Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# PROJECTE DE FI DE CARRERA

**TÍTOL DEL PFC: Construcció d'una maqueta de xarxa OBS. Disseny i Implementació del Pla de Control.**

**TITULACIÓ: Enginyeria de Telecomunicació (segon cicle)**

**AUTOR: Joan Triay Marquès**

**DIRECTOR: Cristina Cervelló i Pastor**

**DATA: 8 de setembre de 2006**

**Títol:** Construcció d'una maqueta de xarxa OBS. Disseny i Implementació del Pla de Control.

**Autor:** Joan Triay Marquès

**Director:** Cristina Cervelló i Pastor

**Data:** 8 de setembre de 2006

**Resum**

Aquest document presenta el disseny i la implementació del Pla de Control d'una xarxa de Commutació de Ràfegues Òptiques (OBS, *Optical Burst Switching*). En aquests tipus de xarxes, les dades d'usuari s'ensamblen en ràfegues de major longitud en els nodes frontera d'ingrés, i es transmeten passat un temps d'*offset*. Aquest temps d'*offset* permet que un paquet de control realitzi la senyalització i reserva de recursos òptics per a que la ràfega pugui commutar-se des del node frontera d'origen al node destí, on es realitzarà el desensamblatge de la ràfega en les seves unitats de dades originals.

L'objectiu d'aquest projecte és la creació del Pla de Control que governa la senyalització del camí entre origen i destinació pel qual la ràfega viatjarà, el qual es concep per a ser utilitzat indiferentment en qualsevol tipus de node OBS (frontera o central). Paral·lelament en el document s'explica el protocol de missatges desenvolupat i la forma en que s'assignen i reserven els recursos òptics en els commutadors i en els elements de transmissió i recepció de les ràfegues. La realització del projecte implica part de disseny hardware (VHDL) i programació del software de control (en C).

La implementació del Pla de Control s'ha realitzat sobre unes plaques de desenvolupament hardware/software que integren una FPGA Virtex-II Pro de Xilinx.

Aquest Projecte de Fi de Carrera s'emmarca dins d'un projecte de disseny i implementació d'una maqueta de xarxa OBS, com a part del projecte TIC2003-09042-C03-02, finançat pel Ministeri de Ciència i Tecnologia, juntament amb el projecte *Machine* de la Fundació i2CAT.

**Title:** Assembling of an OBS network testbed. Design and Implementation of the Control Plane.

**Author:** Joan Triay Marquès

**Director:** Cristina Cervelló i Pastor

**Date:** September, 8th 2006

**Overview**

This document presents the design and implementation of the Control Plane of an Optical Burst Switched Network (OBSN). In this kind of network, user's data is assembled in a burst at the ingress node and transmitted after an offset time. This offset time allows for a control packet to signal and reserve optical resources for switching the burst from the ingress node to its destination, where the disassembling of the burst is done.

The aim of this project is to create the Control Plane which controls the signaling of the path between origin and destination, throughout the burst is switched. This Control Plane can be either used on an edge or on a core OBS network node. The document also presents the message protocol and the way of reserving optical resources on the switches and on the transmission/reception interfaces. On the implementation of this project, some hardware design (VHDL) has been necessary, as well as the programming of the control plane software (in C).

The implementation of the Control Plane has been done on a hardware/software development board which it fits a Virtex-II Pro FPGA of Xilinx.

This PFC is part of a bigger project for the design and implementation of an OBS network testbed, within the framework of the TIC2003-09042-C03-02 project, founded by the Spanish Ministerio de Ciencia y Tecnología (MCYT), and the *Machine* project of the I2CAT's Foundation.

# ÍNDEX

# ÍNDEX DE FIGURES i GRÀFICS

# ÍNDEX DE TAULES

# INTRODUCCIÓ

En els últims anys hi ha hagut molts avenços en el camp de les tecnologies de multiplexació d'ones òptiques, el que ha permès augmentar en varis ordres de magnitud les capacitats de les fibres òptiques. Aquest increment ha estat paral·lel al creixement dels amples de banda necessaris per a transportar els continus increments de tràfic a Internet. En aquest sentit, es necessiten, per tant, noves tecnologies que ofereixin grans capacitats de transmissió i commutació de la informació.

L'*Optical Burst Switching* (OBS, Commutació de Ràfegues Òptiques) és un nou paradigma capaç d'explotar els terabits d'ample de banda que la tecnologia de transmissió *Wavelength-Division Multiplexing* (WDM, Multiplexació per Divisió d'Ona) permet actualment. En l'OBS, els paquets de dades que tenen una destinació comuna s'agreguen en un node frontera d'ingrés en una ràfega (de major longitud que els paquets). Abans de transmetre's la ràfega, es marca i reserva el camí pel qual "viatjarà" òpticament. Posteriorment, aquesta ràfega es transmet, s'encamina i commuta individualment. D'aquesta forma, les dades útils viatgen completament en el domini òptic, i no es requereix cap conversió O/E/O, que provoca que el rendiment de commutació baixi. En canvi, per a realitzar la reserva al llarg del camí, sí que es requereix que els paquets de control es processin en el domini elèctric.

Un cop la ràfega òptica arriba al node frontera de sortida, es realitza en aquest el desensamblatge de la ràfega en les diferents unitats o paquets de dades, i s'encaminen finalment cap a la seva destinació.

En els darrers anys s'han proposat molts i variats protocols per a realitzar, tant les reserves de recursos òptics en els nodes de commutació, com per a resoldre les possibles contencions a l'hora de commutar les ràfegues. Però gran part de tota aquesta investigació en el camp OBS s'ha desenvolupat per mitjà de càlculs i simulacions. Si bé és important realitzar simulacions per a validar alguns resultats, és necessari, també, crear i fabricar maquetes reals de xarxes OBS per a estudiar el comportament dels diferents algoritmes proposats sota condicions reals de funcionament. Com es veurà, el nombre de maquetes i proves de concepte d'OBS són reduïts a nivell mundial, i d'aquí la necessitat del present Projecte de Fi de Carrera.

L'objectiu del projecte, globalment, és la creació d'una maqueta de xarxa OBS real, amb dispositius i nodes físics, i la seva interconnexió amb commutadors òptics. En particular, el present PFC es centra en la definició i la implementació del Pla de Control. Aquest necessita de la creació del protocol d'intercanvi de missatges i paquets de control, i la creació del programa de control (OCPS, *OBS Control Plane Software*) que permeti realitzar les reserves dels recursos i assignar-los a la transmissió, recepció i commutació de les ràfegues òptiques.

La memòria del PFC està organitzada de la següent manera. En el Capítol 1 es realitza una breu introducció al OBS, definint les principals característiques de

les xarxes OBS. També es descriuen els diferents tipus de nodes que les integren i es defineixen quines funcionalitats ofereix cadascun. Aquest punt és important, perquè conceptualment s'intenta diferenciar clarament els dos tipus de node (frontera/central). Però de fet, el disseny i la implementació del nostre Pla de Control es concep per a que funcioni indiferentment del tipus de node, i fins i tot, està concebut per a que funcioni en nodes que realitzen tant les funcions de frontera com de node central.

El disseny del programa de control pren molts conceptes dels Sistemes de Temps Real. En el Capítol 2 es presenta una breu introducció a aquest paradigma, i en especial, al tipus d'algoritme del programa que es pretén utilitzar en el nostre programa.

El Capítol 3 descriu el disseny del Pla de Control. En el capítol es presenten la maqueta de xarxa que es vol crear en el projecte i els requeriments concrets per al Pla de Control de qualsevol node d'aquesta xarxa. A continuació es realitza la descripció del disseny, tant la funcional, com la dels protocols de missatges proposats. També es descriu com es defineixen els recursos en el nostre sistema, i com es fa possible la commutació de les ràfegues i l'encaminament dels missatges de control. Finalment, s'analitza més detalladament l'algorítmica del nostre programa de control.

En el següent capítol es descriu la implementació del sistema sobre les plaques de desenvolupament FPGA d'Avnet, que disposen d'una FPGA Virtex-II Pro 30 de Xilinx. La implementació del sistema requereix d'una part hardware, possible gràcies a la FPGA, i d'una part software (el programa de control) que s'executa sobre un microprocessador PowerPC 405 d'IBM (integrat dins la FPGA).

El Capítol 5 ofereix els resultats a diferents proves realitzades per a, per una banda validar el correcte funcionament del disseny i implementació proposats, i per una altra, realitzar una primera valoració al rendiment del programa. També en aquest capítol s'estudien els rendiments en la reserva de recursos i es realitzen alguns càlculs per a determinar el *throughput* aconseguit en el sistema.

El darrer capítol presenta les conclusions del projecte i les línies futures a tenir en consideració per a seguir desenvolupant i millorant el disseny proposat en aquest PFC. També en aquest capítol es realitza un petit estudi de l'impacte mediambiental del projecte.

La memòria es complementa amb una sèrie d'annexos. Els primers dos annexos són de caire més informatiu sobre el Sistemes de Temps Real i la seva possible aplicació al processador utilitzat en la nostra implementació. En els annexos també hi ha els fulls d'especificacions de la placa de desenvolupament i la FPGA, així com també gràfics, taules i descripcions de configuracions de parts del hardware, i els diagrames de flux de diferents funcions del nostre programa de control. Per falta d'espai en el cos de la memòria, s'inclouen en els annexos les taules de resultats. Finalment, també s'inclou un article presentat al EUNICE que ha estat acceptat per a presentar-lo a Stuttgart, Alemanya, en el mes de setembre d'aquest any 2006.

# CAPÍTOL 1. OPTICAL BURST SWITCHING

L'objectiu primer d'aquest projecte és el disseny i implementació del Pla de Control per a una xarxa basada en la commutació de ràfegues òptiques (OBS). Aquest capítol presenta un breu resum d'aquesta tecnologia, incidint particularment en les principals característiques d'aquesta i en altres que justifiquen la realització d'un disseny del Pla de Control que ofereixi el millor rendiment possible. El capítol finalitza amb una exposició de l'Estat de l'Art de l'OBS.

## 1.1. Introducció

Els sistemes de comunicació òptics i l'ús de la fibra òptica per a la transmissió de les dades varen guanyar fama arran de la invenció de la Multiplexació per Divisió en Longitud d'Ona (WDM). Aquesta, a la vegada, ha evolucionat en diferents tecnologies [1]: les *Wavelength-Routed Networks* (WRNs), les *Optical Burst Switched Networks* (OBSNs) i *Optical Packet Switched Networks* (OPSNs). Totes aquestes tecnologies s'engloben dins les anomenades *All-Optical Networks* (AONs), si bé aquest terme pot resultar confós, ja que no totes les tecnologies anteriors realitzen tot el processament de la informació a nivell òptic.

L'operació principal de les WRNs es basa en la configuració de circuits/connexions, anomenats *lightpaths*, entre diferents nodes de xarxa (connexió orientada a circuit). Presenta certs desavantatges, com ara que el nombre de longituds d'ona està limitat, i per tant, el crear una xarxa mallada entre els usuaris és impossible; així com també el grau d'utilització del canal és menor al estar el circuit disponible de forma dedicada. Aquest tipus de xarxes solen utilitzar el GMPLS com a protocol de senyalització.

En el cas de les OPSNs, el tràfic d'usuari es transporta íntegrament en "paquets òptics" juntament amb la informació de control. Però aquests tipus de xarxes també presenten inconvenients com la inexistència de *buffers* òptics equiparables als elèctrics i les dificultats per a que la commutació dels paquets es realitzi el suficientment ràpid (es requeririen commutadors òptics amb capacitat de realitzar més de 10MOperacions/s).

A camí entre els dos tipus de xarxes anteriors, tenim les OBSNs. Aquestes es basen en la commutació de ràfegues òptiques i en una forta diferenciació entre el seu Pla de Dades i el Pla de Control. Com que les ràfegues òptiques són agregació de paquets, i per tant, són de major longitud, els temps de commutació sí són possibles en aquest cas, i per tant, presenten un clar avantatge respecte la commutació òptica de paquets [2][3].

A continuació es descriuen més detingudament les característiques, funcionalitats i protocols de reserva de recursos de les xarxes OBS.

## 1.2. Característiques de les xarxes OBS

En les xarxes OBS el bloc de dades bàsic que es transmet s'anomena *burst* (ràfega). Una ràfega és un conjunt de paquets de dades que tenen la mateixa adreça de xarxa destí o que tenen altres atributs comuns, com per exemple requeriments de Qualitat de Servei (QoS).

Els paquets s'ensamblen en ràfegues en el node frontera d'ingrés a la xarxa, s'encaminen a través de la xarxa OBS mitjançant commutacions òptiques en els nodes centrals i es desensamblen en paquets en el node frontera de sortida de la xarxa OBS. Posteriorment, aquests paquets poden arribar a la destinació utilitzant la informació d'encaminament de les capçaleres IP (en cas de tractar-se de paquets IP).

Una ràfega consisteix en una capçalera de ràfega, o paquet de control (terme utilitzat al llarg del document) i en una ràfega de dades. En OBS, la ràfega de dades i el seu paquet de control es transmeten de forma separada en diferents canals/plans, amb el paquet de control just per davant de les dades. A més, la ràfega de dades es commuta totalment en el domini òptic, mentre que els paquets de control, com que requereixen cert processament s'encaminen i processen en el domini elèctric. Aquests paquets de control contenen tota la informació necessària per a que en els nodes de la xarxa es pugui commutar la ràfega òpticament.

Les ràfegues de dades i els paquets de control diferencien el Pla de Dades i el Pla de Control (veure Fig. 1.1). Ambdós tenen requeriments diferents; per una banda el Pla de Dades ha de realitzar l'ensamblatge i desensamblatge de ràfegues el més ràpid possible. Per a això, es requereix l'ús de memòries ràpides, i també grans, ja que les ràfegues poden arribar a ser de centenars de milers de bytes. En canvi, el Pla de Control ha d'aconseguir que el processament de la informació continguda en els paquets de control, la creació i recepció d'aquests sigui molt ràpid, així com també el processament dels algoritmes de reserva.

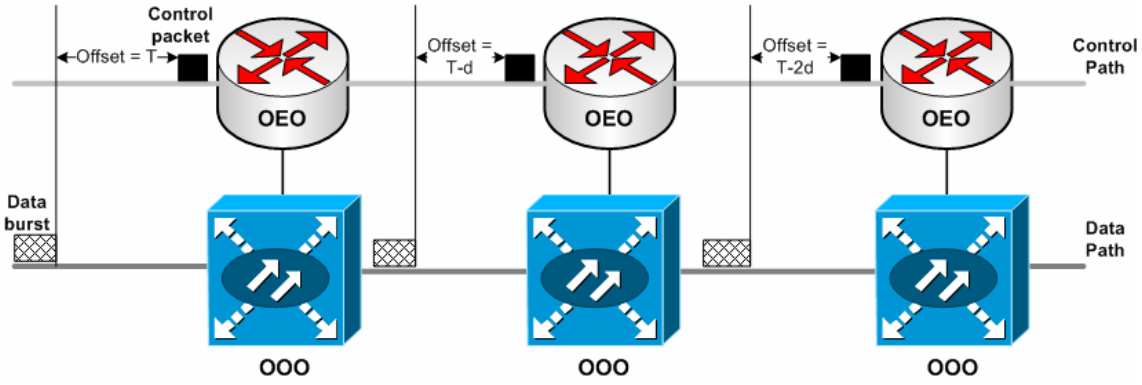



**Fig. 1.1.** Plans de Control i Dades d'una xarxa OBS separats.

## 1.3. Nodes en una xarxa OBS

En la xarxa OBS es diferencien dos tipus de nodes segons les tasques que realitzen cadascun d'ells i la situació d'aquests en relació amb els usuaris. Els nodes frontera, els més propers als usuaris (o a les xarxes dels usuaris), tenen les següents funcionalitats:

- Ensamblatge i desensamblatge de les ràfegues en funció de l'adreça de destinació o d'altres atributs.
- Transmissió i recepció de ràfegues en certs moments en una determinada longitud d'ona o canal.
- Creació de paquets de control en funció del tipus de tràfic (ràfegues) que es transmetrà.
- Recepció i processament dels paquets de control enviats pels altres nodes.

Per altra banda, els nodes del nucli de la xarxa realitzen les següents funcions:

- Recepció i processament dels paquets de control.
- Modificació del paquet de control, si es requereix, i transmissió al següent node OBS en el camí fins al node frontera de sortida.
- Reserva i alliberació de recursos en la matriu de commutació òptica (OXC) segons la informació continguda en els paquets de control rebuts.
- Commutació de les ràfegues òptiques de dades.

Per a que el rendiment global de la xarxa sigui el més òptim possible, totes les tasques presentades en els punts anteriors s'han de realitzar el més ràpidament possible, ja que així es permetrà que la taxa de commutació/transmissió en els nodes sigui el major possible.

### 1.3.1. Node OBS frontera

En els nodes frontera es realitza una de les funcions bàsiques en OBS, la recol·lecció de les dades dels usuaris, la seva distribució segons l'adreça de destinació, o algun altre atribut, i la generació de la ràfega. Els algoritmes per a generar aquesta ràfega són diversos i cadascun ofereix certs avantatges que impacten sobre el rendiment global de la xarxa [2][4][5]. Bàsicament aquests algoritmes es diferencien en la forma de generar les ràfegues: de longitud fixa, per temps fix, dinàmics (els valors de longitud i temps poden variar en el temps), o composats (si s'arriba a la longitud establerta o al temps, indistintament, es finalitza la creació de la ràfega). Així mateix, segons els valors de longitud i temps es poden crear diferents classes de servei (CoS) per a tractar els fluxos en la xarxa segons diferents requeriments.

Un cop la ràfega està construïda s'ha de transmetre. Per a tal efecte, cal primer de tot senyalitzar la connexió que s'establirà al llarg de la xarxa, i sobre la qual la ràfega es transmetrà òpticament. Les diferents formes de realitzar aquesta reserva de recursos òptics es comenten en l'apartat 1.3.2.1.

La ràfega de dades no es transmet tant punt està feta, sinó que es guarda en memòria un temps d'espera, conegut com *offset*. Posteriorment la ràfega es

transmet sense que s'esperi un reconeixement positiu (ACK) que notifiqui que el camí extrem a extrem ha estat establert (o reservat). Aquest sistema de reserva és apropiat per a una xarxa OBS, que de fet està concebuda per a actuar com una xarxa de llarga distància. Algunes de les proves realitzades amb aquest tipus de reserva *one-way*, o *tell-and-*go, han concluït que el temps de configuració és menor, i per tant, el rendiment de *throughput* millora. Però aquest esquema de reserva únic pot provocar pèrdues perquè els paquets de control poden no complir la seva tasca de reservar recursos en els nodes intermedis de la xarxa, i per tant, que la ràfega es transmeti per la xarxa sense cap tipus de reserva establerta, la qual comportaria la pèrdua de la ràfega i la informació continguda en ella.

Per altra banda, en els sistemes de reserva *two-way*, l'*offset* és el temps que es necessita per a rebre la confirmació del destí. El principal desavantatge d'aquesta classe de sistema és el gran temps d'*offset*, que provoca grans retards en la transmissió de les dades. Alguns protocols exemples d'aquest esquema són el *Tell-And-Wait* (TAW) [6] i el WR-OBS, proposat en [7].

#### *1.3.1.1. Càlcul del temps d'Offsett*

Abans de transmetre la ràfega de dades, el node frontera s'espera un temps d'*offset* després de transmetre el paquet de control. Aquest temps d'*offset* permet que el paquet de control pugui reservar els recursos necessaris al llarg de la xarxa i establir el camí pel qual la ràfega òptica viatjarà. Els nodes intermedis necessiten d'aquest temps per a poder configurar la commutació i realitzar la reserva dels ports/canals en els nodes.

Idealment, l'estimació de l'*offset* hauria de ser segons el nombre de salts entre l'origen i destinació, i segons el nivell de congestió en la xarxa. Si es realitza aquesta estimació de forma incorrecta, es poden produir pèrdues de ràfega perquè aquesta pot arribar al node OBS abans que la commutació òptica estigui activa en el node. Per tant, la determinació d'aquest *offset* és de vital importància per a que el rendiment i la taxa de pèrdues de ràfegues sigui òptim.

Hi ha diferents variacions per a calcular aquest període:

- *Offset Fix:* És l'esquema més popular i s'utilitza en el *Just-Enough-Time*, JET, [8]. En aquest el temps d'*offset* és fix i igual a la suma del temps total de processament en els salts OBS intermedis, més el temps de configuració dels commutadors òptics. Aquest càlcul requereix el nombre precís de salts entre l'origen i la destinació. A més, s'assumeix que el temps de commutació és pràcticament el mateix per a cada node, però a la pràctica, aquests temps poden variar en funció dels retards d'encuament.

- *Offset Estadístic:* Proposat per Verma [9], aquest esquema està basat en la generació de l'*offset* de forma variable, en el qual, el node frontera d'ingrés genera *tokens* de transmissió basant-se en un procés de *Poisson* amb un temps d'arribada predeterminat. Un cop la ràfega està

generada s'envia immediatament el paquet de control, i la ràfega de dades s'espera fins que obté un *token* de transmissió.

- *Offset WR-OBS:* En aquesta arquitectura l'*offset* es calcula com la suma de temps que pren un node client OBS en demanar recursos des d'un planificador centralitzat, el temps de computació de l'encaminament i l'algoritme d'assignació de longitud d'ona, i el temps de senyalització del camí.

## 1.3.2. Node OBS central

En els nodes centrals de la xarxa es realitza principalment la reserva de recursos òptics per a que la commutació òptica de les ràfegues sigui possible. La reserva de recursos es duu a terme segons la informació continguda en els paquets de control. A continuació s'anomenen els principals esquemes de reserva de recursos (o longituds d'ona).

### *1.3.2.1. Esquemes de reserva de longituds d'ona*

Segons els mecanismes de *setup* (configuració/reserva de recursos) i *release* (alliberament de recursos) en [10], es descriuen quatre tipus d'Esquemes de Reserva de Longitud d'Ona:

1) *Setup* explícit i *Release* explícit. En aquest esquema, el paquet de control conté l'*offset* de la ràfega, però no la duració d'aquesta. La reserva dels recursos comença immediatament després que el node rebi el missatge de *Setup*, i finalitza quan es rep el missatge de *Release*. Una implementació particular d'aquest esquema és el *Just-In-Time* (JIT), proposat per Wei i McFarland [11]. El diagrama A) de la Fig. 1.2 mostra un exemple del seu funcionament.

2) *Setup* explícit i *Release* estimat. En aquest cas, el missatge de *Setup* conté l'*offset* i la duració de la ràfega. Cada longitud d'ona té un temps de finalització associat que indica quan el recurs passarà a estar lliure. La reserva comença quan el node rep el missatge de *Setup* i finalitza quan la ràfega ha estat commutada i transmesa, temps que es calcula utilitzant el camp de duració de la ràfega. Un exemple d'aquest tipus és l'esquema *Horizon* proposat per Turner [12], i del que se'n mostra un exemple en el diagrama B) de la figura Fig. 1.2.

