp;
   SIGNAL tempT1 : Timestamp;
   SIGNAL tempT2 : Timestamp;
   SIGNAL tempT3 : Timestamp;

   TYPE STATE_TYPE IS (
      init,
      waitSync,
      waitFollowUp,
      sendDelayReq,
      waitDelayResp,
      compute1,
      compute2,
      compute3,
      updateClock,
      temp0
   );

   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;

   -- Declare any pre-registered internal signals
   SIGNAL flagFieldTx_cld : std_ulogic_vector (15 DOWNTO 0);
   SIGNAL messageTypeTx_cld : std_ulogic_vector (3 DOWNTO 0);
   SIGNAL sequenceIDTx_cld : std_ulogic_vector (15 DOWNTO 0);
   SIGNAL state_cld : std_ulogic_vector (3 DOWNTO 0);
   SIGNAL syncTrigger_cld : std_ulogic ;
   SIGNAL txNanosecond_cld : unsigned (UInt32BitNb-1 DOWNTO 0);
   SIGNAL txSecond_cld : unsigned (UInt48BitNb-1 DOWNTO 0);

BEGIN

   ----------------------------------------------------------------
   clocked_proc : PROCESS (
      clock,
```

```
60          reset
61      )
62      constant margeBit : positive := 4;
63           variable tempDelayS: signed(47+margeBit downto 0);
64           variable tempDelayN: signed(31+margeBit downto 0);
65           variable offsetSecCalc: signed(47+margeBit downto 0);
66           variable offsetNanoCalc: signed(31+margeBit downto 0);
67      ------------------------------------------------------------------
68      BEGIN
69         IF (reset = '1') THEN
70            current_state <= init;
71            -- Default Reset Values
72            flagFieldTx_cld <= (others => '0');
73            messageTypeTx_cld <= (others => '0');
74            sequenceIDTx_cld <= (others => '0');
75            state_cld <= (others => '0');
76            syncTrigger_cld <= '0';
77            txNanosecond_cld <= (others => '0');
78            txSecond_cld <= (others => '0');
79            currentSequId <= (others => '0');
80            oldT1 <= ((others => '0'),(others => '0'));
81            oldT2 <= ((others => '0'),(others => '0'));
82            sendPtp <= '0';
83            t1 <= ((others => '0'),(others => '0'));
84            t2 <= ((others => '0'),(others => '0'));
85            t3 <= ((others => '0'),(others => '0'));
86            t4 <= ((others => '0'),(others => '0'));
87            tempT1 <= ((others => '0'),(others => '0'));
88            tempT2 <= ((others => '0'),(others => '0'));
89            tempT3 <= ((others => '0'),(others => '0'));
90         ELSIF (rising_edge(clock)) THEN
91            current_state <= next_state;
92            -- Default Assignment To Internals
93            sendPtp <= '0';
94            state_cld <= (others => '0');
95            syncTrigger_cld <= '0';
96
97            -- Combined Actions
98            CASE current_state IS
99               WHEN init =>
100                 state_cld <= X"0";
101                 messageTypeTx_cld <= (others => '0');
102                 flagFieldTx_cld <= (others => '0');
103                 txSecond_cld <= (others => '0');
104                 txNanosecond_cld <= (others => '0');
105                 sequenceIDTx_cld <= (others => '0');
106              WHEN waitSync =>
107                 state_cld <= X"1";
108                 IF (doneRx = '1' and messageTypeRx = Sync) THEN
109                    currentSequId <= unsigned(sequenceIdRx);
110                    tempT2.secondsField <= timestampRxSec;
111                    tempT2.nanosecondsField <= timestampRxNano;
112                 END IF;
113              WHEN waitFollowUp =>
114                 state_cld <= X"2";
115                 IF ((doneRx = '1' and messageTypeRx = Follow_Up) AND (unsigned(
                    sequenceIdRx) = currentSequId)) THEN
116                    tempT1.secondsField <= rxSecond;
117                    tempT1.nanosecondsField <= rxNanosecond;
```

```
118                ELSIF (doneRx = '1' and messageTypeRx = Follow_Up) THEN
119                END IF;
120             WHEN sendDelayReq =>
121                state_cld <= X"3";
122                if ethBusy = '0' then
123                   messageTypeTx_cld <= Delay_Req;
124                   flagFieldTx_cld <= flags;
125                   txSecond_cld <= sec;
126                   txNanosecond_cld <= nano;
127                   sequenceIDTx_cld <= std_ulogic_vector(currentSequId);
128                   sendPtp    <= '1';
129                end if;
130             WHEN waitDelayResp =>
131                state_cld <= X"4";
132                if doneTsTx = '1' then
133                   tempT3.secondsField <= timestampTxSec;
134                   tempT3.nanosecondsField <= timestampTxNano;
135                end if;
136                IF ((doneRx = '1' and messageTypeRx = Delay_Resp) AND (unsigned(
                   sequenceIdRx) = currentSequId)) THEN
137                   t1 <= tempT1;
138                   t2 <= tempT2;
139                   t3 <= tempT3;
140                   t4.secondsField <= rxSecond;
141                   t4.nanosecondsField <= rxNanosecond;
142                   oldT1 <= t1;
143                   oldT2 <= t2;
144                ELSIF (doneRx = '1' and messageTypeRx = Delay_Resp) THEN
145                END IF;
146             WHEN compute1 =>
147
148             -- assign timestamp outputs
149
150             t1s <= t1.secondsField;
151             t1n <= t1.nanosecondsField;
152             t2s <= t2.secondsField;
153             t2n <= t2.nanosecondsField;
154             t3s <= t3.secondsField;
155             t3n <= t3.nanosecondsField;
156             t4s <= t4.secondsField;
157             t4n <= t4.nanosecondsField;
158
159             --
160
161                state_cld <= X"5";
162

163   --              tempDelayS :=       signed(     resize(t2.secondsField,
      t2.secondsField'length + margeBit)
164   --                                                                    -
      resize(t1.secondsField, t1.secondsField'length + margeBit)
165   --                                                                    +
      resize(t4.secondsField, t4.secondsField'length + margeBit)
166   --                                                                    -
      resize(t3.secondsField, t3.secondsField'length + margeBit)
167   --                                                                 );
168   --
169   --              tempDelayN :=       signed(     resize(t2.nanosecondsField,
      t2.nanosecondsField'length + margeBit)
```

```
170   --                                                                -
      resize(t1.nanosecondsField, t1.nanosecondsField'length + margeBit)
171   --                                                               +
      resize(t4.nanosecondsField, t4.nanosecondsField'length + margeBit)
172   --                                                               -
      resize(t3.nanosecondsField, t3.nanosecondsField'length + margeBit)
173   --                                                                  );
174   --              if tempDelayN < 0 then
175   --                  tempDelayN := tempDelayN + 1e9;
176   --                  tempDelayS := tempDelayS - 1;
177   --              elsif tempDelayN >= 1e9 then                -- >= or > ??
178   --                  tempDelayN := tempDelayN - 1e9;
179   --                  tempDelayS := tempDelayS + 1;
180   --              end if;
181   --
182   --              if tempDelayS(0) = '1' then
183   --                  tempdelayN := tempdelayN + 1e9;
184   --              end if;
185   --
186   --              tempDelayS := shift_right(tempDelayS,1);
187   --              tempDelayN := shift_right(tempDelayN,1);
188   --
189   --              delayS <= tempDelayS;
190   --              delayN <= tempDelayN;
191
192                                                   offsetSecCalc :=       signed(
                                                          resize(t2.secondsField, t2.
                                                      secondsField'length + margeBit)
193                                                               - resize(t1.
                                                               secondsField, t1.
                                                               secondsField'
                                                               length + margeBit
                                                               ));
194
195                                                   offsetNanoCalc :=      signed(
                                                          resize(t2.nanosecondsField,
                                                      t2.nanosecondsField'length +
                                                      margeBit)
196                                                               - resize(t1.
                                                               nanosecondsField,
                                                                t1.
                                                               nanosecondsField'
                                                               length + margeBit
                                                               ));
197
198               WHEN compute2 =>
199                                                   offsetSecCalc := offsetSecCalc -
                                                      signed(resize(t4.secondsField, t4
                                                      .secondsField'length + margeBit));
200                                                   offsetNanoCalc := offsetNanoCalc
                                                      - signed(resize(t4.
                                                      nanosecondsField, t4.
                                                      nanosecondsField'length +
                                                      margeBit));
201               WHEN compute3 =>                    offsetSecCalc := offsetSecCalc +
                  signed(resize(t3.secondsField, t3.secondsField'length + margeBit));
202                                                   offsetNanoCalc := offsetNanoCalc
                                                      + signed(resize(t3.
```