3) *Setup* estimat i *Release* explícit. Aquest esquema es caracteritza per utilitzar missatges de control de *Setup* que contenen només l'*offset* de la ràfega. La reserva comença just a l'inici de la ràfega. Aquest moment es calcula utilitzant la informació d'*offset* prèvia. L'alliberament de recursos s'executa un cop el missatge de *Release* arriba.

4) *Setup* estimat i *Release* estimat. El missatge de *Setup* d'aquest esquema té l'*offset* i la duració de la ràfega. La reserva i alliberació de

recursos es calculen amb les dades anteriors. Qiao i Yoo proposaren un exemple d'aquest esquema, l'esquema *Just-Enough-Time* (JET) [2].

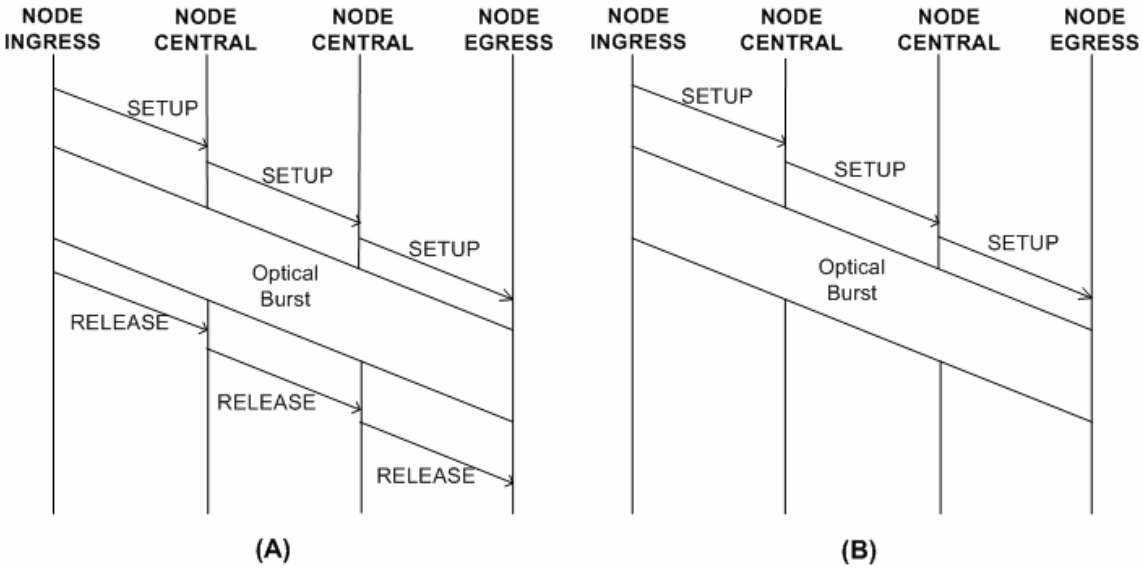



**Fig. 1.2.** A) Senyalització de *Setup* i *Release* explícites. B) Senyalització de *Setup* explícita.

El protocol JIT és molt més simple que el JET o l'Horizon, ja que no implica una planificació o algoritmes de càlcul complexes. Per tant, el JIT és molt més fàcil de ser implementat en hardware. La diferència entre el JET i el JIT resideix en la idea de com el JET intenta estimar quan la ràfega de dades serà rebuda, i per tant, no reserva recursos fins aquell instant. Això millora l'eficiència i el rendiment de la xarxa quant a la utilització del canal, però pot portar a que també es produeixin pèrdues si l'instant no es calcula de forma correcte. Per altra banda, el JIT reserva recursos en l'instant en que el paquet de control de *Setup* es rep. Aquest mètode disminueix el rendiment de la xarxa, però per contra és més fàcil d'implementar.

## 1.4. Estat de l'Art d'OBS

Fins a l'actualitat, la recerca en el camp de les *Optical Burst Switching Networks* s'ha concentrat més en les propietats lògiques d'OBS, sense considerar l'impacte subjacent de les tecnologies òptiques i fotòniques que haurien de formar aquest tipus de xarxes. En aquest camp s'identifiquen dues àrees on intensificar la recerca. Per exemple, no hi ha treballs publicats sobre l'impacte de la conversió d'ones de rang limitat en OBS. Totes les arquitectures OBS assumeixen les capacitats de conversió de longitud d'ona des de qualsevol longitud d'entrada a qualsevol longitud de sortida. Per al cas de les Xarxes d'Encaminament per Longitud d'Ona (WRNs), Yates realitzà un estudi sobre els efectes d'aquest tipus de conversió [13]. S'haurien de traslladar alguns d'aquests anàlisis al camp d'OBS.

Un altre camp és el de les tecnologies de transmissió fotòniques. S'han realitzat alguns treballs sobre l'efecte de guany i saturació dels nivells de potència en fibra dels *Erbium Doped Fibre Amplifiers* (EDFA) al transportar tràfic a ràfegues [14]. Com que les xarxes OBS haurien de transportar aquesta mena de tràfic, s'haurien de realitzar anàlisis similars i simulacions de com els EDFAs reaccionen al tràfic a ràfegues que ha estat modelat per a una xarxa OBS.

Un altre aspecte important sobre el que no s'han realitzat grans estudis és sobre els efectes dels protocols de més alt nivell que les xarxes OBS podrien portar. Per exemple, com a protocol *end-to-end*, IP sol portar fluxos TCP.

Pel que fa a la construcció de maquetes de xarxes OBS, ens els darrers anys s'ha progressat en aquest tipus de recerca, i s'han creat alguns prototipus i realitzat algunes demostracions de prova de concepte. Per exemple, el hardware JITPAC [15], desenvolupat pel MCNC-RDI, implementa el protocol de senyalització JIT i els autors en [16], presenten el disseny i implementació del protocol JET.

# CAPÍTOL 2. SISTEMES DE TEMPS REAL

El disseny del sistema i del programa de control utilitza moltes idees i característiques dels Sistemes de Temps Real (RTOS, *Real Time Operating Systems*). Per a això, en aquest capítol es presenta una breu introducció a aquests tipus de sistemes que ens servirà de base per a comprendre millor el disseny i la implementació que presentarem en els següents capítols. En l'ANNEX B es descriuen algunes de les característiques del processador PowerPC 405, d'IBM, [17] sobre el que es realitzarà la posterior implementació, que permeten l'operació amb Sistemes de Temps Real.

## 2.1. Introducció

Les tasques de processament en els sistemes de temps real generalment tenen restriccions temporals, les quals especifiquen quan les operacions comencen, quan temps prenen, i quan finalitzen.

En la programació en aquests sistemes, "temps real" significa que un programa ha de respondre als esdeveniments en el seu entorn en un temps de finalització específic. Aquests sistemes basats en esdeveniments (*event-driven*) es poden caracteritzar en termes de latència. En aquest cas la latència es defineix com l'interval de temps entre que un esdeveniment sorgeix fins que el sistema pren una acció en resposta a aquest esdeveniment.

En els sistemes de temps real es demana un límit superior de latència, també anomenat *scheduling deadline*. Aquests sistemes es poden dividir en dos grups:

- *Hard real-time*: el sistema ha de complir el *schedulling deadline* sempre. Si falla en complir aquest límit, el sistema podria sofrir conseqüències catastròfiques, o inclòs, que el sistema es quedés "penjat". Per exemple, un algoritme de control d'altitud en un avió és un sistema crític.

- *Soft real-time*: en aquest cas el *schedulling deadline* és més un objectiu que un requeriment absolut. S'espera que el sistema compleixi el temps límit la majoria de les vegades, però si en alguna ocasió en particular no succeeix això, el sistema pot seguir funcionant en normalitat. Aquest seria el nostre cas. Si no s'aconsegueix processar una reserva en el temps necessari, no s'ha de bloquejar el sistema, sinó que s'ha de seguir processant la resta de peticions i tasques.

## 2.2. Conceptes de Sistemes de Temps Real

A continuació s'introdueixen alguns conceptes importants utilitzats en la definició dels sistemes de temps real.

### 2.2.1. Polling vs. Interrupcions

Hi ha dues formes fonamentals per a que un programa en un entorn de temps real respongui als esdeveniments. El primer és el *polling* (sondeig), del qual el següent codi n'és un exemple.

El programa comença amb algun tipus d'inicialització i llavors entra en un *loop* infinit que comprova cada possible esdeveniment als quals el sistema ha de respondre. Per a cada esdeveniment configurat, el programa invoca la funció apropiada per a servir tal esdeveniment.

```
int main (void)
{
      sys_init();
      while(TRUE)
      {
            if(event_1)
                  service_event_1();
            if(event_2)
                  service_event_2();
            .
            .
            .
            If(event_n)
                  service_event_n();
      }
}
```

Aquest sistema és molt fàcil d'implementar i molt adequat per a sistemes petits on els requeriments de temps de resposta siguin lleugers. Però presenta alguns problemes obvis:

- El temps de resposta a un esdeveniment varia àmpliament depenent del punt on el programa es troba en el *loop* quan succeeix l'esdeveniment.
- El temps de resposta és funció del nombre d'esdeveniments que succeeixen en un mateix moment i conseqüentment són servits en la mateixa passada de *loop*.
- Tots els esdeveniments es tracten amb la mateixa prioritat.
- Quan nous esdeveniments s'afegeixen al sistema, el *loop* s'allarga, i per tant, augmenta també el temps de resposta.

El segon mètode utilitza les interrupcions, i és molt més eficient, a la vegada que també més difícil de programar. La idea d'interrupció és que l'ocurrència d'un esdeveniment "interromp" el flux actual d'execució d'instruccions, i invoca un altre corrent d'instruccions que serveix l'esdeveniment. Quan el servei s'ha completat, el control retorna a on el corrent d'instruccions original va ser interromput (veure Fig. 2.1). El corrent d'instruccions que serveix un esdeveniment s'anomena Rutina de Servei a la Interrupció (ISR, *Interrupt Service Routine*).

En la majoria de processadors actuals es distingeixen tres tipus d'interrupcions:

- La instrucció INT. Es comporta com una crida de subrutina. Aquesta forma d'interrupció és asíncrona respecte a l'execució d'instruccions.

- Excepcions del Processador. Són condicions de fallida, com la divisió per 0 o la referència il·legal a memòria. També és asíncrona respecte a l'execució d'instruccions.

- Interrupcions generades per esdeveniments externs al processador. Aquestes són generades per hardware d'entrada/sortida i succeeixen asíncronament respecte a l'execució d'instruccions.

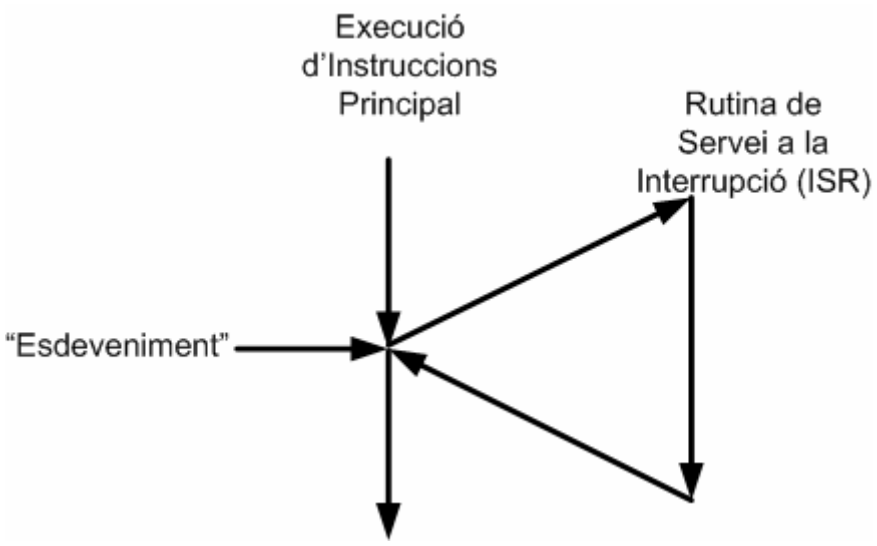



**Fig. 2.1.** Procés d'atenció a l'excepció.

En la pràctica, la majoria de sistemes incorporen un perifèric especialitzat anomenat Controlador d'Interrupcions que s'encarrega de manegar en detall les interrupcions del sistema. Alguns d'aquests controladors proporcionen mecanismes per a prioritzar les interrupcions que són més importants, de forma que dispositius crítics tenen major preferència en front a dispositius menys importants.

Així mateix, les interrupcions poden habilitar-se o deshabilitar-se a voluntat, ja sigui mitjançant l'accés directe a les instruccions que ofereixi el processador, o mitjançant les funcions que proporciona el controlador d'interrupcions. Aquesta funcionalitat és molt important a l'hora de dissenyar i implementar el software de temps real.

## 2.3. Sistemes de Temps Real Híbrids

Segons [18] tenim molts i diferents tipus de sistemes que actuen segons el paradigma *real time*. Però el tipus de sistema que potser millor s'ajusta als requeriments del nostre programa de control és un sistema de tipus híbrid. Aquests sistemes inclouen interrupcions que s'activen tant de forma periòdica, o semiperiòdica, com de forma totalment esporàdica, i són molt corrents en aplicacions incrustades (*embedded*).

Les interrupcions esporàdiques poden utilitzar-se per a manegar errors o tasques crítiques que requereixen una atenció immediata, i per tant tenen la prioritat més alta.

Un tipus de sistema híbrid molt comú en sistemes operatius comercials és la combinació del sistemes *round-robin* i *preemptive*. En aquests sistemes, les tasques de major prioritat poden sempre prevaler sobre les de menor prioritat. No obstant, si dues o més tasques de la mateixa prioritat estan preparades per executar-se simultàniament, llavors s'executen en *round-robin*.

De forma resumida, els sistemes que només tracten interrupcions solen tenir temps de resposta ràpids perquè el procés de planificació pot realitzar-se via hardware. Aquests sistemes són un cas particular dels sistemes *foreground/background*, que també són molt comuns en els sistemes incrustats.

Una debilitat dels sistemes que tracten únicament interrupcions és el temps gastat en el *loop jump-to-self* i la dificultat per proveir serveis avançats. Aquests serveis poden incloure la programació de *drivers* de dispositius i interfícies a xarxes. Una altra feblesa és la vulnerabilitat a malfunctionaments provocats per variacions en les temporitzacions, condicions no anticipades, fallides de *hardware*, etc.

### 2.3.1. Sistemes Foreground/Background

Els sistemes *Foreground/Background* són una millora respecte als sistemes que sols tracten interrupcions, ja que el *loop* de *polling* (sondeig) es substitueix per un codi que realitza algun processament útil. Aquests sistemes són els més comuns en les aplicacions incrustades. La següent figura Fig. 2.2 exemplifica esquemàticament aquest tipus de sistemes.

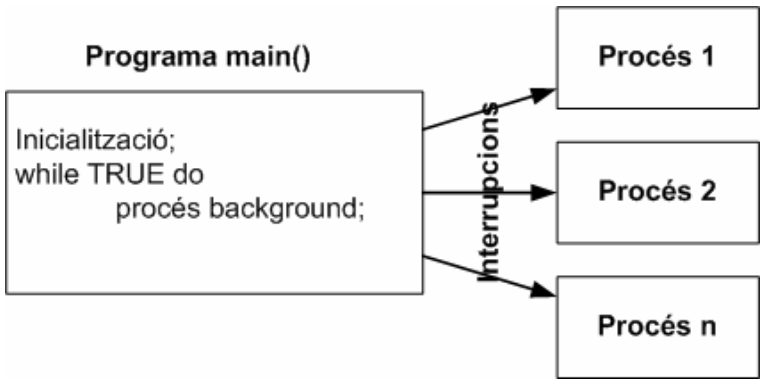



**Fig. 2.2.** Algoritme a alt nivell d'un sistema foreground/background**.**

Aquests sistemes involucren un conjunt de processos no activats per interrupcions que conformen el *background*. Les tasques del *foreground*

s'executen en qualsevol dels tres mètodes: *round-robin*, *preemptive priority* o híbrida. La tasca que actua en *background* ha de cedir l'execució a qualsevol altre tasca *foreground*, i d'aquesta forma, representa la tasca de menor prioritat en el sistema.

#### *2.3.1.1. Processament background*

El processament de *background* és la tasca que no manega les interrupcions, i normalment inclou qualsevol funció que no és crítica en el temps. Si en el sistema no sorgeix cap excepció, aquesta tasca hauria d'executar-se el 100% del temps.

Per a controlar el correcte funcionament d'aquest sistema, és comú, per exemple, incrementar un comptador en el *background* per tal de proveir una mesura de la càrrega de temps del procés o per a detectar si cap procés *foreground* es queda penjat. És desitjable, també, proveir de comptadors individuals cadascun dels processos *foreground*, els quals es reinicialitzen en aquests processos. Si el procés de *background* detecta que algun dels comptadors no està sent reinicialitzat molt sovint, es pot assumir que la tasca encarregada de reiniciar-lo no s'està executant, i per tant, que sigui un indici de fallida.

En aquest procés també poden realitzar-se certs tipus de comprovacions de baixa prioritat. Per exemple, en molts sistemes, es realitza un test complet del conjunt d'instruccions de la CPU. Aquesta mena de test no s'hauria de realitzar mai en els processos *foreground*, ja que sovint incorporen la comprovació dels controladors d'interrupcions, i per tant, podrien afectar el procés *foreground*.

#### *2.3.1.2. Inicialització*

La inicialització del sistema *foreground/background* la conformen els següents passos:

1. Deshabilitar les interrupcions.
2. Configurar el vector i piles d'interrupcions.
3. Realitzar l'autocomprovació.
4. Realitzar la inicialització del sistema.
5. Habilitar les interrupcions.

La inicialització és de fet la primera part del procés *background*. És important deshabilitar les interrupcions perquè molts sistemes arrenquen amb les interrupcions habilitades. La configuració consisteix en la inicialització de les adreces del vector d'interrupcions, configurar les piles si es tracta d'un sistema de nivells múltiples d'interrupcions, inicialitzar qualsevol tipus de dades, comptadors, *arrays* i altres. A més, és necessari realitzar algun tipus de test per autodiagnosticar el sistema abans d'habilitar definitivament qualsevol interrupció. Un cop tot això s'ha fet, el processament de temps real comença.

#### *2.3.1.3. Operació de temps real*

L'operació de temps real o *foreground* en el sistema *background/foreground* és el mateix que el d'un sistema que només tracta interrupcions. En l'ús de les instruccions per a habilitar i deshabilitar les interrupcions s'assumeix que un cop es rep la interrupció, la CPU manté la resta d'interrupcions fins que amb la instrucció EPI (*Enable Process Interruption*), es torna a rehabilitar de forma explícita el tractament d'aquestes.

En termes de commutació de contextos, és necessari guardar els registres generals en la pila. El procés de *foreground* s'executa fins que es completa, i per tant, els seus contextos no tenen perquè ser guardats. El següent codi en ensamblador mostra la inicialització d'un sistema *foreground/background*.

```
DPI                     ; deshabilitar interrupcions
STORE &handler,5        ; posar el handler d'interrupció en 5
EPI                     ; habilitar interrupcions
```

Altres tasques d'inicialització no estan representades en el codi anterior, però s'han de realitzar també. El codi per a guardar i restaurar els registres quan salten els *handlers* d'interrupcions seria com el que es mostra a continuació:

```
DPI                     ; deshabilitar les interrupcions
STORE R0,&reg0          ; guardar registre 0
STORE R0,&reg1          ; guardar registre 1

JU @APP                 ; executar programa de temps real
LOAD R7, &reg1          ; restaurar registre 1
LOAD R7, &reg0          ; restaurar registre 0

EPI                     ; habilitar interrupcions
RI                      ; retorn de la interrupció
```

El programa *background* ha d'incloure el procediment d'inicialització i qualsevol procés que no sigui crític. El seu programa en "C" seria:

```
/* assignar espai disponible per als contextes */
int reg0, reg1;

/* declarar altres variables globals */

void main (void)
{
      init();           /* inicialitzar el sistema */

      while(TRUE)       /* loop background */
            background();     /* processament no temps real */
}
```

Els sistemes *foreground/background* típicament tenen bons temps de resposta, ja que gran part de la planificació es duu a terme amb hardware. A més, aquests tipus de sistemes s'implementen millor quan el nombre de tasques *foreground* es coneix a priori i no varia en l'aplicació (com previsiblement serà el nostre cas, veure CAPÍTOL 3).

# CAPÍTOL 3. DISSENY DEL PLA DE CONTROL

Un cop plantejats els objectius i les bases de la tecnologia es pot passar a descriure el disseny del Pla de Control OBS. El disseny s'ha de basar en la reutilització del sistema tant per a nodes que actuen en la frontera, com en el nucli de xarxa. En el present capítol es presenta la maqueta de xarxa sobre la que el pla de control estarà funcionant, així com els fonaments i disseny del programa de control que l'anomenarem OCPS, de *OBS Control Plane Software* (Aplicatiu del Pla de Control OBS).

## 3.1. Introducció

A l'hora de crear un sistema en temps real que sigui veritablement funcional cal realitzar un disseny que permeti que la seva posterior implementació ofereixi:

- Efectivitat: el sistema ha de realitzar les tasques que li han estat encomanades. A partir de certes variables d'entrada, el sistema ha de ser capaç de processar les dades i oferir els valors de sortida correctes.

- Eficiència: la forma en que es realitza aquest processament de les dades sigui la que faci un ús dels recursos millor i de forma més eficient. Això permet, de forma indirecta, que la resta de processos/sistemes integrats tinguin més marge i recursos per a poder desenvolupar les seves tasques.

Les anteriors dues premisses han d'aplicar-se al present disseny del Pla de Control, el qual ha d'executar-se sobre els nodes d'una maqueta de xarxa com la que s'especifica en el següent apartat.

## 3.2. Especificació de la maqueta de xarxa

El Pla de Control que es planteja en el present PFC és part del disseny i implementació d'una maqueta de xarxa OBS. Per a que la maqueta de xarxa funcioni correctament, apart del Pla de Control també s'ha d'implementar un Pla de Dades que permeti realitzar l'ensamblatge de les ràfegues i la transmissió d'aquestes per via òptica als diferents nodes de la xarxa. El present projecte sols es centra en la definició de la part de control.

La maqueta de xarxa es basa principalment en el desplegament de diferents nodes que puguin realitzar les funcions tant de node frontera com de nucli. Específicament en la maqueta es distribuiran tres nodes, tal com es mostra en la figura Fig. 3.1: dos que sols realitzaran les funcions de node frontera, i un tercer node que actuï tant de node frontera com de nucli. El mateix Pla de Control ha de poder executar-se sobre qualsevol d'aquests nodes.

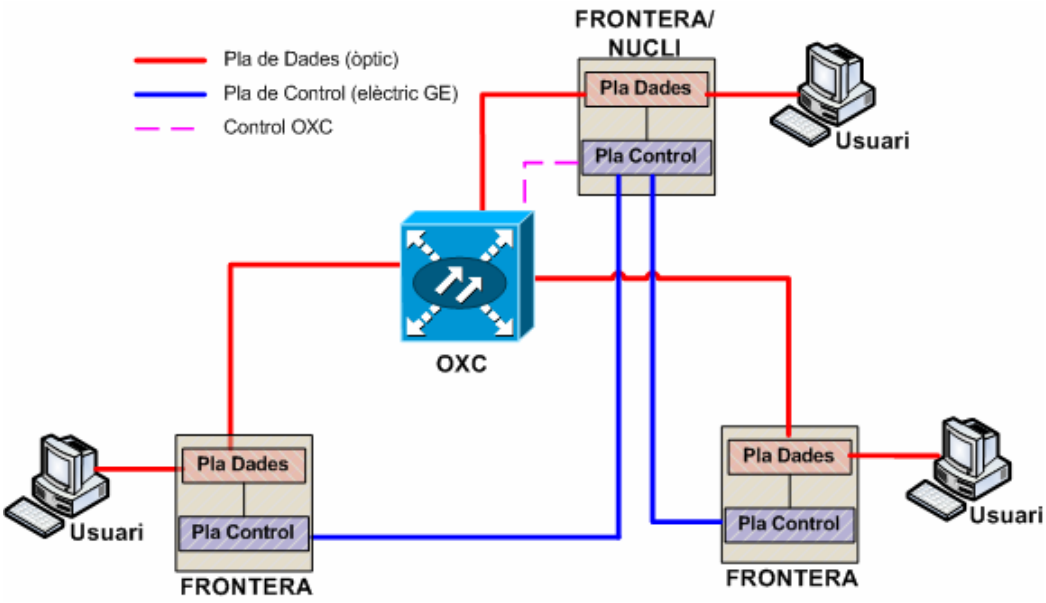



**Fig. 3.1.** Maqueta de la xarxa OBS.

En el centre de la maqueta tenim el commutador òptic el qual permet realitzar la commutació de les ràfegues entre els diferents nodes que la componen. En el nostre cas, per tal d'oferir el major rendiment possible, el commutador és de tipus opto-elèctric, del fabricant CIVCOM [19].

Els commutadors opto-elèctrics utilitzen un material substrat molt birefringent i camps elèctrics per a redireccionar la llum d'un port a un altre. Un dels materials més utilitzats en aquests commutadors és el niobat de liti. El camp elèctric canvia l'índex de refracció del substrat, el qual manipula la llum a través del camí de guia d'ona apropiat al port de sortida desitjat. Aquest tipus de commutador són ràpids i fiables, però tenen unes altes pèrdues d'inserció i certa dependència a la polarització.

## 3.3. Requeriments i funcions del Pla de Control

El Pla de Control ha de realitzar les funcions pròpies de senyalització i reserva dels recursos de commutació, transmissió i recepció de ràfegues. Apart, també ha de realitzar funcions de resolució de la commutació de les ràfegues a executar, així com de l'encaminament dels paquets de control.

Les funcions poden diferenciar-se segons el tipus de node de la xarxa, tal com es mostra en la següent Taula 3.1.

**Taula 3.1.** Funcions del Pla de Control segons el tipus de node.

| Node Frontera | Node Nucli |
|---|---|
| ▪ Rebre i processar les peticions d'enviament de ràfega.<br>▪ Determinar el següent node per al paquet de control.<br>▪ Determinar el port de sortida de la ràfega òptica.<br>▪ Processar la recepció i transmissió dels paquets de control.<br>▪ Determinar la cua de sortida del desensamblatge de la ràfega.<br>▪ Senyalitzar al Pla de Dades per a l'enviament o recepció de ràfegues.<br>▪ Senyalitzar al OXC per a la commutació que permeti l'enviament o recepció de les ràfegues.<br>▪ Processar la reserva dels recursos de recepció o transmissió.<br>▪ Càlcul inicial de l'*offset*. | ▪ Rebre, processar i retransmetre els paquets de control.<br>▪ Processar la reserva de recursos de commutació.<br>▪ Càlcul del nou *offset*.<br>▪ Determinar el següent node per al paquet de control.<br>▪ Determinar el port de sortida i commutació de la ràfega.<br>▪ Senyalitzar al OXC per a la commutació de la ràfega. |

Com es pot veure en l'anterior taula, les funcions entre node frontera i node central són molt semblants. Els nodes frontera, però, han de manegar les peticions de transmissió i recepció de ràfega, i realitzar el càlcul inicial de l'*offset*. És previsible que aquestes tasques afegeixin temps de processament extra.

## 3.4. Disseny del Pla de Control

Les tasques que el Pla de Control ha de realitzar, com s'ha vist en l'anterior apartat, són diverses, i requereixen que es defineixi i dissenyi un marc i estructura sobre el qual les diferents funcions i processos de comunicació entre nodes sigui possible. Per a això, es divideix el disseny en els següents subapartats.

### 3.4.1. Definició funcional

La funció principal a la que ha de fer front el Pla de Control, i el nostre programa OCPS, és la reserva dels recursos, tant per a la transmissió i recepció, en els nodes frontera, com per a la commutació en els nodes centrals a la xarxa OBS.

Funcionalment el disseny de com es realitza el seguiment dels recursos reservats i no reservats és el següent.

- El temps està dividit en ranures temporals.

- L'algoritme del sistema es basa en el recorregut d'un "vector" ranurat de recursos, o SRV (en anglès l'anomenarem *Slotted Resource Vector*).

- El SRV té una longitud finita i es comporta com un *buffer* circular, és a dir, després de l'última posició o ranura, la següent és la posició "0" ó inicial del vector.

- La grandària de les ranures, en temps, ve determinat per un comptador/temporitzador de temps prefixat a l'inici del programa, i que s'autocarrega un cop arriba a "0".

- El valor del temporitzador depèn bàsicament de dos paràmetres (encara que la seva configuració és lliure de tenir qualsevol altre valor):
    - Del període de commutació físic del commutador òptic. En aquest cas, el valor del temporitzador no hauria de ser major al temps de commutació, ja que sinó es perdria certs graus de rendiment que ens ofereix el OXC.
    - De la rapidesa en la que la informació i les tasques es processen en el OCPS.
    - Segons el valor del temporitzador, és d'esperar que el rendiment de la reserva de ranures sigui diferent.

- Els nodes, mitjançant els paquets de control de *Setup* o per petició pròpia del Pla de Dades (quan es té una ràfega preparada i es desitja transmetre-la) realitzen la reserva de ranures en el SRV.

Segons els tipus de recursos que té el node OBS, la reserva de recursos presenta diferents possibilitats. Així, si el Pla de Control controla un commutador òptic de només dos canals, pot determinar que només hi ha dues configuracions possibles d'aquest, i per tant, el commutador o està "creuat", o "pla". Si la commutació de la ràfega requereix sempre el mode creuat, llavors, si aquesta ja està ocupada per una anterior reserva, les peticions següents no podran realitzar-se i s'hauran de descartar. Però també hi hauria la possibilitat que amb una sola configuració del commutador òptic, el nombre de reserves fos major d'un, ja que la configuració és correcte per a realitzar més d'una commutació (veure Fig. 3.2).

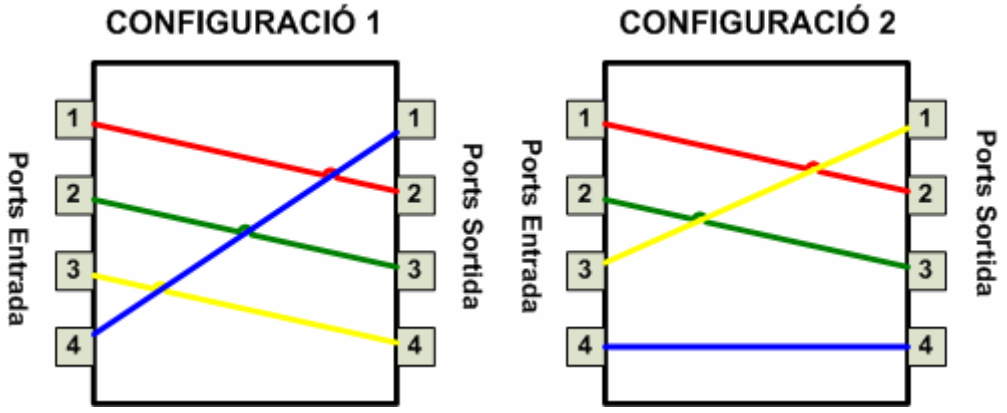



**Fig. 3.2.** Dues configuracions possibles d'un commutador 4x4.

Per exemple, en la primera configuració de la Fig. 3.2, les ràfegues que requereixin una commutació entre els ports d'entrada 1 i 2, amb ports de sortida 2 i 3, podran seguir mantenint-se a pesar de variar, per exemple, la configuració dels altres dos ports, com es mostra en la segona configuració.

En el disseny del node (en una primera fase) s'ha realitzat el càlcul de les reserves de forma que no es concep recepció i transmissió en el node OBS al mateix temps per a facilitar-ne el disseny i l'algoritme d'assignació de les reserves en el SRV. Es representa aquesta possibilitat en els següents esquemes de xarxa (veure Fig. 3.3 i Fig. 3.4).

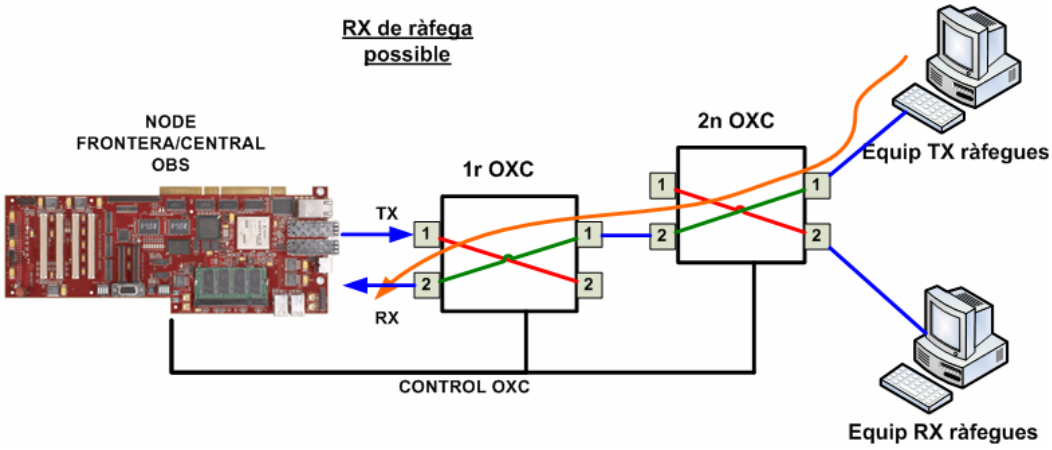



**Fig. 3.3.** Configuració RX en els OXCs.

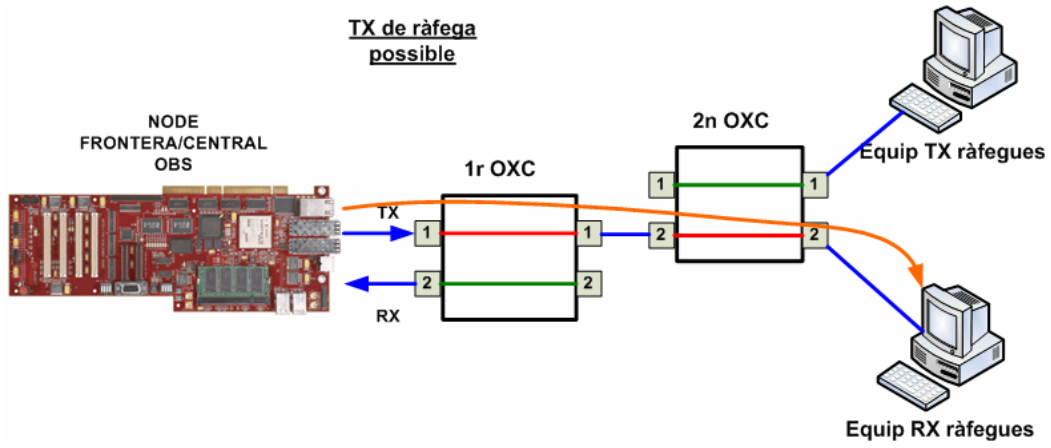



**Fig. 3.4.** Configuració TX en els OXCs.

Com es pot veure, per tal que el node integrat en la placa pugui rebre ràfegues de l'equip client transmisor requereix una configuració diferent dels commutadors òptics que la necessària per a transmetre ràfegues a l'equip receptor d'aquestes.

#### *3.4.1.1. Exemple en la senyalització per a la transmissió de ràfegues*

La següent figura Fig. 3.5 mostra un exemple de com es tractarien les peticions de transmissió de ràfega des d'un node frontera. Mentre el temporitzador global està en marxa, el SRV va actualitzant la seva posició, i el sistema actua segons els paràmetres definits en la ranura.

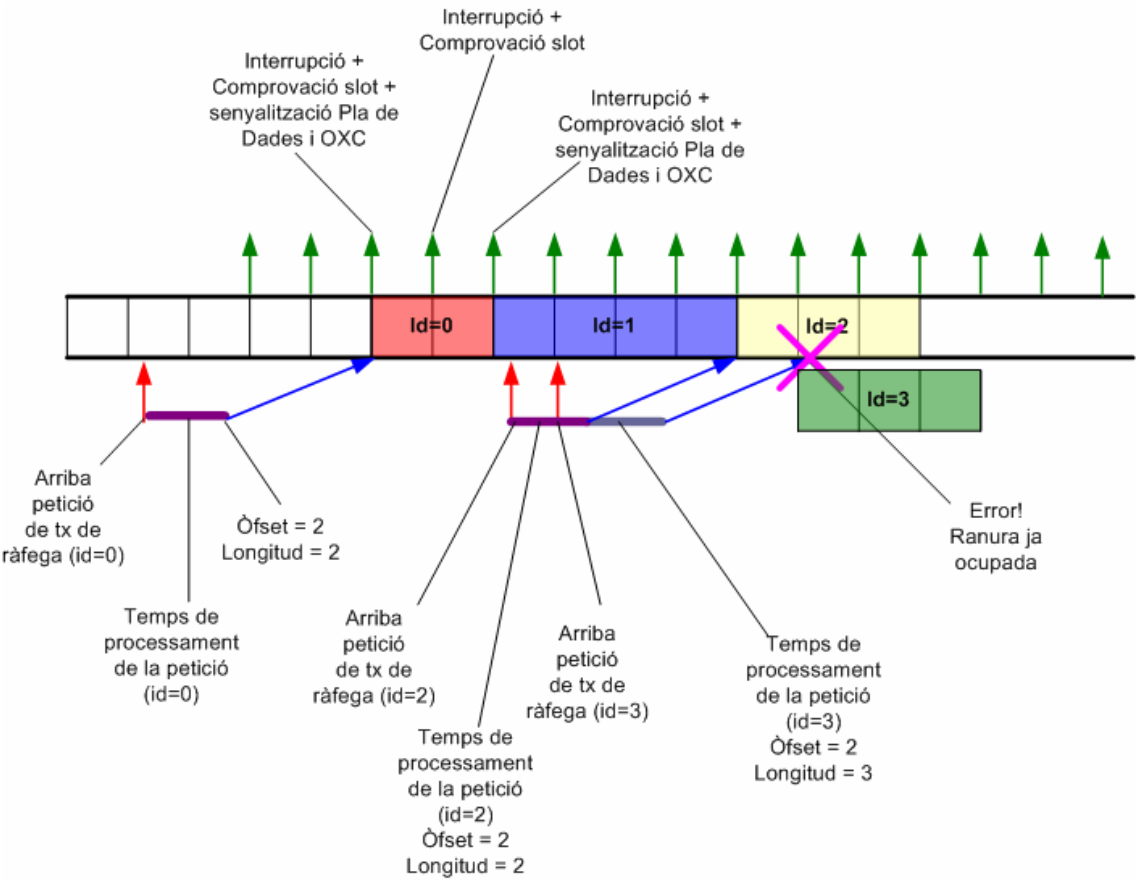



**Fig. 3.5.** Petició i transmissió de ràfegues.

En el cas concret de transmissió de ràfega, en el procés, a grans trets (s'especifica en l'apartat F.1.3 el flux d'aquest cas d'ús), es processa la recepció de la petició, primer de tot calculant un valor d'*offset*. Segons aquest valor d'*offset*, es comprova en el SRV si hi ha recursos disponibles des de l'*offset*, i per a una durada igual a la longitud de la ràfega en nombre de ranures. Si hi ha recursos disponibles, llavors es realitza la reserva. Això és, es marquen les diferents ranures que componen la longitud de la ràfega amb la informació necessària per a que posteriorment es processi la transmissió.

Per exemple, en el cas concret mostrat en la Fig. 3.5, les reserves de les ràfegues amb *id=0* i *id=2* es realitzen correctament, però en canvi, quan es processa la ràfega amb *id=3* es calcula l'*offset*, però justament, aquest concorda amb una ranura que ja està ocupada per la ràfega amb *id=2*, i per tant, la reserva no és possible.

### 3.4.2. Definició del Protocol

L'operació normal del Pla de Control, juntament amb els diferents protocols de reserva, requereix d'un protocol de més baix nivell que permeti l'intercanvi d'informació entre els nodes. Per a això, es necessita la definició d'un protocol de missatges, els quals porten la informació necessària per a realitzar la reserva de recursos al llarg del camí entre origen i destinació de les ràfegues, dins del domini OBS.

En la maqueta que s'ha definit en 3.2, la tecnologia de capa d'enllaç que s'utilitza per a l'intercanvi de dades és el Gigabit Ethernet [20]. Per tant, tots els datagrames definits en la nostra capa de protocol OBS aniran encapsulats en trames d'Ethernet. El format dels paquets de control, encapsulats en les seves corresponents trames d'Ethernet, és el que es mostra en la següent figura Fig. 3.6. El contingut i funció dels diferents camps de la capçalera de trama Ethernet es descriu en la Taula 3.2. I en la Taula 3.3 es descriuen els camps dels nostres paquets de control.

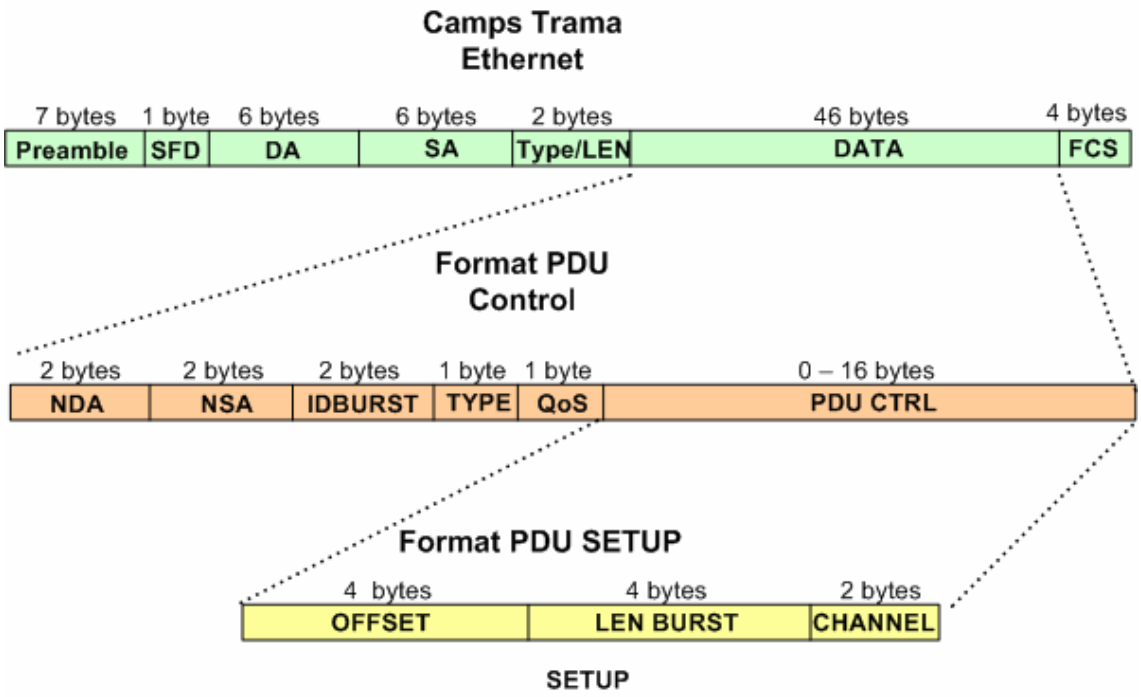



**Fig. 3.6.** Formats i capçaleres de la trama Ethernet i dels Paquets de Control. Exemple de PDU tipus SETUP.

**Taula 3.2.** Camps de les capçaleres de les trames Ethernet.

| Camp | Long (Bytes) | Descripció | Funció |
|---|---|---|---|
| Preamble | 7 | Preamble | S'utilitza per a la sincronització digital entre les diferents NICs Ethernet. És un camp amb '1's i '0's de forma alternada. |
| SFD | 1 | *Start-of-Frame Delimiter* | Aporta la informació necessària per a identificar el començament d'una trama |

| | | | |
|---|---|---|---|
| | | (Delimitador Començament de Trama) | Ethernet. És una plantilla alternativa d'1's i '0's, acabant amb dos '1' consecutius. |
| DA | 6 | Adreça Destinació | Identifica l'estació o NIC que ha de rebre la trama. |
| SA | 6 | Adreça Origen | Identifica l'estació o NIC que ha generat la trama. |
| Type/LEN | 2 | Tipus/Longitud | Aquest camp indica tant el número de bytes que estan continguts en el camp de dades de la trama, o el tipus de trama, si ha estat ensamblada utilitzant un format definit. |
| DATA | 46 | Dades | Són una seqüència de bytes que componen les dades que es volen transmetre en la trama. El camp ha de ser com a mínim de 46 bytes, per a que la longitud total de la trama Ethernet no sigui menor a 64 bytes. |
| FCS | 4 | *Frame Check Sequence* (Seqüència de Comprovació de Trama) | Aquest camp conté un *checksum* que permet revisar la integritat del paquet rebut per a ser entregat a les capes superiors o descartat. |

**Taula 3.3.** Capçalera i camps del paquet de control.

| Camp | Long (Bytes) | Descripció | Funció |
|---|---|---|---|
| NDA | 2 | Adreça Destinació de Node | Identifica l'adreça de destinació del node OBS en el domini OBS, i que és el node frontera de sortida de la xarxa. |
| NSA | 2 | Adreça Origen de Node | Identifica l'adreça d'origen del node OBS, que és el node frontera d'entrada a la xarxa. |
| IDBURST | 2 | Identificador de *Burst* (Ràfega). | Identifica una ràfega generada pel node frontera d'entrada al domini OBS. |
| TYPE | 1 | Tipus | Permet identificar el tipus de PDU de Control. Aquest camp permet definir diferents tipus de PDU, cadascun dels quals permeti realitzar les diferents funcions necessàries en una xarxa OBS, o suportar les diferents opcions dels protocols de reserva. |
| QoS | 1 | Qualitat de Servei | Permet definir bits extres segons si la ràfega té o no certs paràmetres de qualitat de servei que ha d'assolir. |
| PDU CTRL | 0 – 16 | PDU de dades de Control | Són les dades específiques del missatge de control segons el valor del tipus de PDU de control que es transmet. |

Com a exemple, s'ha definit els tipus de dades per a un paquet de control de tipus *Setup* (veure figura Fig. 3.6). Aquest tipus de paquet realitza la reserva de recursos al llarg del camí entre els nodes OBS origen i destinació del domini

OBS. Per tant, per a que la reserva dels recursos sigui possible, el Pla de Control dels diferents nodes ha de tenir informació relativa a les longituds, *offsets* i canals de transmissió de les ràfegues. La següent Taula 3.4 ofereix una breu descripció d'aquests camps.