```
                                                    nanosecondsField, t3.
                                                    nanosecondsField'length +
                                                    margeBit));
203
204                 -- if nanos are less then 0 or greater then 1e9 adjust them and
                    adjust secs too
205
206                 if offsetNanoCalc < 0 then
207                     offsetNanoCalc := offsetNanoCalc + 1e9;
208                     offsetSecCalc := offsetSecCalc - 1;
209                 elsif offsetNanoCalc > 1e9 then
210                     offsetNanoCalc := offsetNanoCalc - 1e9;
211                     offsetSecCalc := offsetSecCalc + 1;
212                 end if;
213
214                 -- if sec are odd add 1e9 to ensure the division by 2 works
215
216                 if offsetSecCalc(0) = '1' then
217                     offsetNanoCalc := offsetNanoCalc + 1e9;
218                 end if;
219
220                 -- divide by 2
221
222                 offsetNanoCalc := shift_right(offsetNanoCalc,1);
223                 offsetSecCalc := shift_right(offsetSecCalc,1);
224
225                 offsetSec <= resize(offsetSecCalc, offsetSec'length);
226                 offsetNano <= resize(offsetNanoCalc, offsetNano'length);
227
228           calcOffsetSec <= resize(offsetSecCalc, calcOffsetSec'length);
229           calcOffsetNano <= resize(offsetNanoCalc, calcOffsetNano'length);
230
231
232               WHEN updateClock =>
233                  state_cld <= X"6";
234                  syncTrigger_cld <= '1';
235               WHEN OTHERS =>
236                  NULL;
237           END CASE;
238       END IF;
239    END PROCESS clocked_proc;
240
241    ------------------------------------------------------------------
242    nextstate_proc : PROCESS (
243       currentSequId,
244       current_state,
245       doneRx,
246       messageTypeRx,
247       sendPtp,
248       sequenceIDRx,
249       timeout
250    )
251    ------------------------------------------------------------------
252    BEGIN
253       CASE current_state IS
254          WHEN init =>
255             next_state <= waitSync;
256          WHEN waitSync =>
257             IF (doneRx = '1' and messageTypeRx = Sync) THEN
```

```
258                next_state <= waitFollowUp;
259             ELSE
260                next_state <= waitSync;
261             END IF;
262          WHEN waitFollowUp =>
263             IF ((doneRx = '1' and messageTypeRx = Follow_Up) AND (unsigned(
                sequenceIdRx) = currentSequId)) THEN
264                next_state <= sendDelayReq;
265             ELSIF (doneRx = '1' and messageTypeRx = Follow_Up) THEN
266                next_state <= waitSync;
267             ELSIF (timeout = '1') THEN
268                next_state <= waitSync;
269             ELSE
270                next_state <= waitFollowUp;
271             END IF;
272          WHEN sendDelayReq =>
273             IF (sendPtp = '1') THEN
274                next_state <= temp0;
275             ELSE
276                next_state <= sendDelayReq;
277             END IF;
278          WHEN waitDelayResp =>
279             IF ((doneRx = '1' and messageTypeRx = Delay_Resp) AND (unsigned(
                sequenceIdRx) = currentSequId)) THEN
280                next_state <= compute1;
281             ELSIF (doneRx = '1' and messageTypeRx = Delay_Resp) THEN
282                next_state <= waitSync;
283             ELSIF (timeout = '1') THEN
284                next_state <= waitSync;
285             ELSE
286                next_state <= waitDelayResp;
287             END IF;
288          WHEN compute1 =>
289             next_state <= compute2;
290          WHEN compute2 =>
291             next_state <= compute3;
292          WHEN compute3 =>
293             next_state <= updateClock;
294          WHEN updateClock =>
295             next_state <= waitSync;
296          WHEN temp0 =>
297             IF (sendPtp = '0') THEN
298                next_state <= waitDelayResp;
299             ELSE
300                next_state <= temp0;
301             END IF;
302          WHEN OTHERS =>
303             next_state <= init;
304       END CASE;
305    END PROCESS nextstate_proc;
306
307    ------------------------------------------------------------------
308    -- Default Assignment
309    clockIdentity <= (others => '0');
310    portNumber <= (others => '0');
311    -- Default Assignment To Internals
312    flags <= X"0200";
313
314    -- Concurrent Statements
```

```
315      -- Clocked output assignments
316      flagFieldTx <= flagFieldTx_cld;
317      messageTypeTx <= messageTypeTx_cld;
318      sequenceIDTx <= sequenceIDTx_cld;
319      state <= state_cld;
320      syncTrigger <= syncTrigger_cld;
321      txNanosecond <= txNanosecond_cld;
322      txSecond <= txSecond_cld;
323      sendPtpFrame <= sendPTP;
324      destIpAddr <= sourceIPAddr;
325      destMacAddr <= sourceMacAddr;
326
327  END ARCHITECTURE rtl;
328
329
```

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:

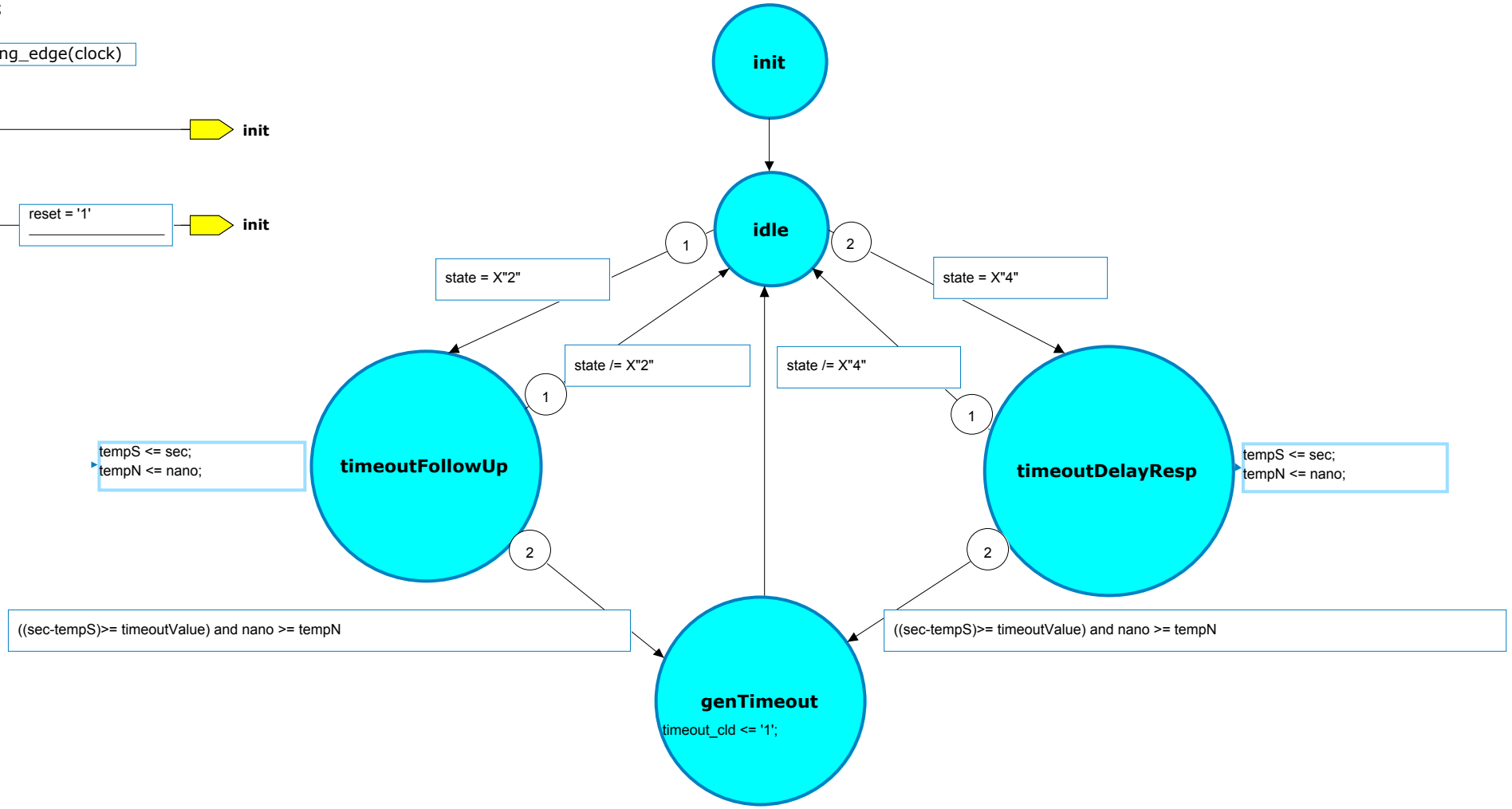
Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;
LIBRARY PTP;
  USE PTP.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:	PTP/slaveTimeoutControl/fsm	
Edited:	by samuel.stucky on 15 juin 2015	

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

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

Signal Status

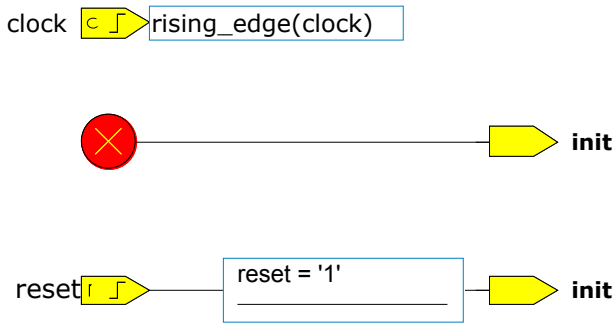
SIGNAL	MODE	DEFAULT	RESET	SCHEME
ethBusy	OUT	'1'	'1'	CLKD
counter	LOCAL		(others => '0')	CLKD

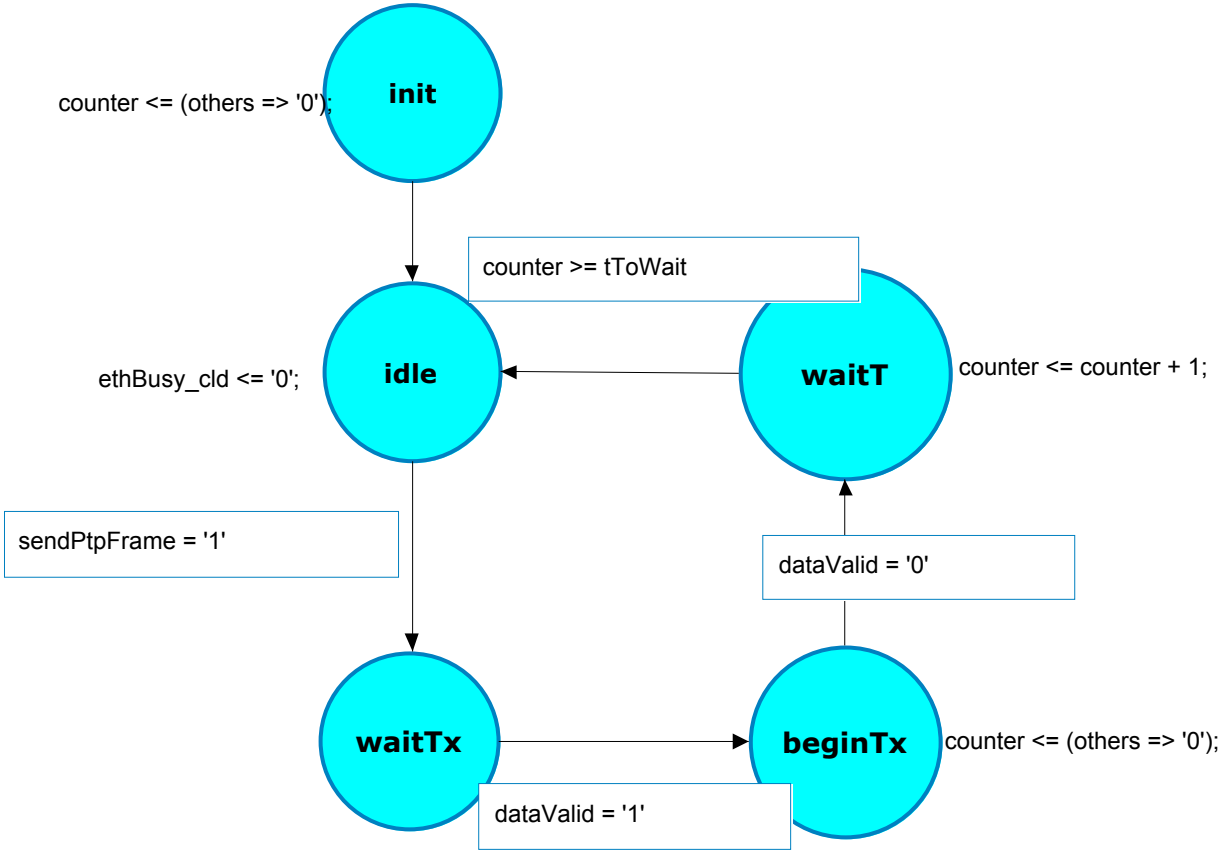
State Register Statements

Process Declarations
Clocked Process:
Output Process:

Package List

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





HES-SO Valais		Project:
Title:	Network Control FSM	Base de temps pour systèmes distribués
Path:	PTP/busyNetworkControl/fsm	
Edited:	by samuel.stucky on 22 juin 2015	

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY Ethernet;
USE Ethernet.ethernet.ALL;
LIBRARY PTP;
 USE PTP.config.all;
```