**Taula 3.4.** Camps del PDU de Control de tipus SETUP.

| Camp | Long. (bytes) | Descripció | Funció |
|---|---|---|---|
| OFFSET | 4 | Òffset | És l'*offset* calculat en el node frontera d'ingrés i recalculat en els nodes centrals. La longitud de 4 bytes identifica el nombre de ranures d'espera fins a la recepció de la ràfega en el node que processa aquest paquet de control. |
| LEN BURST | 4 | Longitud de *Burst* (Ràfega) | És la longitud de la ràfega, en nombre de ranures. |
| CHANNEL | 2 | Canal o Port | Identifica la longitud d'ona o port pel qual es rebrà la ràfega, i que funciona com si fos un etiquetatge, mitjançant el qual es realitza la commutació en el node central (o frontera, si també té OXC). |

En el cas concret dels camps OFFSET i LEN BURST, s'utilitza com a valor de resolució el nombre de ranures, però en d'altres implementacions, aquests camps podrien portar la quantitat de temps (per exemple en resolució de nanosegons) d'espera a la recepció de la ràfega, o la longitud en bytes de la ràfega, respectivament.

Altres tipus de paquets de control podrien ser els que s'especifiquen en la següent Taula 3.5. En aquesta es representa a mode orientatiu quin podria ser el valor del camp TYPE de la capçalera del paquet de control.

**Taula 3.5.** Altres tipus de paquets de control.

| Tipus | Valor TYPE | Descripció |
|---|---|---|
| SETUP | 0x01 | Realitza la reserva de recursos al llarg dels nodes que componen el camí entre origen i destinació en el domini OBS. |
| ACK | 0x02 | Si el mètode de reserva és de tipus *Two-Way*, és a dir, abans de transmetre la ràfega s'espera confirmació, aquest paquet de control serviria per indicar que la reserva ha estat satisfactòria. |
| NACK | 0x03 | Si el mètode de reserva és de tipus *Two-Way*, és a dir, abans de transmetre la ràfega s'espera confirmació, aquest paquet de control serviria per indicar que la reserva no ha estat possible, i es podria indicar la raó. |
| RELEASE | 0x04 | Aquest tipus de missatge s'utilitzaria quan el tipus de protocol de reserva fos de tipus *Explicit Release*, és a dir, que s'especifica de forma explícita l'alliberament del recurs. |

### 3.4.3. Càlcul de l'Offset

El càlcul de l'*offset* es pot realitzar seguint les indicacions comentades en l'apartat 1.3.1.1. Com ja s'ha esmentat anteriorment, hi ha dues formes principals per al càlcul d'aquest paràmetre: de forma fixe, seguint el nombre de salts que realitza la ràfega entre origen i destinació, i els temps deterministes de processament dels paquets de control; o de forma estadística, seguint els valors aportats per una variable aleatòria.

Si bé la opció idònia en el disseny seria permetre tant l'ús d'una forma o altra de càlcul de l'*offset*, a causa de que la nostra maqueta de xarxa OBS (veure Fig. 3.1) és petita, i el nombre de salts entre nodes és molt baix, el càlcul fix implicaria que en les proves de funcionament i rendiment de la maqueta, aquest càlcul fos sempre el mateix entre un origen i destinació. Com que aquest valor seria sempre el mateix, totes les ràfegues realitzarien la mateix petició d'*offset*, i per tant, al sobrecarregar el sistema (per avaluar-ne el rendiment), es produirien molts solapaments sol·licitant els mateixos recursos.

Per tot això, la opció escollida del càlcul de l'*offset* serà una barreja de les dues formes anteriors, de la següent forma:

- Primer es calcularà un cert període determinista en funció dels salts entre nodes i el temps de processament dels paquets de control, més el temps en senyalitzar la commutació al OXC.

- I segon, s'afegirà un valor aleatori de nombre de ranures, entre 2 i 30.

El càlcul fix s'obté de comptabilitzar:

- El nombre de nodes pels quals ha de passar la senyalització de control.
- El temps de transmissió del paquet de control al següent node.
- El temps de propagació del paquet al següent node.
- El temps de recepció del paquet de control.
- El temps necessari en l'atenció de les interrupcions.
- El temps de senyalització de la transmissió de la ràfega al pla de dades i de les senyalitzacions als commutadors òptics.
- El temps de transmissió de la ràfega per la interfície.
- El temps de propagació de la ràfega òptica (aproximat a 0).
- Altres temps que es vulguin afegir depenent de: polítiques de QoS, prioritats de serveis, ús de recursos, etc.

D'aquesta forma, s'obtindran en les proves peticions d'enviament de ràfegues amb valors d'*offsets* diferents i variats que permetran que la reserva de recursos sigui major per a taxes de generació de peticions elevades.

Aquest càlcul variable de l'*offset* es realitza abans de comprovar els recursos de transmissió del node frontera d'ingrés. Si la comprovació comporta que la reserva no sigui possible, es realitza un nou càlcul de l'*offset*, i es torna a comprovar (veure F.1.3). Així s'ofereixen majors possibilitats per a que la

ràfega pugui ser transmesa, a costa d'oferir un major retard en la seva transmissió.

En el cas de realitzar una commutació de ràfega es recalcula el valor de l'*offset* realitzant la resta al valor rebut pel paquet de control del temps d'espera en la cua de processament, més el temps de processament de la reserva de commutació. Posteriorment, aquest nou valor d'*offset* s'introdueix en el paquet de *Setup* que es propaga cap a la destinació (veure F.1.2).

### 3.4.4. Definició dels recursos

Com s'ha introduït en l'apartat 3.4.1, l'algoritme del programa es basa en el recorregut d'un vector de ranures o *slots* temporals, sobre els quals la reserva de recursos es duu a terme.

Aquest vector (SRV) es comporta com una estructura de dades, i anàlogament en llenguatge C, es podria definir de la següent manera:

```
/* sTimerFifo: Vector utilitzat per a l'emmagatzematge I reserve de
recursos */
typedef struct Timer_Fifo {
      Xboolean    Occupied;         /* XTRUE=Ocupat, XFALSE=LLiure */
      Xuint32     TypeOccupied;     /* 1=Slot Inicial, 2=Slot Final,
3=Slot, 4=Free */
      Xuint16     BurstID;          /* BurstID To Tx,XC or RX */
      Xuint32     TypeAction;       /* 1 = Tx, 2 = Rx, 3 = XC */
      Xuint32     OXCPosition;      /* 0 = through, 1 = cross */
      Xuint32     BFAddress;        /* Address to the Burst Data */
} sTimerFifo;
sTimerFifo TimerFifo[TIMER_FIFO_SIZE];
Xuint32 FIFO_head, FIFO_tail, FIFO_Count;
```

Tal com es mostra en el codi anterior, cada ranura del vector té informació relativa a:

- Si està o no ocupada.
- El tipus d'ocupació. En aquest cas diferenciem 4 possibles valors:
  - Ranura inicial de reserva de ràfega.
  - Ranura intermèdia de reserva de ràfega.
  - Ranura final de reserva de ràfega.
  - Ranura lliure (no ocupada). Aquest valor ha de ser coherent amb el valor de la variable *Occupied*.
- El tipus d'acció que es realitza amb la ràfega. També tenim en compte tres possibilitats: Recepció de ràfega, Transmissió de ràfega i Commutació de ràfega (es realitza principalment en nodes centrals a la xarxa).
- La posició en la que es troba el commutador òptic.
- Adreça de memòria on resideix la informació de la ràfega.

De l'anterior caracterització cal remarcar dos punts. El primer és el de la posició en la que es troba el commutador òptic. Aquest valor és important tenir-lo en compte si el commutador té varis ports d'entrada i sortida, ja que com s'ha

comentat en el punt 3.4.1, diferents configuracions poden realitzar una mateixa commutació entre un port d'entrada i sortida. Però per al cas concret d'un commutador 2x2, el nombre d'opcions es redueix a dos: o es commuta, o no es commuta.

El segon punt a remarcar és la diferenciació entre els tipus d'accions a realitzar amb la ràfega. S'ha de tenir en compte que si sols es realitza commutació, el programa de control sols ha de senyalitzar al OXC. En canvi, si es realitza tant transmissió com recepció, caldrà senyalitzar al Pla de Dades, i potser, segons la configuració del node (si és frontera+central), també al OXC.

La informació que es conté de les ràfegues a commutar, transmetre o rebre pot també definir-se en forma d'estructura de la següent forma:

```
typedef struct Burst_Fifo {
      Xuint16     BurstID;          /* BurstID */
      Xuint32     Offset;           /* Offset de la ràfega */
      Xuint32     BurstLength;      /* Longitud de la ràfega */
      sTimerFifo  *pTimer;          /* Punter al sTimerFifo */
      Xuint32     NumSlots;         /* Nombre de slots ocupats */
      Xuint32     realNumSlots;     /* Nombre real de slots ocupats */
      Xuint32     bfc;              /* Posició en vector BurstFifo */
} sBurstFifo;
sBurstFifo BurstFifo[BURST_FIFO_SIZE];
Xuint32 BurstFifoCounter[BURST_FIFO_SIZE];      /* 0 = lliure */
```

En l'anterior estructura es defineix un vector de ràfegues, i de forma paral·lela un vector d'enters per a poder realitzar una cerca ràpida en el *buffer* a l'hora de trobar alguna posició lliure per emmagatzemar la informació relativa a la ràfega, i fer la posterior reserva de recursos.

Així mateix cal diferenciar dos valors concrets dins l'estructura. Un és el *NumSlots*, que és en efecte el nombre de ranures corresponents a la longitud de la ràfega per a la qual necessita realitzar la reserva. Té una relació directe amb el camp *BurstLength*, tant si aquest ja porta el nombre de ranures de longitud, com si la longitud s'expressa en bytes, per exemple. Per altra banda, tenim el camp *realNumSlots*, el qual expressa el nombre real de ranures que es necessiten reservar.

Aquesta diferenciació radica en què en la nostra implementació, per tal d'estalviar-nos errors en l'emmascarament de reserves en una mateixa ranura, el nombre de ranures dels quals es peticiona és sempre *NumSlots* + *2*. Això és així perquè afegim a l'inici de la ràfega i al final, una ranura "de guarda", la qual ens permet que no se'ns solapin reserves en cas que el programa trigui en processar la informació.

### 3.4.5. Definició de la commutació de ràfegues i encaminament de paquets de control

En una xarxa OBS cal diferenciar dues funcionalitats, associades cadascuna a un dels dos plans, dades i control. Per un costat cal definir l'encaminament dels

paquets de control, és a dir, trobar una ruta que compleixi certs requeriments, com ample de banda, menor nombre de salts, etc. Pel concepte d'encaminament, aquest càlcul del següent node en el camí es realitza node per node, i per tant, el node origen no sap a priori quina ruta seguiran els paquets de control.

Per altra banda, associat al pla de dades, es realitza la commutació de les ràfegues. Segons el port d'entrada pel qual arriba la ràfega es commuta a un cert port de sortida. Aquest procés, per a que ofereixi el major rendiment possible, ha de ser el més ràpid possible.

Aquestes dues funcionalitats es veuen completament suportades en el disseny del protocol introduït en l'apartat 3.4.2.

#### *3.4.5.1. Encaminament dels paquets de control*

Els nodes OBS tenen assignats adreces lògiques (camps *NDA* i *NSA*). Mitjançant aquest adreçament es poden confeccionar taules d'encaminament en els diferents nodes, i per tant, quan un d'aquests processa un paquet de control, sap a on ha d'encaminar el seu paquet de control en el camí fins al node OBS destí, on es desensamblarà la ràfega.

Com que a nivell d'enllaç s'utilitza Gigabit Ethernet, el paquet de control s'ha d'enviar a la corresponent adreça física de tarja, i per tant, les taules d'encaminament també han de contenir aquesta informació. La Taula 3.6 presenta la relació entre els diferents camps.

**Taula 3.6.** Camps de les taules ARP i d'Encaminament.

| Taula | Camp | Descripció |
|---|---|---|
| ARP | Adreça Ethernet | Identifica l'adreça Ethernet per a l'enviament del paquet de control del següent node que ha de processar la reserva. |
| | Adreça Node OBS | Identifica l'adreça del node frontera destí de les ràfegues. Per tant, per a la present adreça OBS, hi haurà una adreça Ethernet corresponent a qui enviar el següent paquet de control. |
| Encaminament | Adreça IP | És l'adreça IP de destinació, si el que s'envien encapsulades en ràfegues són datagrames IP. |
| | Cua | Per a cada adreça IP de destí, el node frontera té una cua on es guarden les ràfegues ensamblades. Aquesta variable està relacionada amb l'anterior camp. |
| | Adreça OBS | Identifica l'adreça OBS del node frontera de destinació, on es realitzarà el desensamblatge de les ràfegues en datagrames IP. |
| | Canal/Port | Identifica el Port o Canal de sortida en el commutador o node, pel qual s'ha de transmetre o commutar la ràfega. |

L'actualització d'aquestes taules les ha de portar a terme un procés específic d'encaminament. Per a això, i per a simplificar el primer disseny, aquestes taules s'emplenen de forma estàtica.

#### *3.4.5.2. Commutació de les ràfegues*

Per altra banda, la commutació de les ràfegues es realitza amb la informació continguda en el camp CHANNEL dels paquets de control de tipus *Setup*. Aquest camp funciona com si es tractés d'un etiquetatge, tal com el que es defineix en la tecnologia *Multi Protocol Label Switching* (MPLS) [21]. Juntament amb els valors de les variables que indiquen la configuració de commutació en el OXC, permeten identificar per a un port o canal d'entrada de la ràfega, el port o canal de sortida, en funció del valor de la variable *OXCPosition* de la ranura que s'està processant en el SRV.

Igualment també hi ha d'haver unes taules que identifiquin quines commutacions es realitzen segon l'ID de configuració en el OXC. La configuració d'aquestes taules i el seu corresponent farciment es realitza de forma estàtica al configurar el OXC associat al node pel qual s'instal·la el programa de control.

Desglossant el disseny, es tenen els següents camps d'informació referents a les commutacions (veure la Taula 3.7).

**Taula 3.7.** Camps d'informació per a les taules de commutació.

| Camp | Descripció | Funció |
|---|---|---|
| InPort | Port/Canal d'Entrada | Identifica un port/canal d'entrada d'una commutació possible. |
| OutPort | Port/Canal de Sortida | Identifica un port/canal de sortida d'una commutació possible. |
| IdSw | Identificador de configuració InPort-OutPort | Atorga un identificador a una commutació específica entre un port/canal d'entrada i un port/canal de sortida. |
| IdConf | Identificador de Configuració del OXC | Atorga un identificador a una configuració concreta de commutació entre tots els ports/canals d'entrada i tots els ports/canals de sortida. |

## 3.5. Algorítmica del programa

Un altre punt important en la definició del programa és determinar quin tipus d'algoritme seguirà, així com definir la prioritització relativa de les diferents tasques i funcions que duu a terme.

### 3.5.1. Prioritització de les tasques associades a les interrupcions

La següent llista, Taula 3.8, determina l'ordre de prioritats en el tractament de les tasques en el nostre sistema.

**Taula 3.8.** Escala de prioritats de les principals tasques del programa de control OCPS.

| Prioritat | Tasca | Descripció |
|---|---|---|
| 1 | Atenció a la interrupció del temporitzador de ranures | Aquesta és la tasca més prioritària, perquè determina la possibilitat de rebre, enviar o commutar recursos que prèviament ja han estat reservats, i per als quals ja s'han gastat temps de procés en els altres nodes de la xarxa, i en el node que la processa en particular. |
| 2 | Atenció a la interrupció de recepció de paquets de control. | Al rebre trames de control d'altres nodes se'ns indica que en el domini OBS, en part dels nodes, ja s'han realitzat algunes reserves, i per tant, s'estan consumint recursos. Per tant, abans de nosaltres poder generar trames de control, hauríem de poder processar les que ens arribin. |
| 3 | Atenció a la interrupció de recepció de peticions de transmissió de ràfega. | Un cop es té una ràfega ensamblada, el pla de dades indica al pla de control que necessita transmetre-la. Aquesta tasca s'ha deixat com la menys prioritària en la llista d'interaccions entre els dos plans. |
| 4 | Protocols d'encaminament i resolució d'adreces. | Per tal de poder determinar els nodes de la xarxa pels quals ha de passar una ràfega fins arribar al node de sortida, cal intercanviar informació d'encaminament entre els nodes, i també de resolució d'adreces entre l'identificador de node OBS i adreces físiques MAC. |

Encara que les tres primeres tasques difereixen en el seu ordre de prioritats, l'ideal és que totes tres poguessin tractar-se amb la mateixa prioritat, o millor i tot, que es processessin concurrentment. Una possible solució en el nostre sistema és fer ús dels dos processadors PowerPC que disposa la FPGA Virtex-II Pro, sobre la que es realitzarà la implementació. D'aquesta forma, un processador pot dedicar-se a processar solament les tasques relacionades amb el processament de les trames de control i reserva de recursos; i l'altre processador podria dedicar-se sols a llegir de la taula de recursos els temporitzadors que ha de configurar i arrencar, atendre les interrupcions generades, i tot seguit senyalitzar al Pla de Dades i al OXC la commutació, transmissió o recepció de les ràfegues.

Aquesta última aproximació, però, implica l'ús dels dos processadors PowerPC que inclou la FPGA, i per tant, s'haurà d'avaluar si altres tasques o funcionalitats del sistema també requereixen o no tenir a la seva disposició algun processador sobre el què executar-se.

Respecte l'últim punt marcat de prioritat "4", en una primera versió del programa es poden configurar aquestes taules de forma estàtica. En un futur, se'ns pot presentar l'oportunitat d'afegir sobre el pla de control l'intercanvi d'informació necessari per als protocols d'encaminament i resolució de l'adreçament.

S'ha de tenir en compte que l'ordre de les prioritats pot ser diferent en funció de quines polítiques de Qualitat de Servei es vulguin aplicar, o de si els protocols de reserva de recursos superiors modifiquen aquests valors. De fet en la tasca que s'ha marcat com de prioritat "2" es pot jugar amb els valors dels *offsets*, inserint valor grans, de manera que es doni suficient temps a la resta de nodes de la xarxa a que puguin processar-los. Llavors, si tenen més temps per a ser processats es pot fer que la tasca d'atendre i processar aquestes trames sigui de menor prioritat en el sistema.

Finalment, per tal de no deixar sense transmetre les ràfegues que s'han generat i que de fet, ocupen memòria de sistema, podria integrar-se en el mòdul que notifica al Pla de Control algun mètode per a que si la seva atenció no és atesa, pugui reintentar el procés de senyalització al Pla de Control.

### 3.5.2. Algoritme d'atenció del programa

L'algoritme del programa que s'utilitza en el *OBS Control Path Software* és de tipus *foreground/background*. Seguint la priorització de tasques comentat en els apartats anteriors, es poden classificar aquestes tasques com:

- Prioritàries: les que són generades per dispositius o crides externes, o les que es generen per a realitzar alguna tasca externa al programa principal que s'executa en el PowerPC.

- No prioritàries: les que no involucren tasques de comunicació amb dispositius externs, i que conformen el procés bàsic del node. Aquesta és principalment la tasca del programa de gestió de les taules d'encaminament i de resolució d'adreces.

La tasca més prioritària és la de l'atenció a la interrupció del temporitzador de ranures. És la funció més important, i la que cal atendre de la forma més ràpida possible. Si bé aquesta tasca és síncrona, ja que s'executa cada cert temps (el valor del temporitzador), l'excepció que es genera s'atén amb la seva corresponent interrupció. Aquesta, "interromp" a qualsevol de les tasques que s'estiguin executant en aquell moment. Les interrupcions generades pel dispositiu comptador s'enllacen al port d'Interrupcions Crítiques Externes del PowerPC.

La segona tasca més prioritària és la recepció de les trames de control, però no el seu processament. El que és important és atendre la recepció del paquet de control, per a que no es perdi. Per això en arribar un paquet, s'atén la interrupció, en la qual es duu a terme l'emmagatzemament del paquet en un

*buffer* temporal de creació pròpia. Aquest *buffer* té una capacitat limitada, i per tant, si es reben més paquets dels que es poden processar, aquest es desbordarà, i els paquets rebuts es descartaran. A més a més, l'element de recepció dels paquets en el disseny FPGA és l'IP Core PLB GEMAC [22], el qual, per sí mateix, també conté FIFOs d'entrada i sortida per a la recepció i transmissió de les dades (la grandària d'aquestes FIFOs és configurable),

Per a diferenciar la prioritat entre la funció de processar els paquets de control rebuts amb la de processar les peticions d'enviament de ràfegues, s'ha optat per una solució simple. La forma escollida es basa en executar "més cops" el procés de *background* de processament de trames de control, que l'altra funció de processament de les peticions d'enviament de ràfegues.

La Fig. 3.7 mostra esquemàticament l'algoritme central del programa a molt alt nivell, en el qual, el codi de processament de trames s'executa el doble de vegades que l'altre procés, i per tant, té prioritat doble.

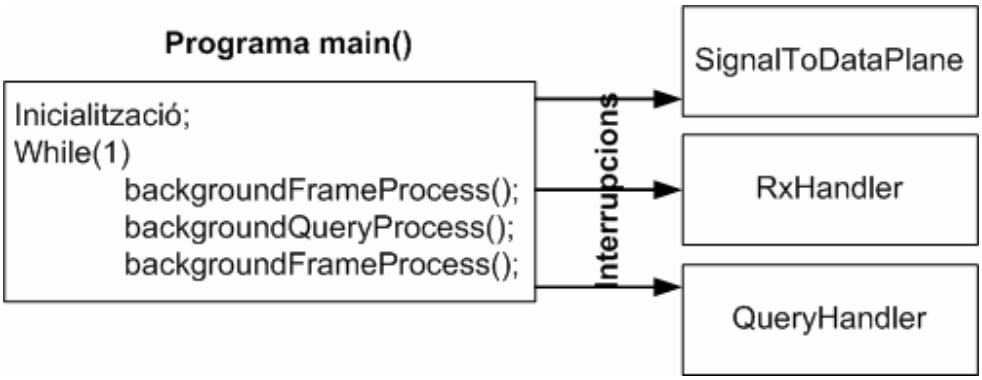



**Fig. 3.7.** Algoritme principal del programa.

# CAPÍTOL 4. IMPLEMENTACIÓ DEL PLA DE CONTROL

Un cop plantejat el disseny en el CAPÍTOL 3, en aquest capítol es descriurà la implementació concreta d'aquest disseny. Es farà un incís en les eines de software utilitzades en la implementació, així com també, s'explicarà la implementació del hardware del node OBS sobre la placa de desenvolupament FPGA de Xilinx.

## 4.1. Introducció

Per a posar en pràctica el disseny proposat del Pla de Control OBS s'han utilitzat unes plaques de desenvolupament FPGA d'Avnet [23]. Aquestes estan equipades amb una FPGA Virtex-II Pro 30 de Xilinx [24]. Xilinx és en l'actualitat el líder mundial en la fabricacions de FPGAs i CPLDs [25].

L'ús de la tecnologia FPGA ens permet dissenyar i implementar hardware tantes vegades com vulguem, modificar els dissenys, reimplementar mòduls, etc. Per tant, per a poder fer una "prova de concepte" d'una maqueta de xarxa OBS, l'ús d'aquesta tecnologia és molt útil, tant econòmicament com funcionalment. A més, si la FPGA porta integrats CPUs, com és el nostre cas, sobre els dissenys hardware es poden també implementar programes i aplicacions (aquesta ha estat la part central del projecte).

A continuació es descriuen breument les capacitats de la placa de desenvolupament, justificant el seu ús. Posteriorment s'explica quina ha estat la implementació hardware utilitzada per a crear el node OBS.

## 4.2. Disseny/Implementació hardware/software per a FPGA

Per a la implementació física del node OBS (el seu Pla de Control) s'ha utilitzat una placa de desenvolupament FPGA d'Avnet. Sobre aquesta s'ha implementat el hardware especificat en l'apartat 4.3.

### 4.2.1. La placa de desenvolupament FPGA de Xilinx

Com s'ha introduït anteriorment, en el projecte s'ha utilitzat una placa de desenvolupament FPGA d'Avnet. Aquesta conté una FPGA de Xilinx, model Virtex-II Pro-30. La placa de desenvolupament (veure Fig. 4.1), conté els següents components (per a més informació consultar l'ANNEX D):

- FPGA Xilinx Virtex-II Pro XC2VP30-FF896.
- 2 Mòduls SFP per a Gigabit Ethernet.
- 2 connectors HSSDC2 (Infiniband)
- 1 interfície 10/100/1000 Mbps Ethernet.
- 32 MBytes de memòria SDRAM.
- 128 MBytes de memòria DDR SDRAM.
- 2 Mbytes de memòria SRAM.

- Mòdul per a memòria Flash.
- 4 connectors d'expansió d'Entrada/Sortida de propòsit general (de 140 pins).
- Ports sèrie RS-232.
- Connector PCI universal (compatibles amb ranures de 32 i 64 bits).

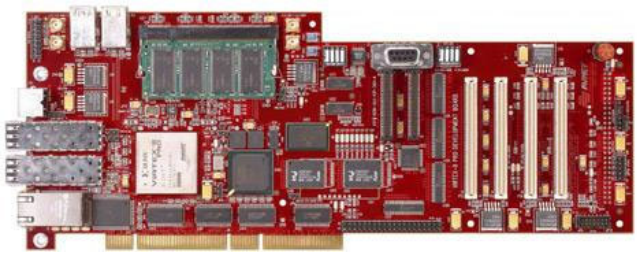


**Fig. 4.1.** Placa de desenvolupament FPGA Virtex-II Pro d'Avnet.

Aquesta placa, proporciona les capacitats necessàries per a connectar al Pla de Control, el Pla de Dades (per les seves interfícies òptiques), així com a un commutador òptic mitjançant els connectors d'expansió.

Per altra banda, les principals característiques de la FPGA (per a més informació consultar l'ANNEX C) són (veure Taula 4.1):

**Taula 4.1.** Característiques de la FPGA Virtex-II Pro 30 de Xilinx.

| **Característica** | | **Descripció** |
|---|---|---|
| **Model** | | XCE2VP30 |
| **Cel·les lògiques** | | 30.816 |
| **Empaquetat** | | FF896 |
| **Velocitat** | | -6 |
| **CLB (1 = 4 *slices* = 128 bits max)** | ***Slices*** | 13.696 |
| | **Max. RAM Dist (Kb)** | 428 |
| **Blocs Multiplicadors 18x18** | | 136 |
| **Blocs SelectRAM+** | **Max. RAM Bloc (Kb)** | 2.448 |
| | **Blocs 18 Kb** | 136 |
| ***Pads* E/S d'usuari (Max)** | | 556 |
| **DCMs** | | 8 |
| **Processadors PowerPC™** | | 2 |
| **Transceptors RocketIO 3,125 Gbps** | | 8 |

### 4.2.2. Software de desenvolupament

Per a la programació i implementació, tant del hardware com del software, en FPGA, Xilinx ofereix un conjunt d'eines de desenvolupament (IDE) [26], amb un entorn gràfic amigable i molt usable. Les principals eines que componen el paquet són:

- **Xilinx Platform Studio (XPS). Embedded Design Kit:** Aquest software s'utilitza per a dissenyar i estructurar el sistema hardware incrustat, que es compon principalment d'un processador, memòria i els seus perifèrics. L'eina permet realitzar la síntesi del disseny FPGA i el *Place & Route*. Al mateix temps, per a als diferents dispositius i perifèrics que componen el disseny associa un *driver* per a poder manegar-los des de les aplicacions que es desenvolupen. Per això s'encarrega de compilar i enllaçar les llibreries. L'entorn també pot utilitzar-se per a realitzar la programació de les aplicacions que poden executar-se sobre el processador (o processadors).

- **Xilinx Platform Studio SDK:** Aquesta eina és de fet l'Eclipse [27], adaptat per al XPS. S'utilitza per a generar, compilar i depurar el codi dels programes i aplicacions en llenguatge C. En cas d'utilitzar el depurador, es fa servir el *Xilinx Microprocessor Debugger* (XMD), un programa "dimoni" que comunica el port JTAG amb algun programa de depuració (gràfic). En el projecte s'ha utilitzat aquesta eina perquè ofereix molts avantatges a l'hora de tractar grans quantitats de codi.

- **Xilinx ISE:** Aquesta eina de desenvolupament integra totes les funcions necessàries per al disseny complet de sistemes lògics. Ofereix *wizards* (guies) per a la creació de sistemes lògics hardware, eines per a l'anàlisi de potència i de disseny orientat a requeriments de temps, simulació d'HDL.

- **ModelSim SE 6.1:** Aquest software s'utilitza per a realitzar simulacions de comportament i de temporització (*behavioural and timing simulations*). A partir dels dissenys HDL permet realitzar una simulació de com es comporta funcionalment el codi. Un cop el disseny es sintetitza i es realitza el *Place & Route*, també permet realitzar simulacions de temps, per a comprovar que les restriccions temporals es compleixen.

- **iMPACT:** Aquest programa permet descarregar els fitxers *bitstream* al xip FPGA. Un cop descarregats, realitzen la configuració de les cel·les i portes lògiques en el silici, i d'aquesta forma, es configura el hardware programat (en VHDL, o Verilog).

## 4.3. Disseny i Implementació hardware

Abans de començar a programar el nostre OCPS s'ha de configurar el sistema hardware que suportarà el nostre software. Per a que el programa pugui executar-se caldrà que hi hagi en el sistema alguna CPU, memòria, busos i alguns perifèrics que aportin les funcionalitats que requereixen el disseny descrit en el CAPÍTOL 3. La Fig. 4.2 mostra un diagrama a alt nivell de la interconnexió dels diferents dispositius. En l'apartat G.1 de l'ANNEX G s'ofereix un diagrama més detallat, proporcionat pel programa XPS. Per altra banda, la Taula 4.2 presenta tots els dispositius, busos i perifèrics que s'han configurat en el nostre sistema hardware. Per a cadascun es descriu el tipus de dispositiu

(*Intellectual Property*, veure [28]), així com la descripció de les tasques que realitza el component en el disseny.

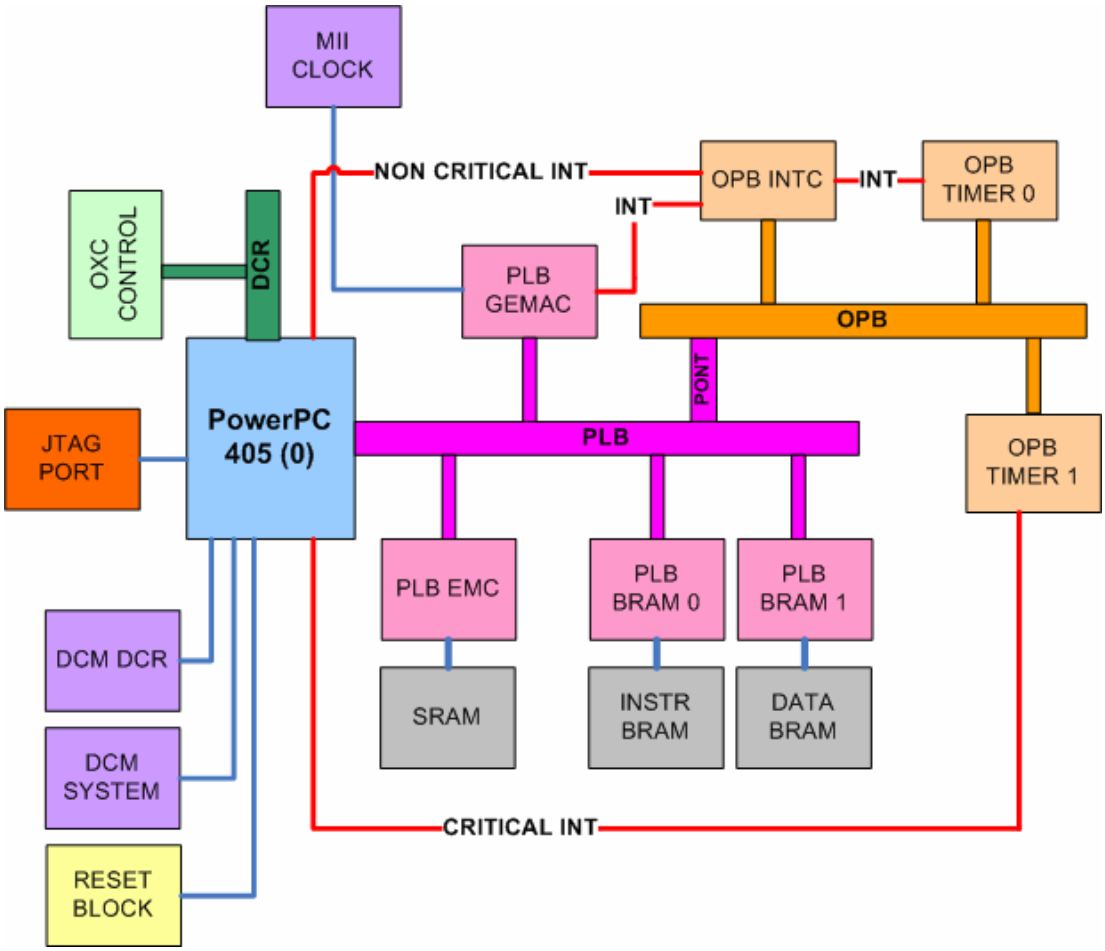



**Fig. 4.2.** Diagrama de la configuració hardware.

Com es pot apreciar en la figura anterior, el sistema disposa de tres busos principals: el *On-chip Peripheral Bus* (OPB), el *Processor Local Bus* (PLB) i el *Device Control Register* (DCR). El rendiment del bus PLB és major que el OPB [29], i pengen d'aquest el dispositiu que ofereix la capa MAC de Gigabit Ethernet al sistema, i les memòries. Sobre el OPB tenim els temporitzadors i el controlador d'interrupcions. Per a controlar el commutador òptic (OXC), es realitza mitjançant la interfície DCR amb un mòdul de creació pròpia en el projecte.

També es pot veure com diferents dispositius estan connectats al processador per via dels seus ports externs d'interrupcions. Un temporitzador, el que aporta el temps de ranura, està connectat al port d'excepcions crítiques del PowerPC 405. El controlador d'interrupcions, que controla les interrupcions del temporitzador que marca la generació de les peticions d'enviament de ràfega, i la transmissió i recepció dels paquets de control, està connectat al port d'excepcions externes no crítiques.

**Taula 4.2.** Llistat de components del disseny hardware del sistema.

| Nom Component | Tipus de Dispositiu | Descripció |
|---|---|---|
| ppc405_0 | Processador PowerPC 405 | Processador PPC 405 (instància 0). Sobre aquest processador s'executa el programa de control. |
| ppc405_1 | Processador PowerPC 405 | Processador PPC 405 (instància 1) que incorpora la Virtex-II Pro 30. Aquest processador no executa cap software, però per tal de poder utilitzar el controlar JTAG en el disseny es necessita que els dos processadors presents en la FPGA siguin instanciats. |
| ppc_plb | Bus PLB | Bus PLB d'IBM (*Processor Local Bus*). Sobre aquest bus pengen la majoria de perifèrics i les memòries (de dades i instruccions). |
| ppc_opb | Bus OPB | Bus OPB (*On-Chip Peripheral Bus*). Sobre aquest bus pengen també alguns perifèrics i ports d'E/S. |
| ppc_dcr | Bus DCR | Bus DCR (*Device Control Register*). Aquesta interfície proporciona el mecanisme per a que el bloc processador pugui inicialitzar i controlar dispositius perifèrics que resideixen en el mateix xip FPGA. Amb aquesta interfície es pot controlar el commutador òptic (OXC). |
| ppc_plb2opb | Pont entre busos | Dispositiu que estableix un pont entre els busos PLB i OPB. |
| jtagppc_0 | Controlador JTAG | Controlador JTAG per a depurar el programa que s'executa en el processador. |
| ppc_plbbram_cntlr | Controlador Memòria BRAM en PLB | És un controlador de memòria BRAM que penja del bus PLB. Aquesta memòria s'utilitza per a emmagatzemar el programa, la part d'instruccions i codi. |
| data_plbbram_cntlr | Controlador Memòria BRAM en PLB | És un controlador de memòria BRAM que penja del bus PLB. Aquesta memòria s'utilitza per a emmagatzemar la part de dades del programa. |
| ppc_bram | Memòria BRAM | Memòria de tipus bloc BRAM. El FPGA Virtex-II Pro inclou memòria de tipus bloc. Aquest bloc s'utilitza per a instruccions i codi, i està controlat pel ppc_plbbram_cntlr |
| data_bram | Memòria BRAM | Memòria de tipus bloc BRAM. El FPGA Virtex-II Pro inclou memòria de tipus bloc. Aquest bloc s'utilitza per a dades, i està controlat pel data_plbbram_cntlr |
| sram | Memòria SRAM | És un controlador de memòria externa al xip FPGA. En aquesta cas, la placa de desenvolupament disposa de 2 MBytes de memòria SRAM, i mitjançant aquest dispositiu es pot accedir a ella. Durant les proves, en algunes, s'utilitza tant per emmagatzemar el programa com les dades. |

| | | |
|---|---|---|
| sram_util_bus_split_0 | Utilitat | Mòdul que s'utilitza per a dividir l'ample de banda d'un bus. S'utilitza per a accedir a la memòria SRAM. |
| dcm_system | Gestor de Rellotge Digital | Els DCMs s'utiltzen per a generar diversos tipus de rellotge a partir d'una senyal de rellotge d'entrada. Aquest s'utilitza per a generar el rellotge del processador (a 300 MHz), i per als busos PLB i OPB (100 MHz). |
| dcm_dcr | Gestor de Rellotge Digital | Aquest DCM s'utilitza per a generar el rellotge del bus DCR (150 MHz). |
| reset_block | Bloc de reinicialització | El mòdul de reinicialització s'utilitza per a distribuir la senyal de *reset* als diferents dispositius que componen el sistema, però principalment al processador. |
| ppc_uart | Port sèrie UART | Aquest dispositiu proporciona una interfície sèrie per a connectar una consola a la placa de desenvolupament. Aquesta interfície proporciona els ports STDIN/STDOUT necessaris per als programes. |
| ppc_gemac | MAC Gigabit Ethernet | El mòdul proporciona una instància MAC de Gigabit Ethernet al sistema. Sobre la interfície s'envien i reben els paquets de control. Físicament, aquesta interfície és un port al Ethernet PHY de la placa de desenvolupament. El dispositiu es connecta al processador per mitjà del bus PLB. |
| ppc_intc | Controlador d'Interrupcions | Aquest dispositiu és un controlador d'interrupcions. Permet gestionar diverses excepcions, atorgant a cadascuna una prioritat relativa, i gestionar les crides als *handlers* de les interrupcions. Es connecta al bus OPB. Aquest controlador s'encarrega de les interrupcions generades pel dispositiu GEMAC i del comptador que simula la generació de peticions de transmissió de ràfega. Al seu temps, el controlador està connectat al port d'excepcions externes no crítiques del processador. |
| ppc_mii_clock_ management | Gestor de rellotge MII | Aquest gestor genera els senyals de rellotge per al *Gigabit Media Independent Interface* (GMII). A partir d'un senyal de rellotge diferencial de 125 MHz que es genera en la placa d'Avnet. |
| opb_timer_0 | Temporitzador | Aquest IP és un temporitzador que es connecta al bus OPB. Es pot configurar per a que realitzi un compte enrera (o endavant) des d'un valor enter (32 bits). En el nostre disseny aquest temporitzador marca la taxa de generació de peticions d'enviament de ràfega. Cada cop que el temporitzador arriba a '0', llança una excepció. Està connectat al controlador d'interrupcions. |
| opb_timer_1 | Temporitzador | Aquest temporitzador proporciona el temps global al SRV (Vector de Reserves Ranurades). És el que determina, cada certs temps, el salt d'una ranura a una altra. Com que en el sistema la seva excepció és més crítica, està connectat al port d'excepcions crítiques externes del processador. |

El senyal de rellotge del sistema prové de dues fonts externes a la FPGA (que estan en la placa de desenvolupament). Una font de rellotge diferencial a 125 MHz que proporciona el rellotge a la interfície GMII del GEMAC. El senyal de rellotge diferencial s'adequa mitjançant el "Gestor de rellorge MII", i s'obtenen els senyals de referència RX i TX de MII. Anàlogament, la placa també ofereix un rellotge de 100 MHz que s'utilitza per marcar el senyal de rellotge dels diferents busos i al processador. El *dcm_system* s'utilitza per a donar el senyal als busos OPB i PLB a 100 MHz, i al DCR a 150 MHz, amb el *dcm_dcr*. Per altra banda, mitjançant multiplicadors i divisors del senyal de rellotge, s'arriba als 300 MHz, freqüència a la que funciona el processador PowerPC 405 en el nostre sistema. La següent Taula 4.3 resumeix la configuració dels rellotges.

**Taula 4.3.** Origen i configuració dels rellotges.

| Font (MHz) | Gestor de Rellotge | Dispositiu | Freq. (MHz) |
|---|---|---|---|
| 100 | dcm_system | PPC405 | 300 |
| | | Bus PLB | 100 |
| | | Bus OPB | 100 |
| | dcm_dcr | DCR | 150 |
| 125 diferencial | mii_management | GEMAC | 125 |

Pel que fa a l'adreçament dels dispositius i les memòries, la següent Taula 4.4 mostra els marges establerts. El PowerPC 405 té un marge d'adreçament de 32 bits, el qual permet adreçar fins a 4 GBytes (4.294.967.296 bytes).

**Taula 4.4.** Adreçament del sistema.

| Instància | Adreça Base | Adreça Superior | Grandària |
|---|---|---|---|
| ppc_gemac | 0x00000000 | 0x0000FFFF | 64 KBytes |
| ppc_uart | 0xF0000000 | 0xF00000FF | 256 bytes |
| opb_timer_0 | 0xF0000100 | 0xF00001FF | 256 bytes |
| opb_timer_1 | 0xF0000200 | 0xF00002FF | 256 bytes |
| ppc_intc | 0xF0010000 | 0xF001FFFF | 64 KBytes |
| data_plbbram_cntlr | 0xF8000000 | 0xF801FFFF | 128 KBytes |
| ppc_plbbram_cntlr | 0xFFFF0000 | 0xFFFFFFFF | 64 KBytes |
| sram | 0x10000000 | 0x100FFFFF | 1 MByte |
| ppc_plb2opb | 0xF0000000 | 0xF7FFFFFF | 128 MByte |

En els tests s'ha comprovat el rendiment en dues configuracions de memòria; una sobre els BRAMs (tant les dades com les instruccions), i una altra sobre la SRAM. Així i tot, en aquest últim cas, el sector d'arrencament del programa resideix en la posició de memòria 0xFFFFFFFC, i per tant, està sobre el dispositiu controlador de memòria BRAM *ppc_plbbram_cntlr*.

## 4.4. Implementació del Software

A continuació es descriu l'estructura de la nostra implementació concreta del programa de control OCPS (*OBS Control Plane Software*). El llenguatge utilitzat en la seva programació ha estat el C, i en la seva implementació s'han seguit algunes de les pautes descrites en [30][31][32] per a millorar el rendiment d'execució del programa. A l'hora de crear el programa s'ha d'intentar que aquest s'executi de la forma més ràpida possible, ja que previsiblement, si el temps de servei de les peticions i processament dels paquets de control és menor, la capacitat del nombre de processaments augmentarà, i la probabilitat de bloqueig disminuirà.