Declarations

Ports:

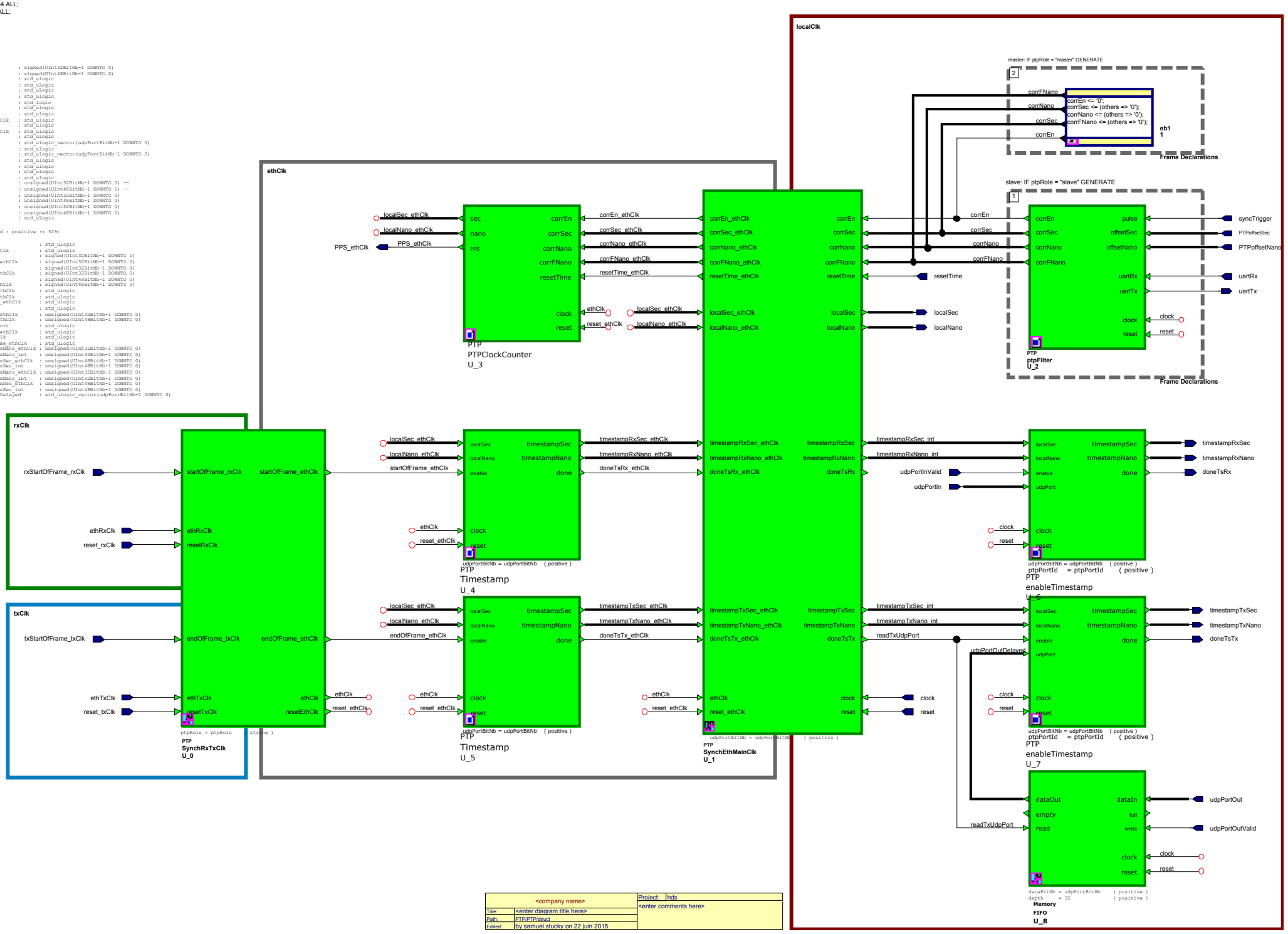
```
PTPoffsetNano          : signed(UInt32BitNb-1 DOWNTO 0)
PTPoffsetSec           : signed(UInt48BitNb-1 DOWNTO 0)
clock                  : std_ulogic
ethRxClk               : std_ulogic
ethTxClk               : std_ulogic
reset                  : std_ulogic
resetTime              : std_logic
reset_rxClk            : std_ulogic
reset_txClk            : std_ulogic
rxStartOfFrame_rxClk   : std_ulogic
syncTrigger            : std_ulogic
txStartOfFrame_txClk   : std_ulogic
uartRx                 : std_ulogic
udpPortIn              : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
udpPortInValid         : std_ulogic
udpPortOut             : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
udpPortOutValid        : std_ulogic
PPS_ethClk             : std_ulogic
doneTsRx               : std_ulogic
doneTsTx               : std_ulogic
localNano              : unsigned(UInt32BitNb-1 DOWNTO 0) --
localSec               : unsigned(UInt48BitNb-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)
uartTx                 : std_uLogic
```