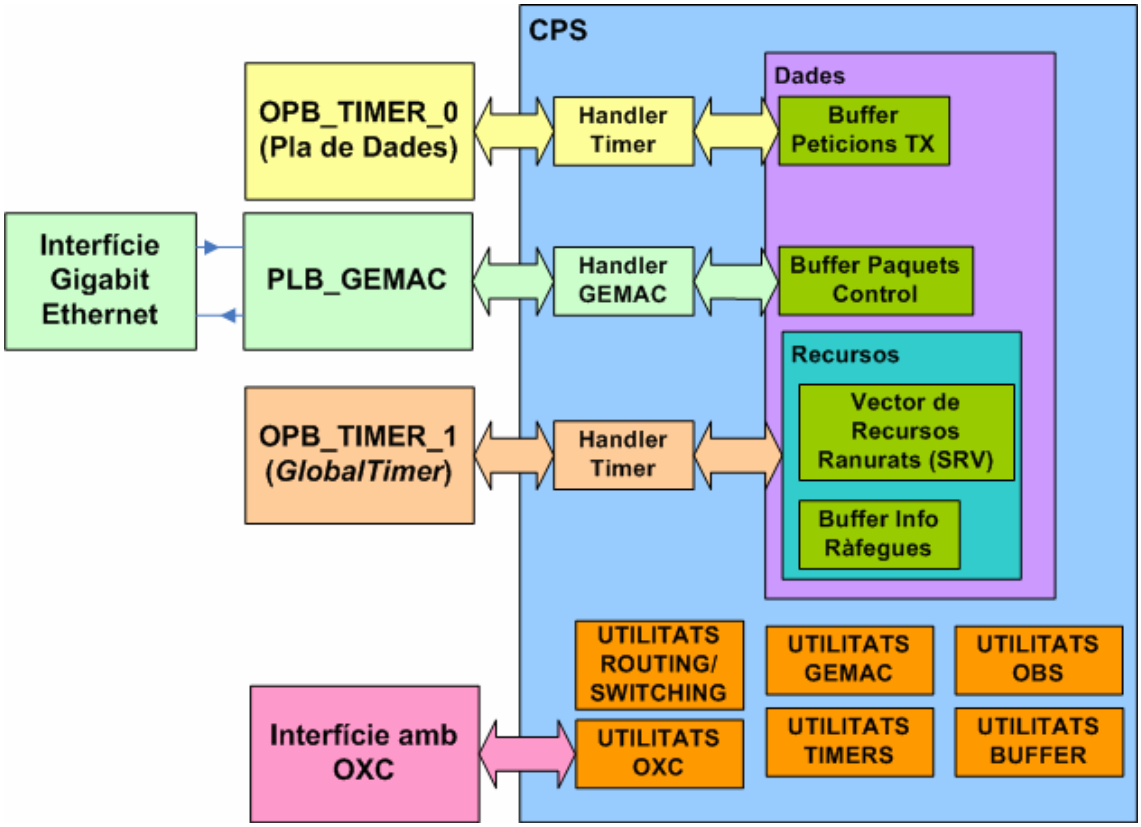



**Fig. 4.3.** Implementació a alt nivell del programa.

El codi està estructurat en diferents fitxers. El programa principal realitza la inicialització dels diferents dispositius hardware del sistema, així com de les estructures i memòries *buffer* que s'utilitzen. Al final de la inicialització arrenca els temporitzadors, inicialitza els comptadors de trames, peticions, reserves, etc, i introdueix al bucle del programa principal. Aquest *loop* executa per ordre de prioritats els processos de gestió de les peticions de transmissió de ràfega i els de gestió dels paquets de control. A intervals marcats pel *GlobalTimer*, s'executa la interrupció del temporitzador de ranuració, el qual processa la senyalització marcada en el Vector de Recursos Ranurats (SRV).

Altres fitxers contenen les funcions que realitzen les tasques descrites a continuació:

- **Utilitats *Buffer*:** En aquest fitxer resideixen les estructures i funcions necessàries per utilitzar els *buffers* de recepció de paquets de control i de peticions de TX de ràfega.

- **Utilitats OBS:** Aquest fitxer conté les estructures de dades i funcions necessàries per a processar i gestionar els missatges de control i les peticions de TX de ràfega. Interactua amb el SRV, en el qual, segons la informació processada, realitza la reserva de recursos.

- **Utilitats Timers:** Aquest fitxer d'utilitats conté l'estructura de dades SRV i el *buffer* per a emmagatzemar la informació de les ràfegues que han pogut reservar recursos. També té les funcions per a inicialitzar els temporitzadors, configurar-los, i tractar les excepcions. En una d'aquestes executa la funció de senyalització en funció dels valors del SRV.

- **Utilitats GEMAC:** Les funcions d'aquest fitxer permeten manegar les interrupcions produïdes al rebre i enviar paquets de control. Conté les funcions per a realitzar l'enviament dels paquets de control, la traducció del format dels paquets de control a bytes, i també les necessàries per a inicialitzar el dispositiu físic. L'enviament i recepció dels paquets utilitza les funcions proporcionades per la llibreria del PLB GEMAC.

- **Utilitats *Routing/Switching*:** Conté les funcions i estructures de dades necessaris per a realitzar l'addició, consulta i eliminació de camps en les taules d'encaminament i commutació. També conté les funcions per a inicialitzar aquestes taules, així com una funció per a realitzar la introducció de valors per defecte i de forma estàtica (els utilitzats en les proves).

- **Utilitats OXC:** Conté bàsicament dues funcions: una per a escriure un valor de configuració en el commutador òptic per mitjà de la interfície DCR; i una altra per a llegir l'estat actual de configuració del commutador.

# CAPÍTOL 5. PROVES I RESULTATS

Un cop plantejat el disseny i realitzada la implementació del programa de control, s'ha d'avaluar el correcte funcionament del programa i extreure alguns resultats per a determinar el seu rendiment. En aquest capítol es presenten les proves realitzades, els seus resultats, i al final, un estudi més analític dels resultats, per a que ens ajudi a parametritzar millor el nostre sistema. Les proves s'han realitzat sobre la maqueta que s'explica en l'apartat 5.1.

## 5.1. Maqueta de proves

La maqueta de proves per a comprovar el correcte funcionament del sistema és molt senzilla i més reduïda que la maqueta de xarxa proposada en l'apartat 3.2 (maqueta que s'utilitzarà al combinar el programa de control, junt amb OXC i el Pla de Dades). En la figura Fig. 5.1 es mostra un diagrama d'aquesta maqueta.

En el PC client s'executa un programa client d'OBS generador de paquets de control. El codi d'aquest programa es pot consultar en l'apartat E.4 de l'ANNEX E. Aquest programa genera paquets de control, els quals sol·liciten fer un *Setup* en els recursos òptics (commutador + receptor òptic) que controla el node Frontera/Central d'OBS, implementat en la placa de desenvolupament.

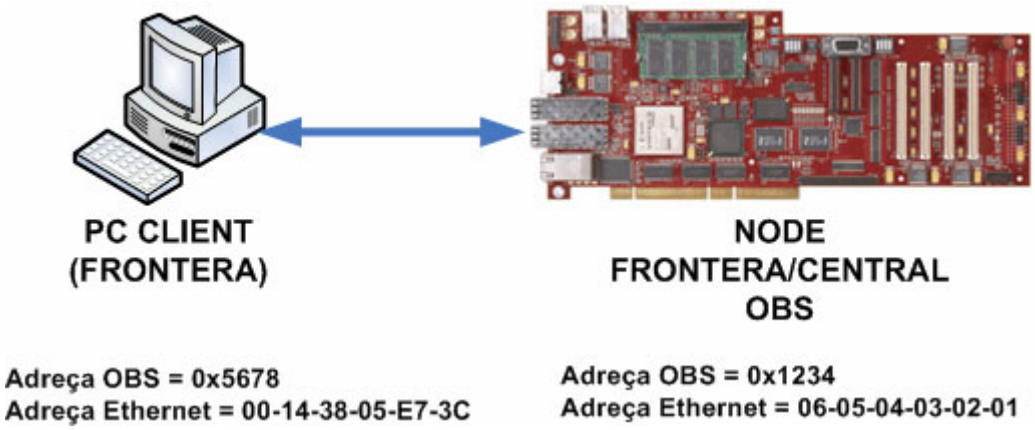



**Fig. 5.1.** Maqueta OBS de proves.

El càlcul de l'*offset* i de la llargària de la ràfega en el programa client s'ha realitzat mitjançant les llibreries *random* de Linux. En concret, el valor d'*offset* sorgeix d'una variable aleatòria lineal entre 1 i 99, i el de la longitud, entre 1 i 10. Ambdós valors representen el nombre de ranures.

Com que la taxa de generació de paquets de control és baixa (a causa del processament de generar els números aleatoris, crear el paquet, conformar-lo amb la llibreria Libnet, i enviar-lo), ha calgut utilitzar programes extres per a generar paquets a una taxa determinada. Mitjançant el nostre programa client

s'han generat paquets de control, els quals s'han capturat amb el programa Ethereal [33]. Posteriorment, amb el programa TCPReplay [34] i amb el fitxer de captura s'han generat els mateixos paquets generats anteriorment, però a una taxa fixa.

Per altra banda, en la placa de desenvolupament s'executa el programa de control implementat, i que funciona segons el descrit en els capítols CAPÍTOL 3 i CAPÍTOL 4.

## 5.2. Proves de rendiment

Les proves s'han centrat en la obtenció, principalment, de dos tipus de resultats. Per un costat, resultats sobre el temps de servei de les principals funcions que componen el programa en funció de la càrrega del sistema. Els altres resultats es centren en determinar el *throughput* de reserva de recursos en funció de les reserves realitzades, també, sota diferents condicions de càrrega. Aquests últims resultats no són determinants en la mesura que no s'ha realitzat un estudi exhaustiu dels millors mètodes per a generar els *offsets* ni per a aplicar resolució de contencions.

### 5.2.1. Configuració de les proves

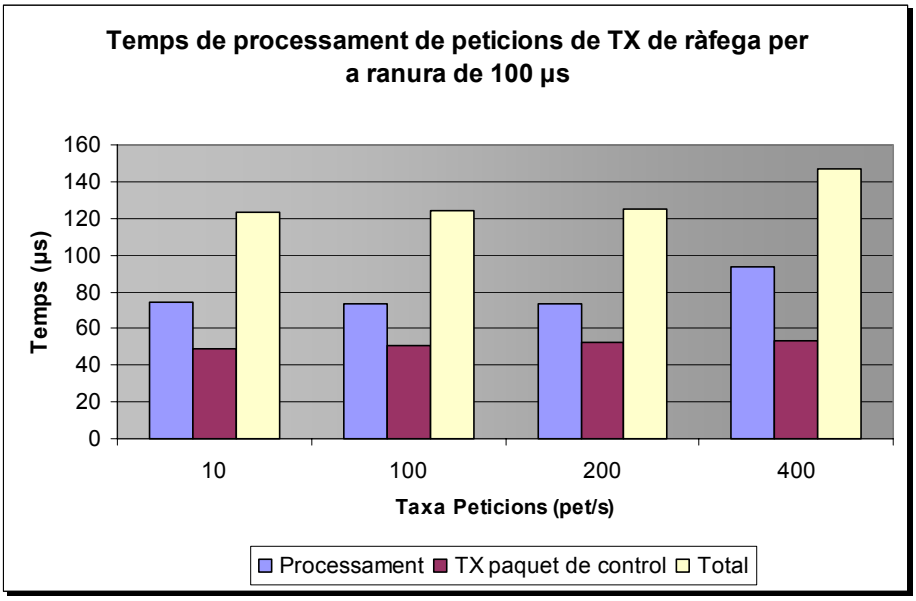
La configuració del sistema en les proves de rendiment és:

- Memòria: Dades i Programa s'executen en memòria BRAM (*Block* RAM).
- Longitud del SRV = 1024 ranures.
- Longitud *buffer* informació de ràfegues reservades = 512 camps.
- Longitud *buffer* recepció de paquets de control = 256 paquets (60 Bytes/paquet).
- Longitud *buffer* recepció de peticions de TX local = 32 peticions.

### 5.2.2. Proves processament de peticions de TX de ràfega

En aquests tests s'ha mesurat el temps de servei en realitzar el processament de les peticions de transmissió de ràfega i el rendiment de reserva obtingut en relació al màxim possible (100% de ranures reservats útils). La petició que es realitzava era alternadament ràfegues de 250.000 bytes o 350.000 bytes. Per tant, al romandre aquest valor constant al llarg de les proves, a temps de ranura menors, el nombre de ranures de longitud de la ràfega és major, i per tant, es realitza la petició de reserves de major longitud. Així mateix, el temps de procés sols s'ha calculat per a processaments on la reserva era efectiva. Per tant, per a peticions que no han obtingut reserva, el seu temps no es quantifica.

Només es descriuran les gràfiques més representatives. La resta estan disponibles en l'ANNEX H, així com també les taules de valors.

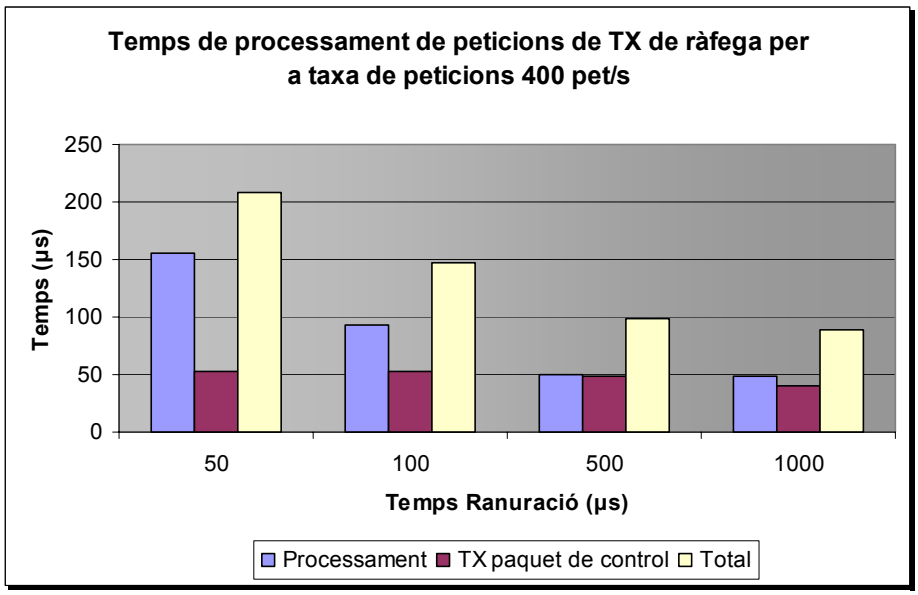



**Gràf. 5.1.** Temps de processament de peticions de TX de ràfega amb ranuració de 100 µs.

En Gràf. 5.1 es pot apreciar l'evolució del temps de processament de les funcions de "processament de peticions" i "TX del paquet de control" al augmentar la taxa de peticions per a una ranuració de 100 µs. Com es pot veure, els diferents temps es mantenen quasi bé constants per a taxes baixes, però al augmentar a 400 pet/s, el temps de processament augmenta considerablement (+ de 20 µs). Aquest comportament és normal, ja que al tractar més peticions, es provoquen més interrupcions en el flux normal del programa, i per tant, el processament es veu interromput més cops, provocant un augment del temps de procés.

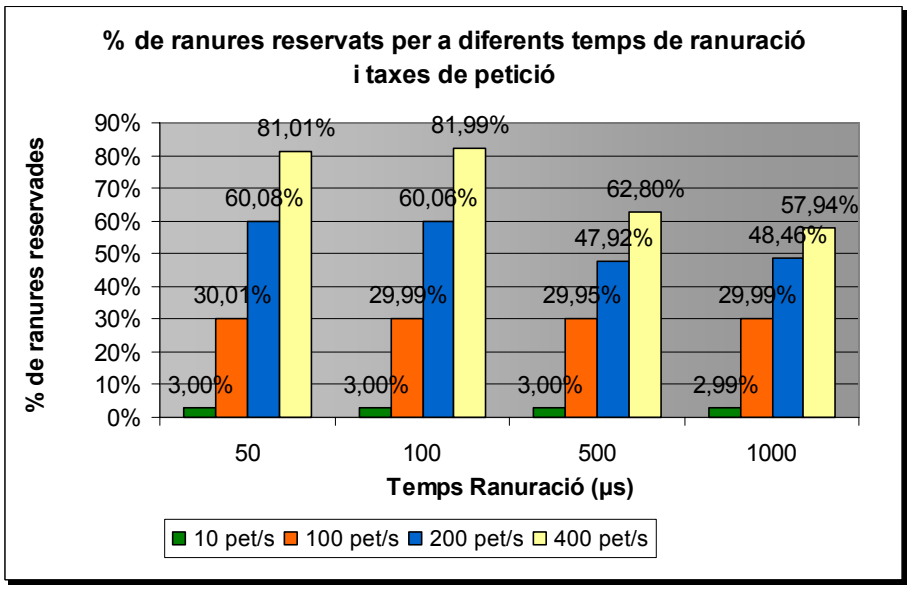
Per altra banda, el Gràf. 5.2 representa l'evolució per a una taxa de peticions fixa al disminuir el temps de ranuració. En la gràfica es passa d'un temps de ranura de 50 µs a un temps d'1 ms. A menor temps de ranura, es produeixen més interrupcions en el sistema associades al *GlobalTimer*. Això explica que per tant, el temps de procés augmenta al disminuir el temps de ranura. Així i tot, es pot apreciar com el temps de transmissió del paquet de control (un cop s'ha realitzat la reserva) es manté quasi constant, sigui quin sigui el temps de ranura. Això és així perquè la transmissió del paquet de control no la controla el processador, i passa a ser controlat pel dispositiu GEMAC, el qual, un cop transmès el paquet, avisa amb una interrupció al processador.

El Gràf. 5.3 mostra quina és la taxa de rendiment de reserva de ranures al variar el temps de ranuració i la taxa de peticions. S'ha de notar que el % de reserva no serà mai del 100%, ja que com hem comentat, el nombre de ranures reservades és sempre igual al nombre de ranures útils + 2. Aquestes dues s'utilitzen com a marge de guarda. En les proves es comptabilitzava el nombre de ranures útils.




**Gràf. 5.2.** Temps de processament de peticions de TX de ràfega per a taxa de 400 pet/s.

Es pot veure com al augmentar la taxa de peticions, també augmenta el % de reserva de ranures. Aquest increment de la reserva augmenta també paral·lelament al nombre de peticions descartades (per a les quals no s'han pogut reservar ranures). Es pot apreciar aquest valor en les taules de l'apartat H.1.1.

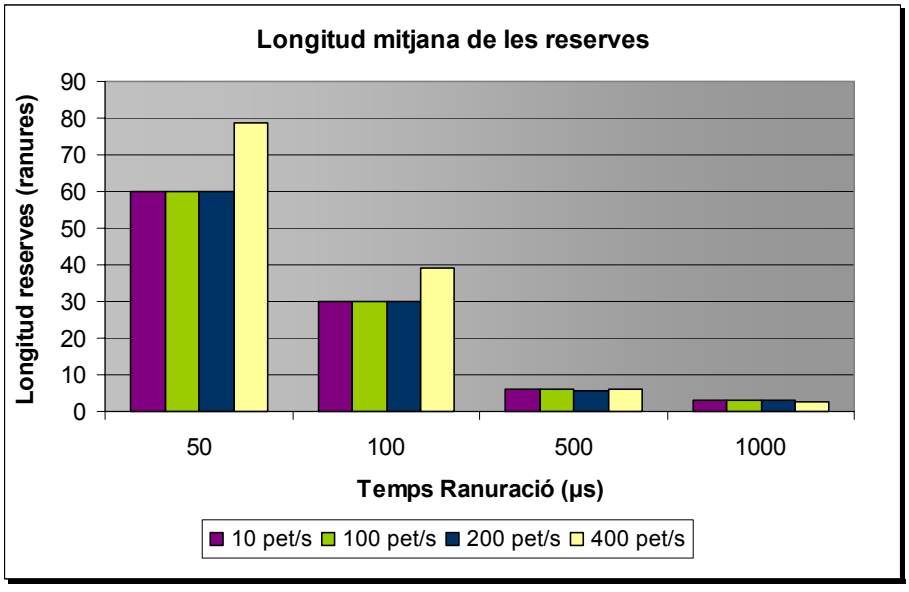



**Gràf. 5.3.** % ranures reservades per a diferents temps de ranuració i taxes de petició.

Per altra banda, al variar el temps de ranuració, es pot veure com a taxes de petició baixes (10 pet/s o 100 pet/s), el % de reserva es manté quasi bé constant. Això és així per dues causes: la primera és que totes les peticions aconsegueixen reservar recursos; la segona raó és que al disminuir la ranuració, augmenta el nombre de ranures a reservar. Si no fos així, que augmentessin el nombre de ranures, la taxa de reserva disminuiria, perquè el nombre de ranures totals reservades respecte al nombre de ranures totals passades amb el temporitzadors seria molt menor.

Per a taxes de petició més elevades, al disminuir el temps de ranuració, el rendiment de reserva augmenta. Aquest comportament es deu a que arriben moltes més peticions, que ofereixen més valors diferents d'*offsets* i longituds de ràfega, i per tant, hi ha més probabilitats que alguna d'aquestes peticions pugui ser reservada.

Finalment en el Gràf. 5.4 es mostra la longitud mitjana de les reserves. Aquesta longitud varia en funció del temps de ranuració (tal com hem explicat abans). Per al cas de ranures d'1 ms les dues longituds possibles eren de 2 i 4 ranures. En les proves, la mitjana de longitud reservada ha estat de 3. Per al cas de ranuració de 500 µs la mitjana de longitud s'ha situat sobre 6. Per als dos casos de ranuració més petita (100 µs i 50 µs) per a taxes baixes, les reserves es situaven sobre la mitjana. Però en canvi, per a una taxa de petició de 400 pet/s, la majoria de reserves es produïen per a peticions de reserva de ràfegues amb major longitud (80 ranures i 40 ranures, respectivament).