Pre User:

```
constant ptpPortId : positive := 319;
```

Diagram Signals:

```
SIGNAL corrEn                : std_ulogic
SIGNAL corrEn_ethClk         : std_ulogic
SIGNAL corrFNano             : signed(UInt32BitNb-1 DOWNTO 0)
SIGNAL corrFNano_ethClk      : signed(UInt32BitNb-1 DOWNTO 0)
SIGNAL corrNano              : signed(UInt32BitNb-1 DOWNTO 0)
SIGNAL corrNano_ethClk       : signed(UInt32BitNb-1 DOWNTO 0)
SIGNAL corrSec               : signed(UInt48BitNb-1 DOWNTO 0)
SIGNAL corrSec_ethClk        : signed(UInt48BitNb-1 DOWNTO 0)
SIGNAL doneTsRx_ethClk       : std_ulogic
SIGNAL doneTsTx_ethClk       : std_ulogic
SIGNAL endOfFrame_ethClk     : std_ulogic
SIGNAL ethClk                : std_ulogic
SIGNAL localNano_ethClk      : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL localSec_ethClk       : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL readTxUdpPort         : std_ulogic
SIGNAL resetTime_ethClk      : std_ulogic
SIGNAL reset_ethClk          : std_ulogic
SIGNAL startOfFrame_ethClk   : std_ulogic
SIGNAL timestampRxNano_ethClk : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampRxNano_int   : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampRxSec_ethClk : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL timestampRxSec_int    : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL timestampTxNano_ethClk : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampTxNano_int   : unsigned(UInt32BitNb-1 DOWNTO 0)
SIGNAL timestampTxSec_ethClk : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL timestampTxSec_int    : unsigned(UInt48BitNb-1 DOWNTO 0)
SIGNAL udpPortOutDelayed     : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```



<company name>	Project: hds
Title: <enter diagram title here>	<enter comments here>
Path: PTP/PTP/struct	
Edited: by samuel.stucky on 22 juin 2015	

```vhdl
--
-- VHDL Architecture TD_TimeBase4DiS.PTPClockCounter.rtl
--
-- Created:
--          by - drozj1.UNKNOWN (WE4136)
--          at - 11:26:24 04.06.2012
--
-- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
--
ARCHITECTURE rtl OF PTPClockCounter IS
    constant margeBit : positive := 2;
    -- signal second: unsigned(sec'length+margeBit downto 0);
    signal second: unsigned(sec'length downto 0);
    signal nanosecond: signed(nano'length+margeBit downto 0);
    signal fractionNano: signed(nano'length+margeBit downto 0);
    signal carryNano: signed(1 downto 0);
    signal carrySec: signed(1 downto 0);
    constant defaultFractionNanoIncrement: unsigned(nano'range) := to_unsigned(0,
    nano'length);
    constant defaultNanoIncrement: unsigned(nano'range) := to_unsigned(400000,nano'
    length);

BEGIN

    sec <= resize(unsigned(second),sec'length);
    nano <= resize(unsigned(nanosecond),nano'length);

    -- increment the fraction of nanosecond every rising edge of clock signal
    -- update carry signal and manage overflow
    fractionsNano:process(clock, reset)
        variable nextFNano: signed(nano'length+margeBit downto 0);
    begin
        if reset = '1' then
            fractionNano <= (others => '0');
            carryNano <= to_signed(0,carryNano'length);
        elsif rising_edge(clock) then
            if resetTime = '1' then
                fractionNano <= (others => '0');
            else
                carryNano <= to_signed(0,carryNano'length);

                if corrEn = '1' then
                    nextFNano := resize(signed(defaultFractionNanoIncrement) +
                    fractionNano + corrFNano,fractionNano'length);
                else
                    nextFNano := resize(signed(defaultFractionNanoIncrement) +
                    fractionNano,fractionNano'length);
                end if;

                if nextFNano >= 1e9 then
                    fractionNano <= nextFNano - 1e9;
                    carryNano <= to_signed(1,carryNano'length);
                elsif nextFNano < 0 then
                    fractionNano <= nextFNano + 1e9;
                    carryNano <= to_signed(-1,carryNano'length);
                else
                    fractionNano <= nextFNano;
                end if;
            end if;
```

```
56
57          end if;
58      end process;
59
60      -- increment the nanosecond every rising edge of clock signal
61      -- update carry signal and manage overflow
62      -- possibility to change nanosecond value
63      ns:process(clock, reset)
64          variable nextNano: signed(nano'length+margeBit downto 0);
65      begin
66          if reset = '1' then
67              nanosecond <= (others => '0');
68              carrySec <= to_signed(0,carryNano'length);
69          elsif rising_edge(clock) then
70                  if resetTime = '1' then
71                      nanosecond <= (others => '0');
72                  else
73                      carrySec <= to_signed(0,carryNano'length);
74
75                      if corrEn = '1' then
76                          nextNano := resize(signed(defaultNanoIncrement) + nanosecond
                            + carryNano + corrNano, nanosecond'length);
77                      else
78                          nextNano := resize(signed(defaultNanoIncrement) + nanosecond
                            + carryNano, nanosecond'length);
79                      end if;
80
81                      if nextNano >= 1e9 then
82                              nanosecond <= nextNano - 1e9;
83                              carrySec <= to_signed(1,carryNano'length);
84                      elsif nextNano < 0 then
85                              nanosecond <= nextNano + 1e9;
86                              carrySec <= to_signed(-1,carryNano'length);
87                      else
88                              nanosecond <= nextNano;
89                      end if;
90                  end if;
91          end if;
92      end process;
93
94      -- increment the second every carry from nanosecond process
95      -- possibility to change second value
96      -- generate PPS signal
97      s:process(clock, reset)
98      begin
99          if reset = '1' then
100             second <= to_unsigned(1350000000,second'length); -- set a date in reset
101             PPS <= '0';
102         elsif rising_edge(clock) then
103             if resetTime = '1' then
104                 second <= to_unsigned(1350000000,second'length); -- set a date in
                    reset
105             else
106                 PPS <= '0';
107
108                 if corrEn = '1' then
109                     second <= resize(unsigned(signed(second) + carrySec + corrSec),
                        second'length);
110                 else
```

```
111                     second <= resize(unsigned(signed(second) + carrySec),second'
                        length);
112                 end if;
113
114                 if carrySec = 1 then
115                     PPS <= '1';
116                 end if;
117             end if;
118         end if;
119     end process;
120
121 END ARCHITECTURE rtl;
```

Package List

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 USE ieee.numeric_std.all;

LIBRARY Ethernet;
USE Ethernet.ethernet.all;

LIBRARY PTP;
 USE PTP.config.all;
```

Declarations

Ports:

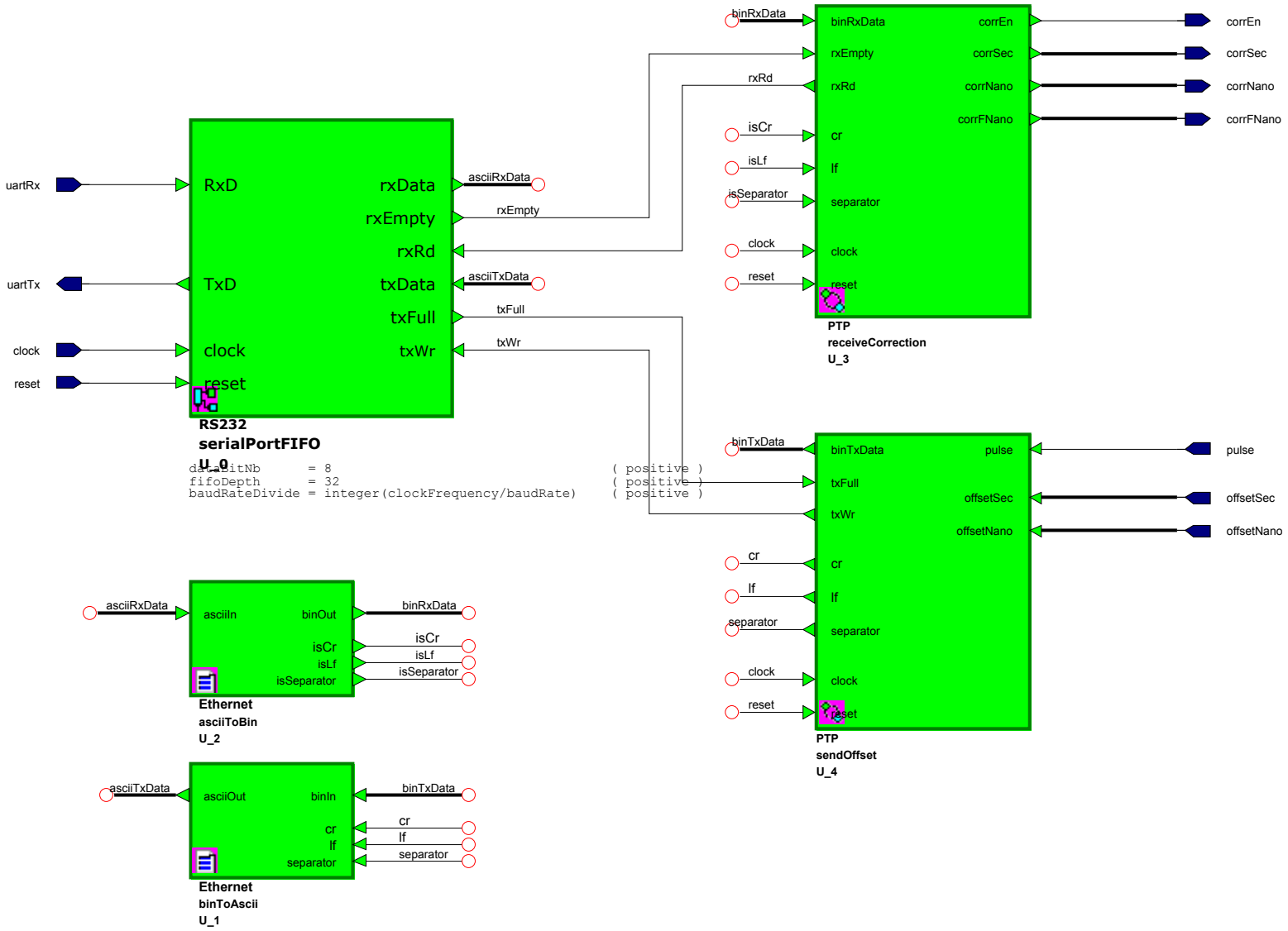
```
    clock       : std_ulogic
    offsetNano  : signed(UInt32BitNb-1 DOWNTO 0)
    offsetSec   : signed(UInt48BitNb-1 DOWNTO 0)
    pulse       : std_ulogic
    reset       : std_ulogic
    uartRx      : std_ulogic
    corrEn      : std_ulogic
    corrFNano   : signed(UInt32BitNb-1 DOWNTO 0)
    corrNano    : signed(UInt32BitNb-1 DOWNTO 0)
    corrSec     : signed(UInt48BitNb-1 DOWNTO 0)
    uartTx      : std_ulogic
```

Pre User:

```
    constant baudRate: real := 115200.0;
    constant clockFrequency: real := 66.0E6;
```

Diagram Signals:

```
    SIGNAL asciiRxData : std_ulogic_vector(7 DOWNTO 0)
    SIGNAL asciiTxData : std_ulogic_vector(7 DOWNTO 0)
    SIGNAL binRxData   : std_ulogic_vector(3 DOWNTO 0)
    SIGNAL binTxData   : std_ulogic_vector(3 DOWNTO 0)
    SIGNAL cr          : std_logic
    SIGNAL isCr        : std_logic
    SIGNAL isLf        : std_logic
    SIGNAL isSeparator : std_logic
    SIGNAL lf          : std_logic
    SIGNAL rxEmpty     : std_ulogic
    SIGNAL rxRd        : std_ulogic
    SIGNAL separator   : std_logic
    SIGNAL txFull      : std_ulogic
    SIGNAL txWr        : std_ulogic
```



<company name>		Project:	hds
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	PTP/ptpFilter/uartFilter		
Edited:	by samuel.stucky on 15 juin 2015		

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 Ethernet;
USE Ethernet.ethernet.all;
```

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
binTxData	OUT		(others => '0')	CLKD
txWr	OUT	'0'	'0'	CLKD
cr	OUT	'0'	'0'	CLKD
lf	OUT	'0'	'0'	CLKD
separator	OUT	'0'	'0'	CLKD
countSec	LOCAL		(others => '0')	CLKD
countNano	LOCAL		(others => '0')	CLKD

State Register Statements

Process Declarations

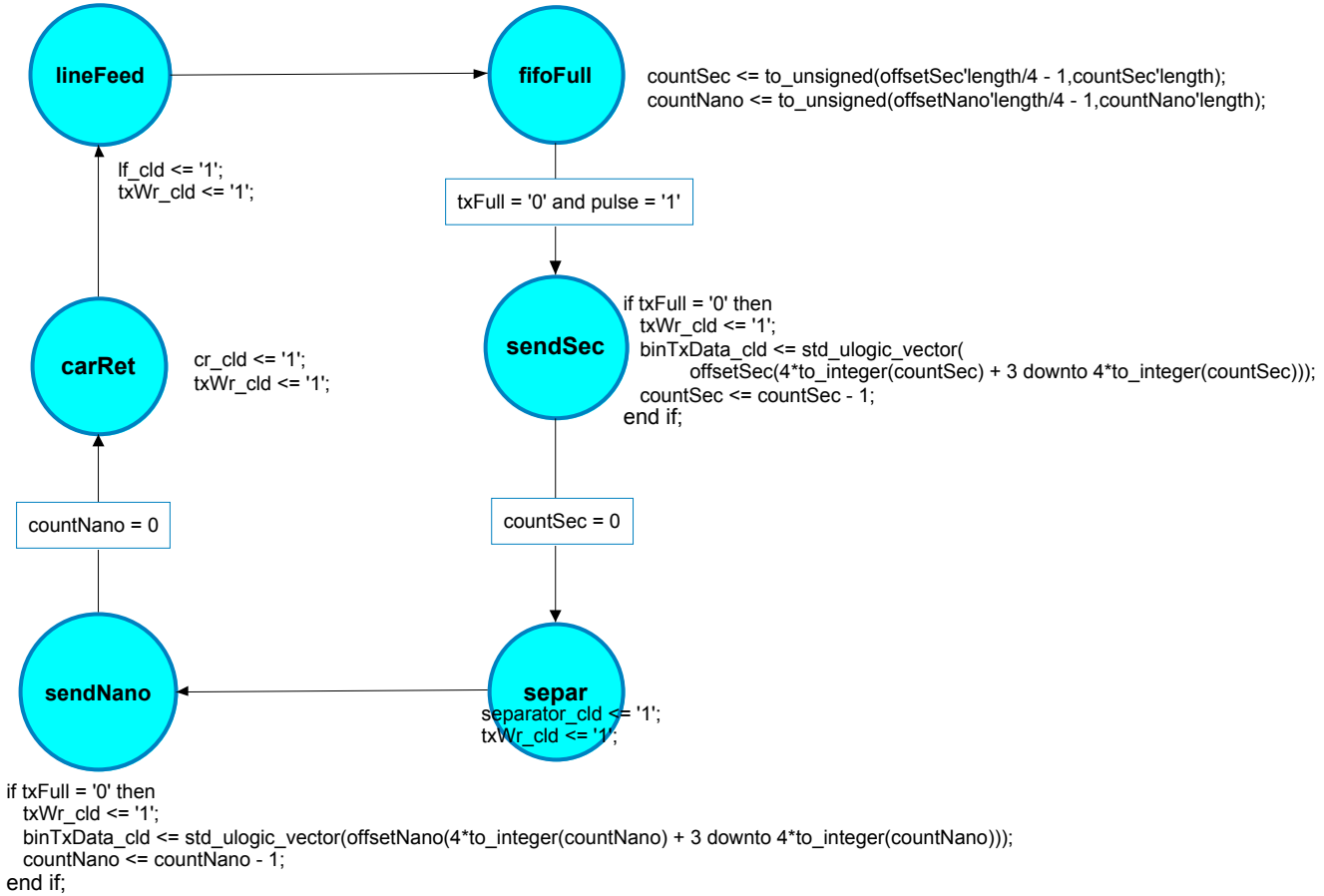
Clocked Process:

Output Process:

clock C clock'EVENT AND clock = '1'

fifoFull

reset r reset = '1' fifoFull




```
if txFull = '0' then
  txWr_cld <= '1';
  binTxData_cld <= std_ulogic_vector(offsetNano(4*to_integer(countNano) + 3 downto 4*to_integer(countNano)));
  countNano <= countNano - 1;
end if;
```

<company name>		Project:	hds
Title:	<enter diagram title here>	<enter comments here>	
Path:	PTP/sendOffset/fsm		
Edited:	by samuel.stucky on 15 juin 2015		

Global Actions

Pre Actions:

Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
corrFNano	OUT		(others => '0')	CLKD
corrNano	OUT		(others => '0')	CLKD
corrSec	OUT		(others => '0')	CLKD
rxRd	OUT	'0'		COMB
countNano	LOCAL		(others => '0')	CLKD
countSec	LOCAL		(others => '0')	CLKD
countFNano	LOCAL		(others => '0')	CLKD
corrFNano_int	LOCAL		(others => '0')	CLKD
corrNano_int	LOCAL		(others => '0')	CLKD
corrSec_int	LOCAL		(others => '0')	CLKD
corrEn	OUT	'0'	'0'	CLKD

State Register Statements

Process Declarations

Clocked Process:

Output Process:

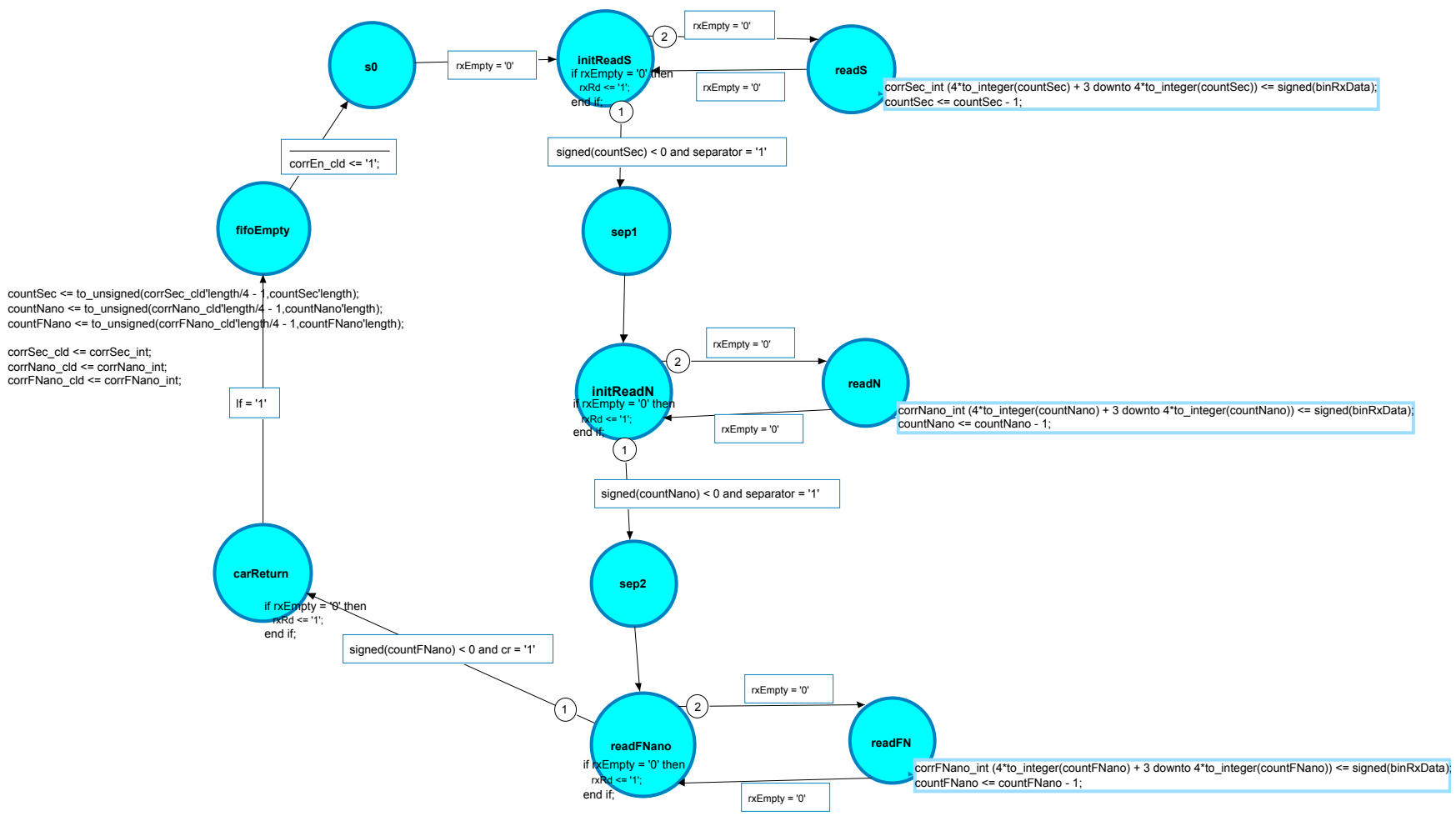
Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

LIBRARY Ethernet;
USE Ethernet.ethernet.all;
```