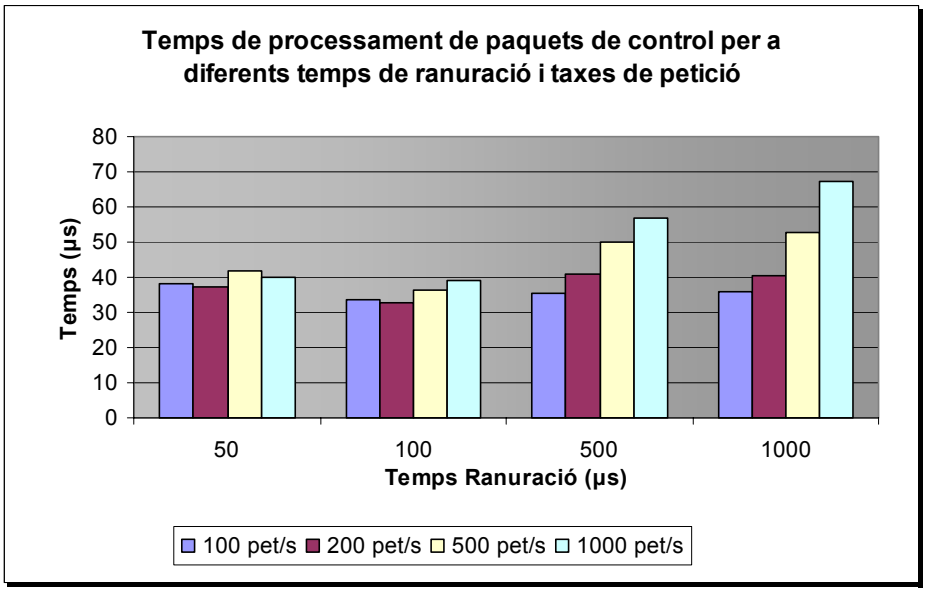

**Gràf. 5.4.** Longitud mitja de les reserves per a diferents temps de ranuració i taxes de petició.

### 5.2.3. Proves processament de paquets de control de Setup

Aquests tests tenen com objectiu mesurar el temps de processament i el rendiment de les reserves quan el node OBS implementat rep paquets de control d'altres nodes (en el nostre cas, del PC client) i n'ha de processar el contingut i realitzar la reserva, si s'escau.

Per a aquest cas, el PC client genera paquets de control amb els camps d'*offset* i longitud de ràfega especificats en l'apartat 5.1.

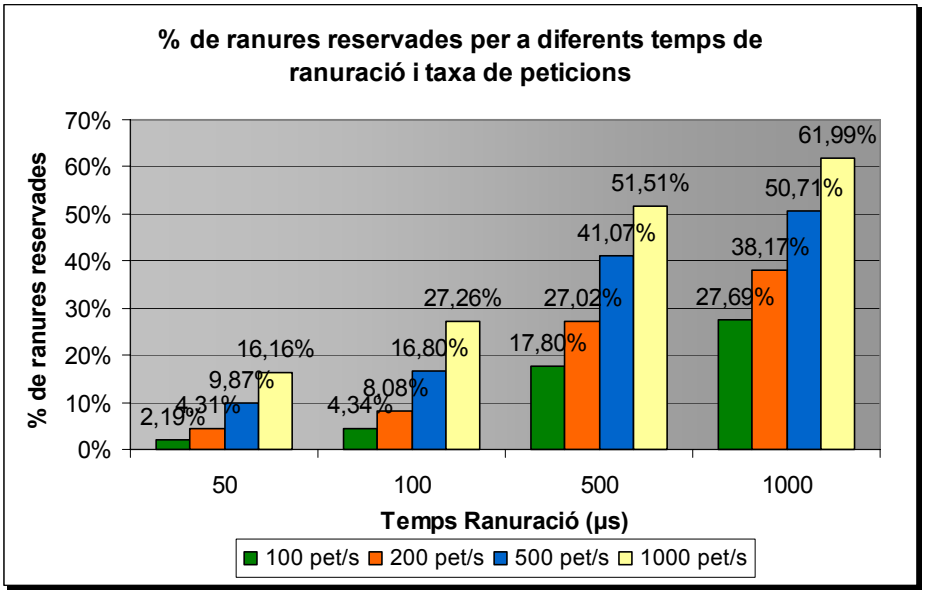
En el Gràf. 5.5 es representen els resultats obtinguts del temps de processament dels paquets de control per a diferents temps de ranuració i taxes de petició. En aquest cas, el temps calculat només és per a reserves efectives. Com es pot observar, per a taxes de paq/s baixes, el temps es manté per a diferents temps de ranura, però quan les taxes augmenten (més de 500 paq/s), el temps disminueix al disminuir el temps de ranura. Aquest comportament és contrari al que hem obtingut en l'apartat 5.2.2. Una possible explicació és que al disminuir el temps de ranura, aquesta segueix sent de l'ordre del temps que triga en realitzar-se el procés. Contràriament, en el cas de l'apartat 5.2.2, el temps de procés era molt major al temps de ranura, i per tant, per a un procés es produïen més interrupcions, el que provocava que augmentés.



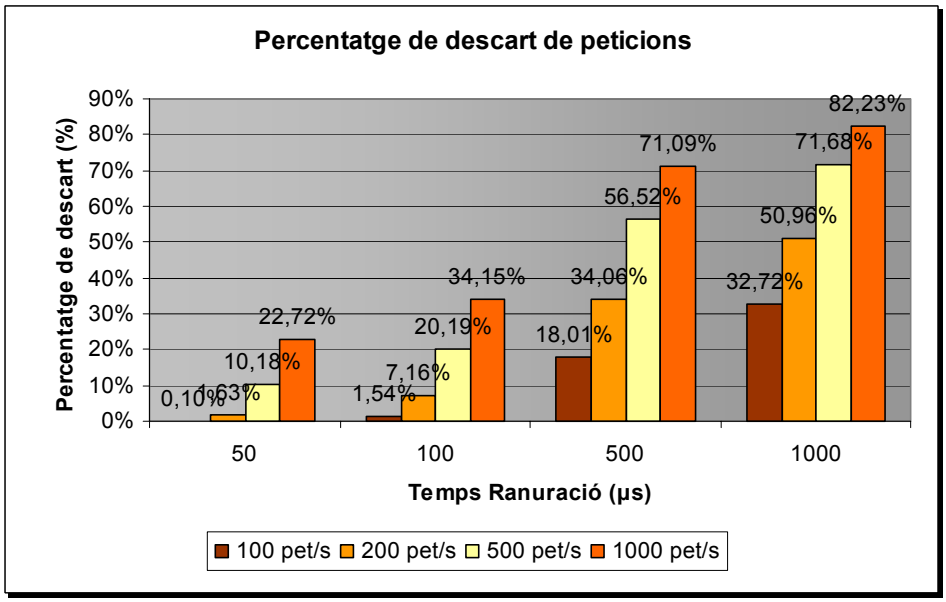

**Gràf. 5.5.** Temps de processament de paquets de control.

Pel que fa al càlcul del rendiment de ranures reservades (veure Gràf. 5.6), la tendència mostra com per a un mateix valor de temps de ranura, la taxa de reserva s'incrementa. Aquest comportament és a causa de que al rebre major nombre de paq/s, hi ha més possibilitats i variabilitat en les peticions, i per tant, és més fàcil que hi hagin possibilitats que les reserves siguin factibles.

Per altra banda, al mantenir el valor de la taxa i disminuir el valor del temps de ranura, la tendència normal és la de disminuir el valor del percentatge de ranures reservades. Aquest comportament està molt relacionat amb els valors de percentatge de peticions descartades (veure Gràf. 5.7). Com es pot veure, al disminuir la taxa de descart de peticions, el nombre de ranures reservades també disminueix.




**Gràf. 5.6.** Rendiment de ranures reservades.




**Gràf. 5.7.** Percentatge de descart de peticions en paquets processats.

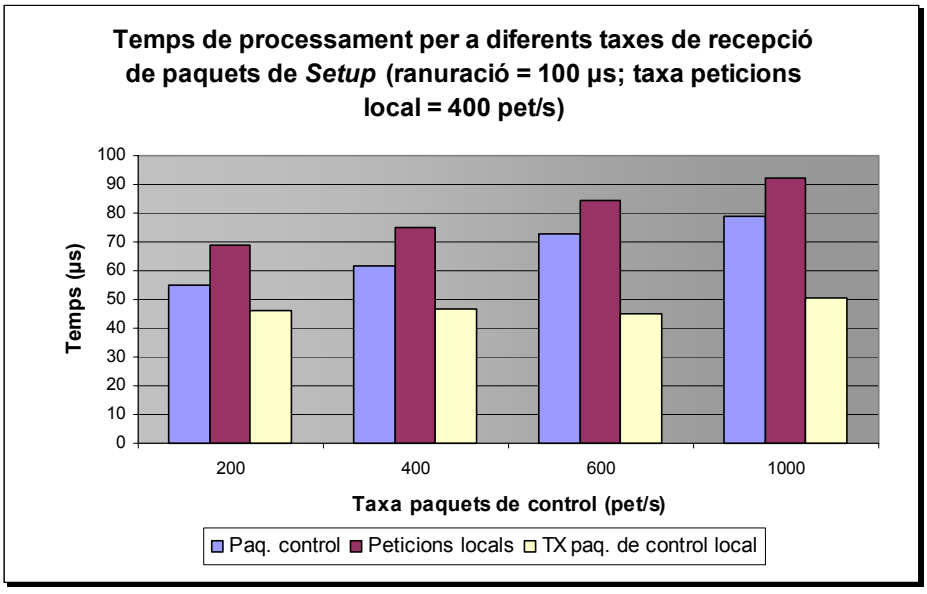
La raó per la qual al disminuir el temps de ranura, la taxa de reserva disminueix, és perquè a ranures majors, es disposa de més temps per a poder processar la correcte reserva d'alguna petició (de les tantes que ens arriben), i per tant, l'efectivitat augmenta.

### 5.2.4. Proves globals

Aquestes proves s'han realitzat per a valorar els temps de processament (de les reserves efectives) i la taxa de reserva quan el node OBS tractava simultàniament peticions locals de transmissió de ràfega i paquets de *Setup* enviats pel PC client.

En el node s'ha realitzat el càlcul de les reserves de forma que no es concep recepció i transmissió en el node OBS al mateix temps per a facilitar-ne el disseny i l'algoritme d'assignació de les reserves en el SRV (tal com s'ha explicat en l'apartat 3.4.1).

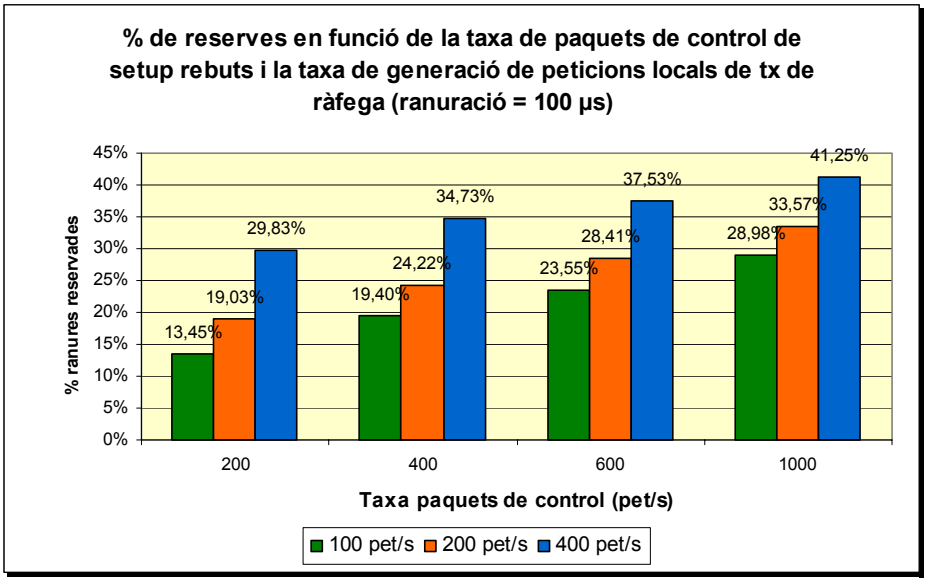
Totes les proves s'han realitzat sobre un temps de ranura de 100 µs. Els valors d'*offset* i longitud de ràfega creats pel programa client són els especificats en l'apartat 5.1. En canvi, en aquestes proves s'han variat les longituds de les peticions de ràfega locals. Ara es realitzen peticions de longituds (alternades) de 4 o 8 ranures. Aquests valors són més equiparables als creats pel programa client, i per tant, els valors comparatius de les taxes de reserva entre els dos seran més equitatius.




**Gràf. 5.8.** Temps de processament per a diferents taxes de recepció de paquets de Setup.

Com es pot veure en el Gràf. 5.8, per a una taxa de peticions locals de 400 pet/s, al augmentar la taxa de paquets de control (de *Setup*) des del PC client,

els temps de processament dels paquets i les peticions augmenta. En canvi, el temps de processament de la transmissió del paquet local es manté quasi estable, sobre els 45-50 µs. Comparativament, s'aprecia com el procés de les peticions locals és major respecte al temps de procés dels paquets de control. Aquests valors, apart del retard que hi pugui haver en executar un major nombre d'instruccions, és també provocat pel flux que hem establert en F.1.3 de l'ANNEX F. En aquest cas, el retard pot augmentar ja que si per a un valor d'*offset* calculat no s'ha trobat reserva possible, es recalcula el valor d'*offset* i es torna a provar. En el nostre cas, el límit d'intents ha estat de 10.




**Gràf. 5.9.** Taxa total de ranures reservades.

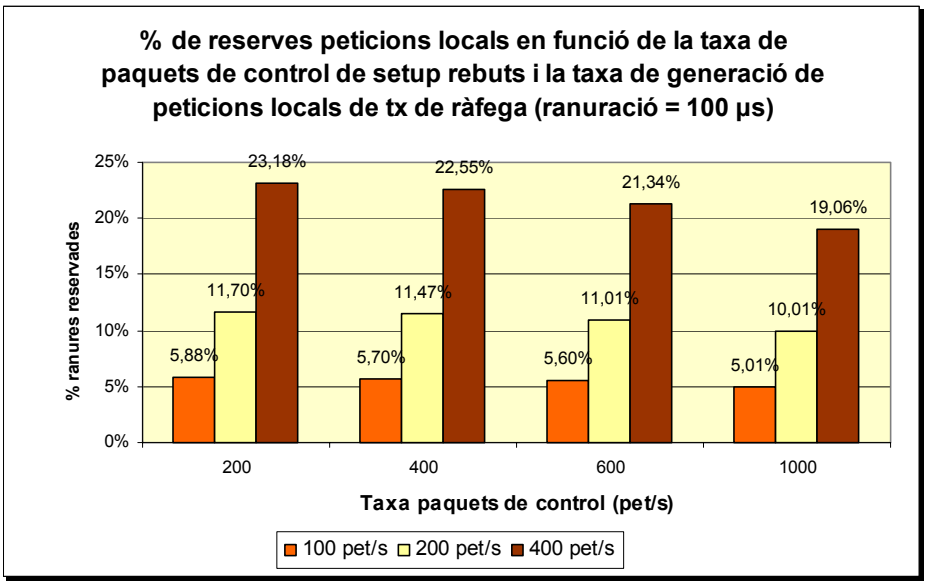
En el Gràf. 5.9 es mostra el percentatge de ranures reservades per a diferents taxes de generació de peticions locals de transmissió de ràfega i paquets de control des del client. En les següents gràfiques, Gràf. 5.10 i Gràf. 5.11, es desglossa el total de reserva segons les peticions locals i els paquets de control.

Respecte els valors de reserva totals, es pot apreciar com per a una taxa de generació de paquets del client determinada, al augmentar la taxa de peticions locals, el % de reserva augmenta. Al augmentar la taxa de peticions, la probabilitat que alguna reserva sigui possible augmenta. Per altra banda, al mantenir una mateixa taxa de generació de peticions locals i augmentar la taxa del client, les reserves també augmenten. Per a taxes elevades de 1000 paq/s i 400 pet/s locals, s'aconsegueix un percentatge de reserva del 41,5% (de ranures efectives).

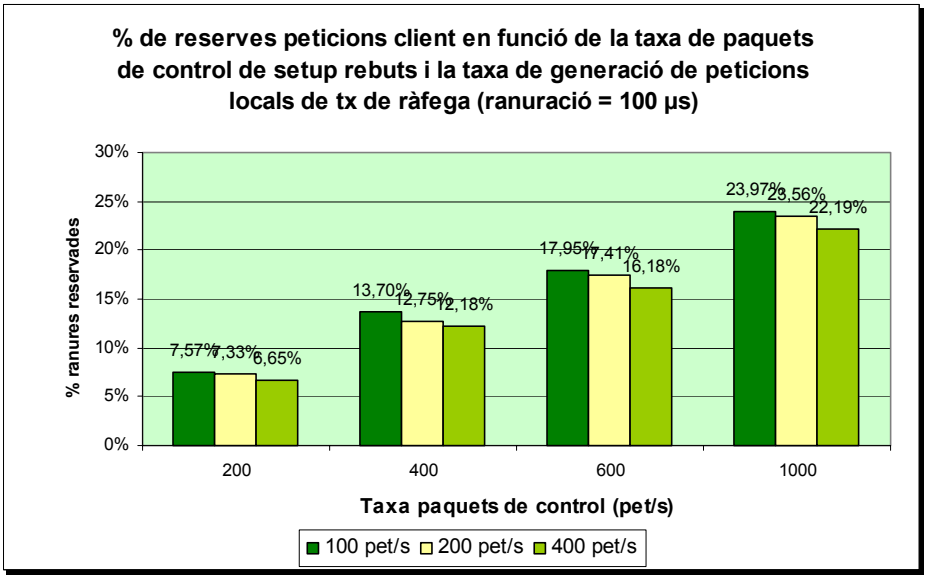
En les següents gràfiques es pot comparar les taxes de reserva per a ambdós processos. D'aquestes es poden extreure alguns resultats interessants:

- Per a una mateixa taxa de paquets de control del client i de peticions locals, la taxa de reserva de ranures per a les peticions locals és major:

  - 200 pet/s i 200 paq/s: reserva local = 11,70%, reserva client = 7,33%.
  - 400 pet/s i 400 paq/s: reserva local = 22,5%, reserva client = 12,18%.
- Al augmentar lleugerament la taxa de paquets del client per sobre de les peticions locals, encara són majors les reserves locals.
- Per a taxes molt més elevades del client (1000 paq/s), les reserves del client guanyen en volum per a qualsevol taxa de peticions locals.




**Gràf. 5.10.** Taxa de ranures reservades per a peticions locals de transmissió de ràfega.



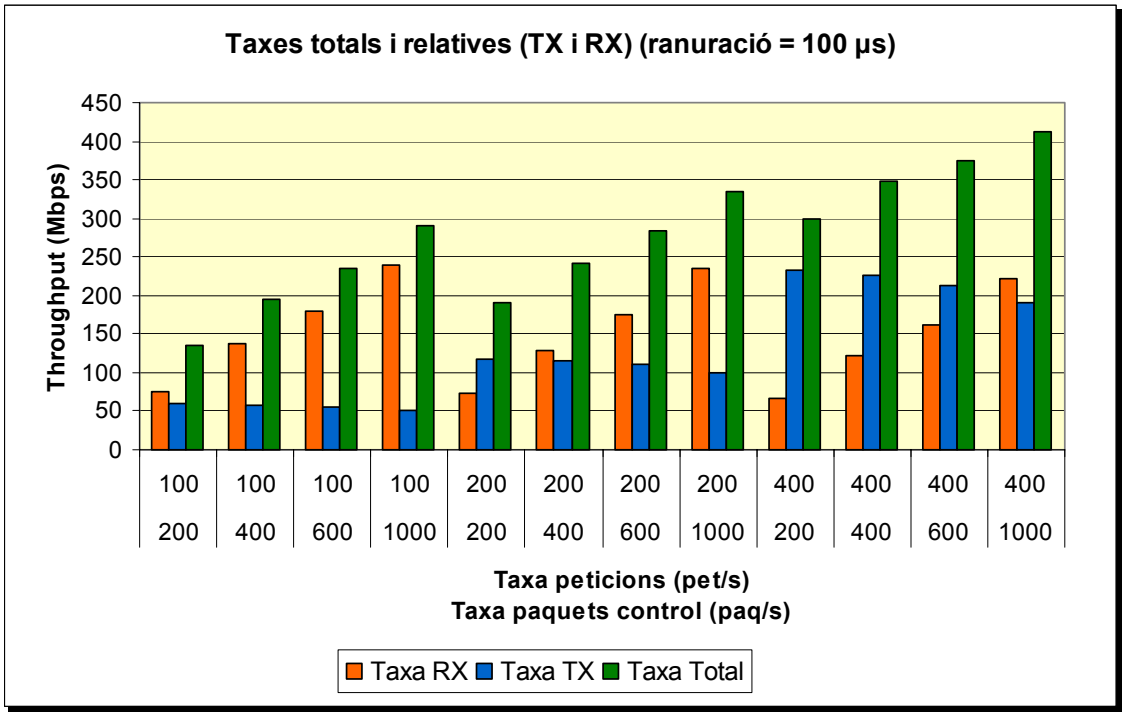

**Gràf. 5.11.** Taxa de ranures reservades per a peticions en paquets de Setup del client.

En les taules de l'apartat H.3.1 es pot apreciar com en mitjana, el nombre de ranures reservades per cada petició local són majors (~ 6 ranures/pet.) a les reserves efectives que aconsegueixen les peticions dels paquets transmesos pel PC client (~ 4 ranures/pet.).

## 5.3. Càlculs de throughput

A partir de les proves anteriors, també es poden realitzar alguns càlculs del *throughput* aconseguit en el node per a la recepció i transmissió de ràfegues. Els seus valors estan molt relacionats amb els obtingut en l'apartat anterior (veure 5.2.4).

En el node s'ha realitzat el càlcul de les reserves de forma que no es concep recepció i transmissió en el node OBS al mateix temps per a facilitar-ne el disseny i l'algoritme d'assignació de les reserves en el SRV. Per tant, s'especifiquen també els totals de *throughput* aconseguits, com la suma de la recepció i transmissió. Aquest valor tindria sentit aplicar-lo al rendiment d'ús aconseguit pel OXC (veure figures Fig. 3.3 i Fig. 3.4).