clock — clock'EVENT AND clock = '1'

fifoEmpty

reset — reset = '1' — fifoEmpty



<company name>		Project:	hds
Title:	<enter diagram title here>	<enter comments here>	
Path:	PTP/receiveCorrection/fsm		
Edited:	by samuel.stucky on 09 juin 2015		

```
__author__ = 'sam'

import serial
import logging

'''
constants
'''

port = 1
baudRate = 115200

nbOfValues = 1
'''
init serial connection
'''
ser = serial.Serial(port - 1, baudRate)

'''
functions
'''


def toHexString(val, nbits):
    return "{0:0{1}X}".format(int((val + (1 << nbits * 4)) % (1 << nbits * 4)), nbits)


def ptpCorrection(seri, offsetlist, offset):
    offsetlist.append(offset)
    if len(offsetlist) >= nbOfValues:
        if all(i > 20e-9 or i < 0 for i in offsetlist):
            log.info("correcting")
            log.info(offsetlist)
            corrSecString = toHexString(-sec, 12)
            corrNanoString = toHexString(-nano, 8)
            corrFNanoFString = toHexString(0, 8)
            del offsetlist[:]
            print("correcting\r\n")
            print(str(sec) + ";" + str(nano) + "\r\n")
            seri.write(corrSecString + ";" + corrNanoString + ";" + corrFNanoFString
            + "\r\n")
            return corrSecString + ";" + corrNanoString + ";" + corrFNanoFString +
            "\r\n"
        offsetlist.pop(0)
    return "000000000000;00000000;00000000\r\n"


offsetlist = list()

'''
logging
'''
formatter = logging.Formatter('%(asctime)s %(message)s')
log = logging.getLogger('simple_example')
fh = logging.FileHandler('ethBusyLog.log')
fh.setFormatter(formatter)
log.addHandler(fh)
log.setLevel(logging.INFO)

```

```
'''
start data collection
'''

while True:
    data = ser.readline()  # read data from serial port
    print(repr(data))

    offsetSec = float(int(data.split(";")[0], 16))
    offsetNano = float(int(data.split(";")[1], 16))

    if offsetSec > 0x7FFFFFFFFFF:
        offsetSec -= 0x1000000000000        # offset is negative

    offset = offsetSec + offsetNano / 1000000000
    sec = int(offset)
    nano = (offset - sec) * 1000000000.0

    print(str(sec) + "sec "
          + str(int(nano / 1000000)) + "ms "
          + str((int(nano / 1000) - int(nano / 1000000) * 1000)) + "us "
          + str(int(nano) - int(nano / 1000) * 1000) + "ns")

    if offset != 0:
        log.info(str(sec) + "sec "
                 + str(int(nano / 1000000)) + "ms "
                 + str((int(nano / 1000) - int(nano / 1000000) * 1000)) + "us "
                 + str(int(nano) - int(nano / 1000) * 1000) + "ns")

    ptpCorrection(ser, offsetlist, offset)

```

```vhdl
--
-- VHDL Architecture PTP.ptpFilter.localFilter
--
-- Created:
--          by - samuel.stucky.UNKNOWN (WE5405)
--          at - 14:17:45 17.06.2015
--
-- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
--
ARCHITECTURE localFilter OF ptpFilter IS
BEGIN

    corrFNano <= (others => '0');
    uartTx <= '0';


    corr: process(clock, reset)
    begin
        if reset = '1' then
            corrEn <= '0';
            corrSec <= (others => '0');
            corrNano <= (others => '0');
        elsif rising_edge(clock) then
            corrEn <= '0';
            if pulse = '1' then
                if (offsetSec /= 0) or (offsetNano < 0) or (offsetNano > 20) then
                    corrEn <= '1';
                    corrSec <= -offsetSec;
                    corrNano <= -offsetNano;
                end if;
            end if;
        end if;

    end process corr;

END ARCHITECTURE localFilter;


```

Package List

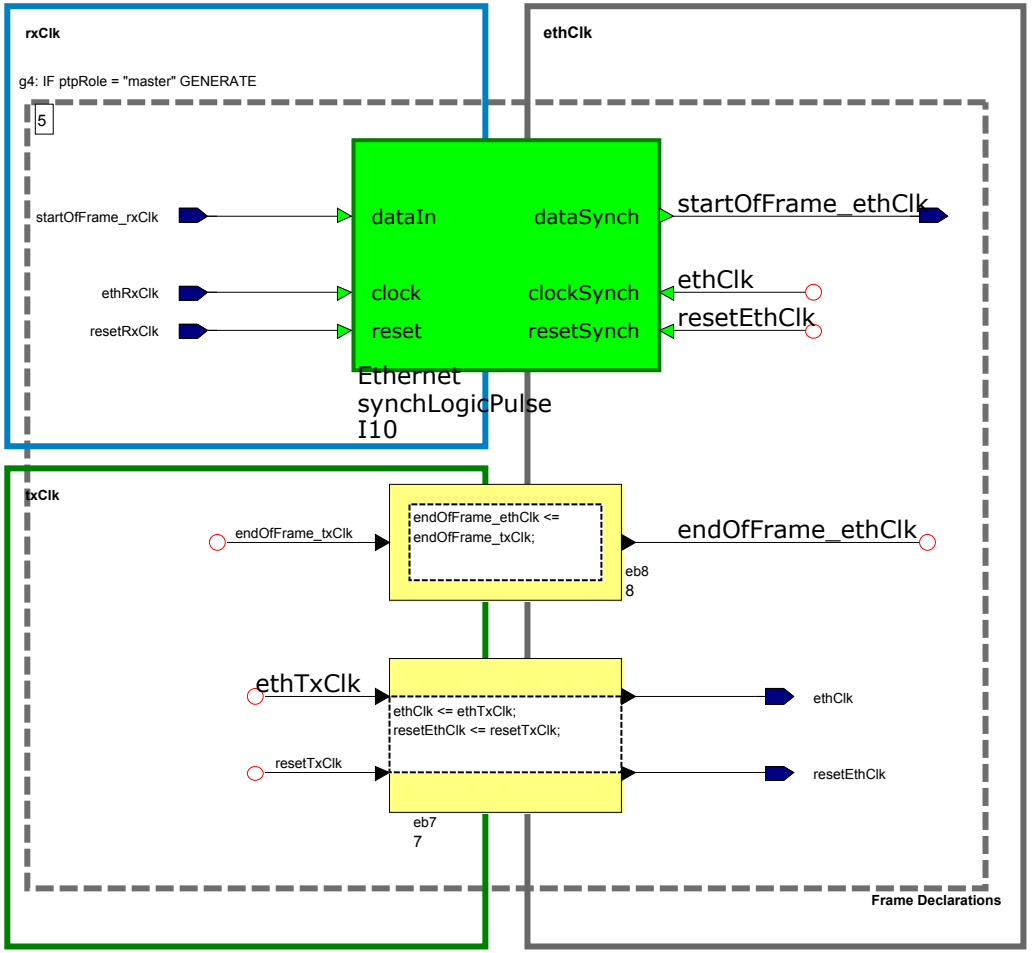
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY Ethernet;
USE Ethernet.ethernet.ALL;
LIBRARY PTP;
 USE PTP.config.all;
```

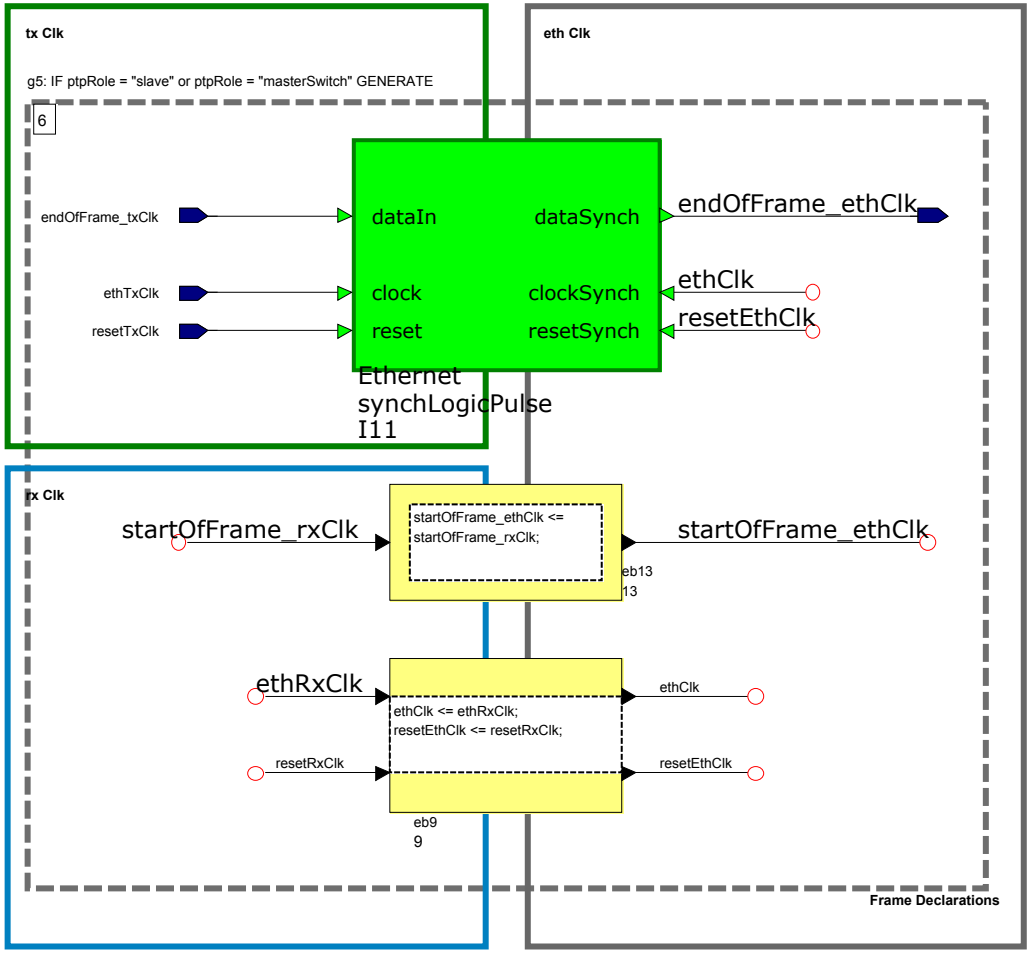
Declarations

Ports:

```
endOfFrame_txClk    : std_ulogic
ethRxClk            : std_uLogic
ethTxClk            : std_uLogic
resetRxClk          : std_ulogic
resetTxClk          : std_ulogic
startOfFrame_rxClk  : std_ulogic
endOfFrame_ethClk   : std_ulogic
ethClk              : std_ulogic
resetEthClk         : std_ulogic
startOfFrame_ethClk : std_ulogic
```

Diagram Signals:





<company name>		Project:	hds
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	PTP/SynchRxTxClk/struct		
Edited:	by samuel.stucky on 18 juin 2015		

Package List

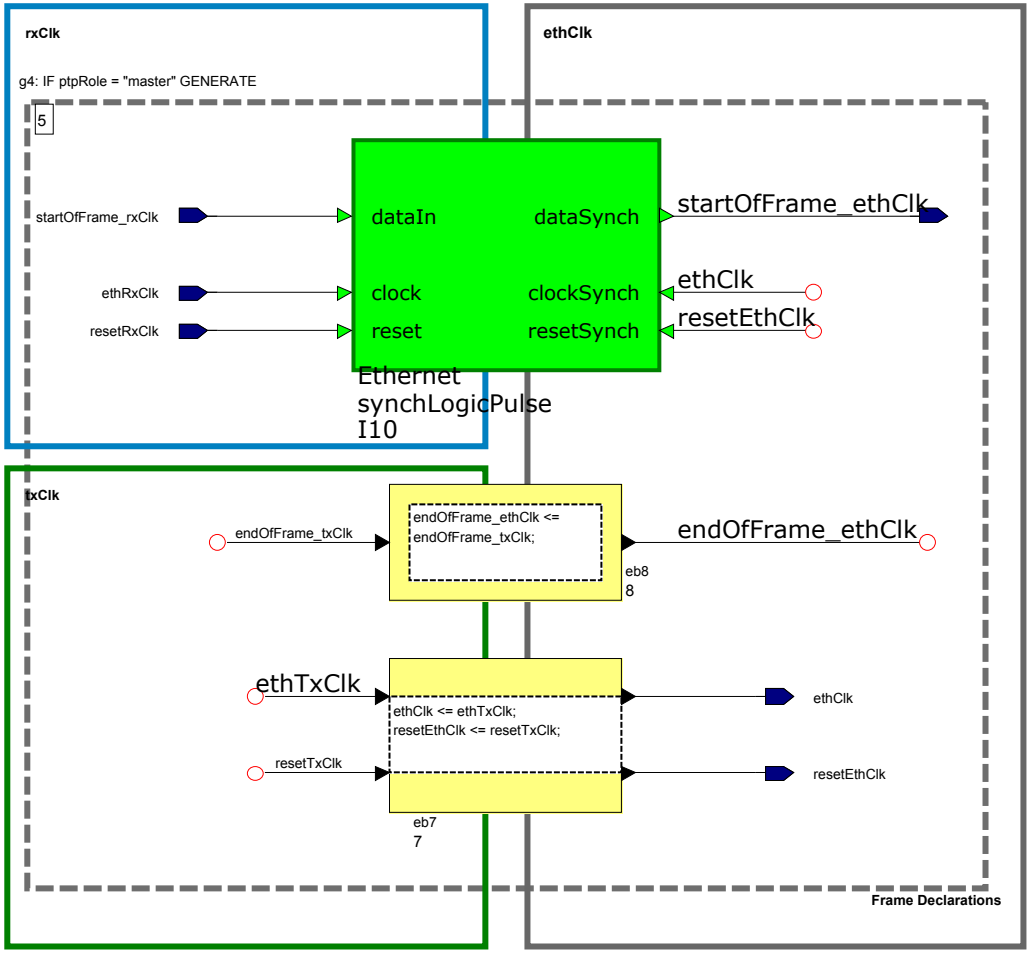
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY Ethernet;
USE Ethernet.ethernet.ALL;
LIBRARY PTP;
 USE PTP.config.all;
```

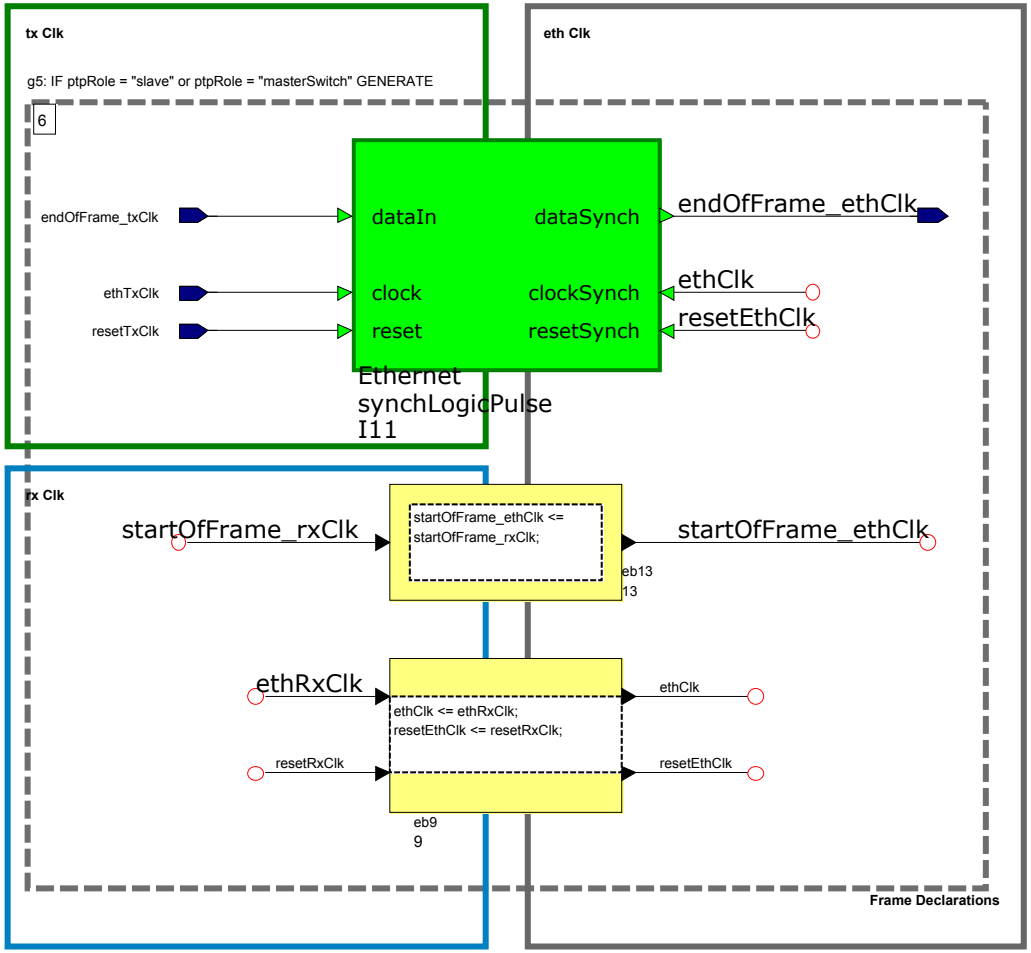
Declarations

Ports:

```
endOfFrame_txClk    : std_ulogic
ethRxClk            : std_uLogic
ethTxClk            : std_uLogic
resetRxClk          : std_ulogic
resetTxClk          : std_ulogic
startOfFrame_rxClk  : std_ulogic
endOfFrame_ethClk   : std_ulogic
ethClk              : std_ulogic
resetEthClk         : std_ulogic
startOfFrame_ethClk : std_ulogic
```

Diagram Signals:





<company name>		Project:	hds
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	PTP/SynchRxTxClk/struct		
Edited:	by samuel.stucky on 18 juin 2015		

Package List

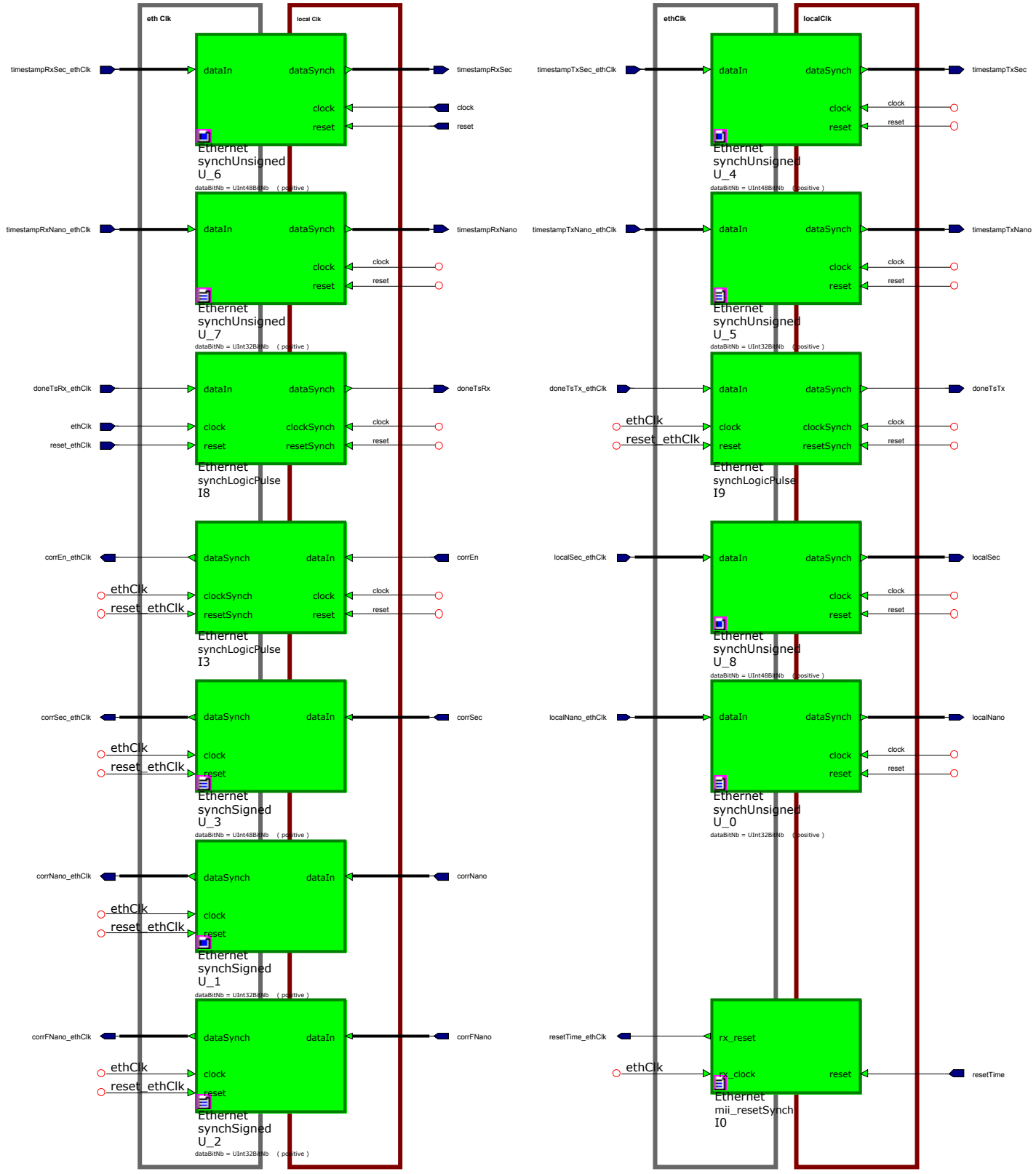
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY Ethernet;
USE Ethernet.ethernet.ALL;
LIBRARY PTP;
 USE PTP.config.all;
```

Declarations

Ports:

```
clock                : std_ulogic
corrEn               : std_ulogic
corrFNano            : signed(UInt32BitNb-1 DOWNTO 0)
corrNano             : signed(UInt32BitNb-1 DOWNTO 0)
corrSec              : signed(UInt48BitNb-1 DOWNTO 0)
doneTsRx_ethClk      : std_ulogic
doneTsTx_ethClk      : std_ulogic
ethClk               : std_ulogic
localNano_ethClk     : unsigned(UInt32BitNb-1 DOWNTO 0)
localSec_ethClk      : unsigned(UInt48BitNb-1 DOWNTO 0)
reset                : std_ulogic
resetTime            : std_logic
reset_ethClk         : std_ulogic
timestampRxNano_ethClk : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampRxSec_ethClk  : unsigned(UInt48BitNb-1 DOWNTO 0)
timestampTxNano_ethClk : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampTxSec_ethClk  : unsigned(UInt48BitNb-1 DOWNTO 0)
corrEn_ethClk        : std_ulogic
corrFNano_ethClk     : signed(UInt32BitNb-1 DOWNTO 0)
corrNano_ethClk      : signed(UInt32BitNb-1 DOWNTO 0)
corrSec_ethClk       : signed(UInt48BitNb-1 DOWNTO 0)
doneTsRx             : std_ulogic
doneTsTx             : std_ulogic
localNano            : unsigned(UInt32BitNb-1 DOWNTO 0) --
localSec             : unsigned(UInt48BitNb-1 DOWNTO 0) --
resetTime_ethClk     : std_ulogic
timestampRxNano      : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampRxSec       : unsigned(UInt48BitNb-1 DOWNTO 0)
timestampTxNano      : unsigned(UInt32BitNb-1 DOWNTO 0)
timestampTxSec       : unsigned(UInt48BitNb-1 DOWNTO 0)
```

Diagram Signals:



<company name>		Project: hds
		<enter comments here>
Title:	<enter diagram title here>	
Path:	PTP/SynchEthMainClk/struct	
Edited:	by samuel.stucky on 18 juin 2015	

```vhdl
--
-- VHDL Architecture Board.initializer.simple
--
-- Created:
--          by - samuel.stucky.UNKNOWN (WE5405)
--          at - 09:08:41 22.04.2015
--
-- using Mentor Graphics HDL Designer(TM) 2009.2 (Build 10)
--

LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

library Common;
  use Common.CommonLib.all;


ARCHITECTURE simple OF initializer IS

  constant registerNb : positive := 32;


  signal writeReg   : std_ulogic;
  signal regAddr    : unsigned(TxAddr'range);
  signal counter    : unsigned((regAddr'length + 2) - 1 downto 0);

BEGIN


  count: process(reset,clock)
  begin
    if reset = '1' then
      counter <= (others=>'0');
    elsif rising_edge(clock) then
      if counter < (2 * registerNb) then
        counter <= counter + 1;
      end if;
    end if;
  end process count;

  regAddr <= resize(counter/2,regAddr'length);
  TxAddr  <= regAddr;
  TxWr    <= writeReg and counter(0);

  setRegisters: process(regAddr, counter)

  begin
    RxRd      <= '0';
    TxRdWr    <= '0';

    case to_integer(regAddr) is
      --when 0      =>  TxData    <= "0010000100000000";
      --                writeReg  <= '1';
      --when 4      =>  TxData    <= "0000000111100001";
      --               writeReg  <= '1';
      --when 23     =>  TxData    <= "0100000000000001";
      --                writeReg  <= '1';
      --when 30     =>  TxData    <= "0000000000000000";
```

```
60          --                  writeReg  <= '1';
61          --when 31     =>   TxData    <= "1000000100000000";
62          --                  writeReg  <= '1';
63          when others =>  TxData    <= (others => '0');
64                          writeReg  <= '0';
65        end case;
66
67      end process setRegisters;
68    END ARCHITECTURE simple;
69
70
```