**Gràf. 5.12.** Throughput total, en recepció i transmissió (Mbps).

La forma en que s'ha calculat el *throughput* de ràfega ha estat la següent (cas del total):

- Primer s'ha calculat la longitud mitja de les ràfegues reservades per a TX i per a RX.

$$\text{Long\_reserva}\left[\frac{\text{ranura}}{\text{reserva}}\right] = \frac{\#\text{ranures}}{\#\text{reserves}} \qquad \textbf{(5.1)}$$

- En segon lloc es calcula el nombre de bits que li corresponen a una ranura (interfície òptica d'1 Gbps).

$$\text{Long\_ranura}\left[\frac{\text{bits}}{\text{ranura}}\right] = \text{Long\_ranura}[\text{seg}] \times \frac{10^9[\text{bit}]}{1[\text{seg}]} \qquad \textbf{(5.2)}$$

- En funció de les reserves efectives que aconsegueixen les peticions, es calcula el nombre de bits reservats aconseguits total. Es calcula aquest valor per al cas de RX i TX.

$$\text{Total\_reserves}[\text{bits}] = \#\text{reserves} \times \text{Long\_reserva} \times \text{Long\_ranura} \qquad \textbf{(5.3)}$$

- Posteriorment es calcula el total de temps aconseguit per a les reserves de TX i RX. Aquest valor, dividint el valor calculat en l'anterior punt proporciona el nombre de bits/s reservats.

$$\text{Temps\_reserves}[\text{seg}] = \frac{\#\text{peticions}}{\text{Taxa\_peticions}\left[\text{pet}/\text{s}\right]} \qquad \textbf{(5.4)}$$

- Finalment es sumen els valors de TX i RX per a obtenir el total.

$$\text{Taxa\_reserva}\left[\frac{\text{bits}}{\text{segon}}\right] = \frac{\text{Total\_reserves}}{\text{Temps\_reserves}} \qquad \textbf{(5.5)}$$

Com es pot veure en el Gràf. 5.12, per a una taxa determinada de peticions locals, al augmentar la taxa de paquets del client, la taxa de RX augmenta. Al mateix temps, al augmentar aquesta taxa, disminueix lleugerament, també, la taxa de transmissió. En el cas extrem de taxes de peticions locals de 400 pet/s i taxes de paquets de control rebuts de 1000 paq/s, s'aconsegueix un *throughput* global que arriba al voltant dels 400 Mbps. De forma individual, ambdues capacitats, TX i RX, arriben sobre els 200 Mbps. Si bé aquests valors poden resultar baixos (comparant amb la capacitat de les interfícies 1 Gbps), aquests resultats depenen en gran mesura dels algoritmes de generació dels *offsets*, i de la forma de crear les ràfegues, dos aspectes que no han estat totalment tractats en el projecte (ja que no era aquest l'objectiu), i per tant, d'aquí el baix rendiment obtingut.

[**NOTA**]: Tots els càlculs anteriors s'han tingut en compte per a interfícies de transmissió/recepció òptiques 1000Base-X.

# CAPÍTOL 6. CONCLUSIONS

En aquest projecte, i en la present memòria escrita, s'ha presentat les bases d'una nova tecnologia de xarxa per a xarxes òptiques basada en la transmissió i commutació de ràfegues de dades. Les xarxes OBS es situen a mig camí entre les xarxes on es realitzen conversions entre dominis òptic i elèctric, O/E/O (*optical-to-electrical-to-optical*), i les totalment òptiques, les OPSNs, on el processament dels paquets es realitza en el propi domini òptic.

El treball s'ha centrat específicament en el disseny i la implementació del Pla de Control per a una xarxa OBS. El disseny d'aquest s'ha basat en la reutilització tant per a nodes frontera, com centrals (o com nodes que actuen de les dues formes). Per al seu disseny s'han estudiat diferents tipus de sistemes de Temps Real, i s'ha vist com els sistemes híbrids de tipus *foreground/background* són una bona opció com a programa de control, ja que el sistema realitza certes tasques de baixa prioritat a la vegada que ha d'atendre les interrupcions degudes a esdeveniments externs (arribades de paquets, peticions de transmissió, etc) que són de major prioritat.

El disseny del programa de control presenta dues parts diferenciades: per un costat hi ha les funcions i tasques que permet realitzar la comunicació i intercanvi de la informació entre els nodes OBS. Això inclou la transmissió, processament i recepció de paquets de control, el manteniment de la informació d'encaminament i commutació, etc. En aquest punt s'ha proposat el protocol de missatges OBS.

Per altra banda hi ha l'algoritme d'atenció a les reserves de recursos. En aquest cas s'ha proposat una solució basada en ranures i en un vector que les allotja. En aquest vector, el SRV (*Slotted Resource Vector*), es marca la ocupació dels recursos, especificant el tipus de senyalització o processament que s'ha de realitzar. El pas d'una ranura a una altra es marca amb l'ús de temporitzadors.

En el projecte s'ha presentat una versió senzilla de l'algoritme per avaluar-ne el seu rendiment, i comprovar si aquesta prova de concepte és factible o no. Per a això, s'ha optat per dissenyar els recursos aplicats a un cas senzill on només es pot realitzar una reserva per ranura. Notar que la forma de definir els recursos té una estreta relació amb el tipus de commutador òptic que el node controla, en el cas que aquest actuï com a node central.

La implementació del Pla de Control s'ha realitzat sobre unes plaques de desenvolupament FPGA d'Avnet. Aquesta tecnologia de disseny hardware permet realitzar totes les modificacions que es requereixin al hardware. Aquesta funcionalitat és primordial a l'hora de realitzar el nostre estudi, ja que es necessiten aquests tipus de facilitats per a trobar i estudiar el millor disseny hardware possible. Per tant, en la implementació també hi ha hagut part de disseny i implementació del hardware. Els IPs de Xilinx han ajudat a integrar els diferents processadors, busos i perifèrics necessaris per a construir el node de

xarxa. Així i tot, la corba d'aprenentatge per a assolir un nivell òptim de coneixements en el disseny hardware i la seva correcta integració és molt elevada.

Sobre un dels processadors PowerPC que integra el xip FPGA de Xilinx s'ha implementat en 'C' el programa de control segons les definicions realitzades en el disseny previ. Per a la seva implementació s'han seguit recomanacions per a optimitzar el seu rendiment, de forma que el flux d'instruccions es reduís el màxim possible. Un menor nombre d'instruccions es redueix en programes que ocupen menor memòria i que realitzen les seves tasques de forma més ràpida. Aquest punt és important, ja que a major temps de servei, la probabilitat de bloqueig de les peticions augmenta.

En el projecte s'han realitzat tests per a comprovar el correcte funcionament del sistema i extreure alguns resultats. Els resultats s'han basat en la obtenció del temps de processament de vàries funcions importants en el tractament dels paquets de control i les peticions de transmissió locals de ràfega. Pel que fa al temps de servei es pot veure com al carregar el sistema amb taxes de petició i paquets de control aquest temps augmenta. Es pot apreciar com en les proves amb ranures de 100 µs, el diferents temps de les funcions estudiades és menor al temps de ranura, però que globalment el supera. Per a això, és de preveure que el rendiment estigui sobre les seves cotes màximes de servei. S'hauran d'estudiar altres possibles solucions per a millorar-lo.

Per altra banda, en les proves s'han extret resultats del rendiment de reserva. Aquests resultats no són definitius i es poden optimitzar molt. La raó principal és que depenen fortament de la forma de càlcul de l'*offset* i la forma en què es generen les ràfegues (el que determina la seva longitud). Aquests dos aspectes no s'han estudiat completament en aquesta primera versió del programa de control, ja que no eren un objectiu primordial. Així i tot, en les proves globals s'han obtingut uns valors de reserva de fins a un 41% sobre el total de ranures disponibles. Aquest % no inclou les ranures que també es reserven i que permeten establir un cert marge de guarda per a que no es sobreescriguin reserves. Aquesta taxa, segons els càlculs realitzats en l'últim apartat, representen un taxa de processament de fins a 400 Mbps.

Com es pot veure, els resultats sobre el funcionament i concepte del programa són força favorables. Així i tot, en el següent apartat, es descriuen les millores de recerca i desenvolupament futur per a millorar el rendiment del programa.

Com a part del projecte s'ha generat un article que serà presentat en el mes de setembre d'aquest any 2006 a Stuttgart, en el congrés EUNICE.

## 6.1. Línies futures de recerca i desenvolupament

Es poden dividir les línies de recerca futures en dos grans blocs. Un primer bloc es centra en optimitzar el rendiment del Pla de Control de forma individual. L'altre bloc es centra en realitzar la integració del Pla de Control amb la resta dels elements que conformen els nodes OBS, és a dir, unir el nostre disseny

amb el Pla de Dades i el commutador òptic. Una vegada estiguin integrats, s'haurà de provar el correcte funcionament del node, i finalment, montar la maqueta de xarxa que s'ha plantejat en la memòria del projecte.

Pel que fa al primer bloc les millores proposades són vàries. Per a millorar el rendiment es pot realitzar l'aprofitament dels dos processadors PowerPC 405 que conté el xip FPGA de Xilinx. Un processador podria encarregar-se de realitzar el seguiment i processar l'algoritme de reserves del SRV. L'altre processador podria utilitzar-se per a processar les peticions de transmissió de ràfega i les que provenen d'altres nodes de la xarxa en forma de paquets de control.

Per altra banda també s'han de realitzar optimitzacions i estudis més complets del rendiment sobre varis tipus de memòria, per així extreure una solució que sigui compatible amb la integració de les altres parts que conformen el node. Això és important, ja que és de preveure que el Pla de Dades necessitarà de grans memòries per a poder realitzar l'emmagatzemament de les ràfegues. Així mateix, també s'ha de realitzar una proposta de millora del SRV per a que pugui contenir diferent tipus de reserves, si així és requereix. S'hauria d'estudiar la forma de com es resolen les contencions de reserva que són compatibles de commutar, transmetre o rebre simultàniament.

Finalment, per a fer completament autònom els nodes de xarxa s'ha d'annexar algun programa o mòdul que pugui realitzar l'actualització dinàmica de les taules d'encaminament.

## 6.2. Impacte mediambiental

Com a punt final a aquesta memòria es realitza un breu estudi de l'impacte mediambiental en la realització del projecte. Un dels punts forts del projecte és la utilització de la tecnologia FPGA. Aquesta permet reutilitzar els xips per a redissenyar i reimplementar el hardware, i per tant, un mateix xip pot utilitzar-se infinitat de vegades. Això permet que la generació de residus sòlids sigui quasi nul, ja que no s'ha de construir un xip cada cop que el disseny canvia, i per tant, l'impacte mediambiental disminueix considerablement.

Un altre consideració futura en relació al desenvolupament de les xarxes òptiques de fibra és si representen en la seva construcció un fort impacte. Si bé en la seva construcció es generen enderrocs, molts cops s'aprofita la pròpia construcció d'una altra infraestructura annexa, per exemple, vies de ferrocarril o autopistes, per a desplegar la fibra. Així, per si sola, la construcció de la xarxa de fibra òptica produeix un menor impacte mediambiental.

Finalment, el punt central del nostre projecte ha estat la implementació del programa de software, i com a tal, el seu propi impacte és baix, a no ser que es comptabilitzi la despesa energètica que pressuposa la seva programació i posterior posada en funcionament.

# REFERÈNCIES

[1] C. Gauger et al., “Service Differentiation in Optical Burst Switching Networks”, *ITG Fachtagung Photonische Netze*, pp. 124-132, Dresden, March 2001.

[2] C. Qiao and M. Yoo, “Optical Burst Switching (OBS) – A New Paradigm for an Optical Internet”, *J. High Speed Nets*, vol. 8, no. 1, pp. 69-84, Amsterdam, Jan. 1999.

[3] Fei Xue, S. J. B. Yoo, H. Yokoyama, and Y. Horiuchi, “Performance comparison of optical burst and circuit switched networks”, *Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC*, vol. 3, pp. 3, March 2005.

[4] K. Dolzer and C. Gauger, “On Burst Assembly in Optical Burst Switching Networks – A Performance Evaluation of Just-Enough-Time”, in 17th Int. Teletraffic Congress, pp. 149-160, Salvador, Brazil, Sept. 2001.

[5] V. Vokkarane and J. P. Jue, “Prioritized Burst Segmentation and Composite Burst Assembly Techniques for QoS Support in Optical Burst Switched Networks”, *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 7, pp. 1198-1209, Sept. 2003.

[6] I. Widjaja, “Performance Analysis of Burst Admission Control Protocols”, *IEE Proc. Communications*, vol. 142, no. 1, pp. 7-14, Feb. 1995.

[7] M. Düser and P. Bayvel, “Analysis of a dynamically wavelength-routed optical burst switched network architecture”, *IEEE Journal of Lightwave Technology*, vol. 20, no. 4, pp. 574-585, Apr. 2002.

[8] M. Yoo, C. Qiao, “Just-enough-time (JET): a high speed protocol for bursty traffic in optical networks”, *Proceeding of IEEE/LEOS Conf. On Technologies for a Global Information Infraestructure*, pp. 26-27, Aug. 1999.

[9] S. Verma, H. Chaskar, and R. Ravikanth, “Optical Burst Switching: A Viable Solution for Terabit IP Backbone”, *IEEE Network*, vol. 14, no. 6, pp. 48-53, Nov. 2000.

[10] I. Baldine, G. N. Rouskas, H. G. Perros, and D. Stevenson, “JumpStart: A just-in-time signaling architecture for WDM Burst-Switched Networks”, *IEEE Communications*, 40(2), pp. 82-89, Feb. 2002.

[11] J. Y. Wei and R. I. McFarland, “Just-in-time signaling for WDM optical burst switching networks”, *IEEE Journal of Lightwave Technology*, vol. 18, no. 12, pp. 2019-2037, Dec. 2000.

[12] J. S. Turner, "Terabit Burst Switching", *Journal of High Speed Networks*, vol. 8, no. 1, pp. 3-16, Jan. 1999.

[13] J. M. Yates, J. P. R. Lacey, M. P. Rumsewicz, and M. A. Summerfield, "Performance of networks using wavelength converters based on four-wave mixing in semiconductor optical amplifiers", *IEEE J. Lightwave Technology*, vol. 17, pp. 782-791, May 1999.

[14] M. Karásek, L. A. Rusch, and M. Menif, "Suppresion of output power and NF excursions in cascades of highly inverted EDFAs with packet switched traffic", *Fiber and Integrated Optics*, vol. 20, pp. 269-277, 2001.

[15] I. Baldine, M. Cassada, A. Bragg, G. Karmous-Edwards, and D. Stevenson, "Just-in-time optical burst switching implementation in the ATDnet all-optical networking testbed", in *Proceedings GLOBECOM'03*, vol. 5, pp. 2777-2781, San Francisco, Dec. 2003.

[16] Y. Sun, T. Hashiguchi, V. Q. Minh, X. Wang, H. Morikawa, and T. Aoyama, "Design and Implementation of an Optical Burst-Switched Network Testbed", *IEE Optical Communications*, vol. 3, no. 4, pp. S44-S55, Nov. 2005.

[17] Xilinx, Inc., *PowerPC 405 Processor* [en línia] [Consulta: Abril 2006]. Disponible a: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/capabilities/powerpc.htm

[18] Laplante, Phillip A., "Real-Time Operating Systems", cap. 3 en *Real-Time Systems Design and Analysis*, IEEE Press, 1st. Edition, pp. 73-88, New Jersey, 2004.

[19] CIVCOM, *Ultra Fast Optical Switch* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.civcom.com/Free_light.asp?MainID=11&Name=Free-X%20Familiy

[20] IEEE 802.3 Working Group, *IEEE 802.3 CSMA/CD (Ethernet)* [en línia] [Consulta: Agost 2006]. Disponible a: http://grouper.ieee.org/groups/802/3

[21] E. Rosen, A. Viswanathan, and R. Callon, *RFC 3031: Multiprotocol Label Switching Architecture*, Network Working Group, Internet Society (2001), Jan. 2001.

[22] Xilinx, Inc., *PLB 1-Gigabit Ethernet Media Access Controller (MAC) with DMA (v1.00b)*, DS460 [en línia] [Consulta: Maig 2006]. Disponible a: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/plb_gemac.pdf

[23] Avnet, Inc., *Xilinx Virtex-II Pro Development Kit* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.silica.com/en/products/evaluationkits/evaluationkit11.html

[24] Xilinx, Inc., *Virtex-II Pro / Pro X FPGAs* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/index.htm

[25] Barr, Michael, *How Programmable Logic Works* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.netrino.com/Articles/ProgrammableLogic/index.html

[26] Xilinx, Inc., *Design Tools* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.xilinx.com/products/design_resources/design_tool/index.htm

[27] The Eclipse Foundation [en línia] [Consulta: Agost 2006]. Disponible a: http://www.eclipse.org

[28] Xilinx, Inc., *Intellectual Property* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.xilinx.com/ipcenter/

[29] Xilinx, Inc., "PLB Usage in Xilinx FPGAs", Part I de *Processor IP Reference Guide* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.xilinx.com/ise/embedded/EDK/proc_ip_ref_guide.pdf

[30] Ghosh, K., *Writing Efficient C and C Code Optimization*, The Code Project [en línia] [Consulta: Maig 2006]. Disponible a: http://www.codeproject.com/cpp/C___Code_Optimization.asp

[31] Davis, G., *Tips for efficient embedded code generation*, Embedded.com [en línia] [Consulta: Maig 2006]. Disponible a: http.//www.embedded.com/showArticle.jhtml?articleID=170704298

[32] Edwards, Stephen A., *Low-Level C Programming*[en línia], Columbia University, 2005, [Consulta: Maig 2006]. Disponible a: http://www1.cs.columbia.edu/~sedwards/classes/2005/emsys-summer/c-programming.pdf

[33] Ethereal, [en línia] [Consulta: Agost 2006]. Disponible a: http://www.ethereal.com

[34] TCPReplay, [en línia] [Consulta: Agost 2006]. Disponible a: http://tcpreplay.synfin.net/trac/

[35] Libnet, [en línia] [Consulta: Maig 2006]. Disponible a: http://www.packetfactory.net/libnet

[36] Schiffman, Mike D., *The Libnet Reference Manual v.02* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.packetfactory.net/Projects/Libnet/manual/

[37] Raynal, F., *Building packets for dummies and others with libnet*, Security-labs.org, [en línia] [Consulta: Agost 2006]. Disponible a: http://www.security-labs.org/index.php3?page=libnet

[38] Libpcap, [en línia] [Consulta: Maig 2006]. Disponible a: http://www.tcpdump.org

[39] López Monge, A., *Aprendiendo a programar con Libpcap* [en línia], Feb. 2005, [Consulta: Maig 2006]. Disponible a: http://www.e-ghost.deusto.es/docs/2005/conferencias/pcap.pdf

[40] Xilinx, Inc., *RocketIO Transceiver User Guide*, UG024 (v2.5), [en línia] [Consulta: Agost 2006]. Disponible a: http://direct.xilinx.com/bvdocs/userguides/ug024.pdf

[41] National Semiconductor, *DP83865 – Gig PHYTER V 10/100/1000 Ethernet Physical Layer* [en línia] [Consulta: Agost 2006]. Disponible a: http://www.national.com/ds.cgi/DP/DP83865.pdf

# GLOSSARI

## A

| | |
|---|---|
| AON | All-Optical Network |

## B

| | |
|---|---|
| BRAM | Block Random Access Memory |

## C

| | |
|---|---|
| CLB | Configurable Logic Block |
| CoS | Class-of-Service |
| CPLD | Complex Programmable Logic Device |
| CPU | Central Processing Unit |

## D

| | |
|---|---|
| DCM | Digital Clock Manager |
| DCR | Device Control Register |
| DDR | Double-Data-Rate |

## E

| | |
|---|---|
| EDFA | Erbium Doped Fibre Amplifiers |
| EUNICE | EUropean Network of universities and companies in Informatio and Communication Engineering |

## F

| | |
|---|---|
| FCS | Frame Checksum Sequence |
| FIFO | First In First Out |
| FPGA | Field Programmable Logic Array |

## G

| | |
|---|---|
| GEMAC | Gigabit Ethernet Medium Access Controller |
| GMII | Gigabit Media Independent Interface |

## H

| | |
|---|---|
| HDL | Hardware Description Language |
| HSSDC2 | High Speed Serial Data Connection 2 |

## I

| | |
|---|---|
| IDE | Integrated Development Environment |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISR | Interrupt Service Routine |

## J

| | |
|---|---|
| JET | Just-Enough-Time |
| JIT | Just-In-Time |
| JTAG | Joint Test Action Group |

## M

| | |
|---|---|
| MAC | Medium Access Controller |
| MGT | Multi-Gigabit Transceiver |
| MII | Media Independent Interface |
| MPLS | Multi Protocol Label Switching |
| MSR | Machine State Register |

## O

| | |
|---|---|
| OBS | Optical Burst Switching |
| OCPS | OBS Control Plane Software |
| OOP | Object-Oriented Programming |
| OPB | On-chip Peripheral Bus |
| OPSN | Optical Packet Switched Network |
| OXC | Optical Cross-Connection |

## P

| | |
|---|---|
| PLB | Processor Local Bus |
| PPC | PowerPC |

## R

| | |
|---|---|
| RISC | Reduced Instruction Set Computer |
| RTOS | Real Time Operating Systems |

## S

| | |
|---|---|
| SDRAM | Synchronous Dynamic Random Access Memory |
| SERDES | SERializer/DESerializer |
| SFD | Start-of-Frame Delimiter |
| SFP | Small Form-factor Pluggable |
| SRAM | Static Random Access Memory |
| SRR | Save/Restore Register |
| SRV | Slotted Resource Vector |

## T

| | |
|---|---|
| TAW | Tell-And-Wait |

## V

| | |
|---|---|
| VHDL | Very High Speed Integrated Circuit Hardware Description Language |

## W

| | |
|---|---|
| WDM | Wavelength-Division Multiplexing |
| WRN | Wavelength-Routed Network |

## X

| | |
|---|---|
| XMD | Xilinx Microprocessor Debugger |
| XPS | Xilinx Platform Studio |

Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# ANNEXOS

**TÍTOL DEL TFC/PFC: Construcció d'una maqueta de xarxa OBS. Disseny i Implementació del Pla de Control.**

**TITULACIÓ: Enginyeria de Telecomunicació (segon cicle)**

**AUTOR: Joan Triay Marquès**

**DIRECTOR: Cristina Cervelló i Pastor**

**DATA: 8 de setembre de 2006**

# INDEX

# ÍNDEX DE FIGURES i GRÀFICS

# ÍNDEX DE TAULES

# ANNEX A. SISTEMES DE TEMPS REAL

Aquest annex és un resum de conceptes utilitzats amb assiduïtat en la descripció de Sistemes de Temps Real. Es presenten algunes definicions importants, així com la descripció de diferents tipus de planificacions d'aquests sistemes.

Aquest annex complementa les descripcions realitzades en el CAPÍTOL 2 de la memòria.

## A.1. Conceptes de Sistemes de Temps Real

A continuació descriurem alguns conceptes importants associats als Sistemes de Temps Real.

### A.1.1. Tasques

La solució correcte per a solucionar les diferents funcions que ha de realitzar un sistema en temps real és la creació d'un sistema multitasca. La multitasca és un paradigma molt potent per a estructurar sistemes de temps real basats en interrupcions.

La idea bàsica és que es pot partir un problema gran en un conjunt de problemes més simples. Cadascun d'aquests petits problemes constitueix una tasca. Cada tasca realitza un funció principal per tal de mantenir-la el més simple possible. Llavors, el que es pretén és que totes aquestes tasques s'executin paral·lelament. De fet, no s'executen paral·lelament a no ser que el sistema sigui multiprocessador (en un sol processador les tasques comparteixen el processador).

Com qualsevol programa, una tasca conté codi que duu a terme la funció per la qual la tasca ha estat dissenyada. Aquest codi està incrustat en una funció de forma anàloga a la funció *main* en un programa en C. El que fa que una tasca sigui diferent d'una funció ordinària és que cada tasca té un context incrustat en la seva pròpia pila (veure Fig. A.1).

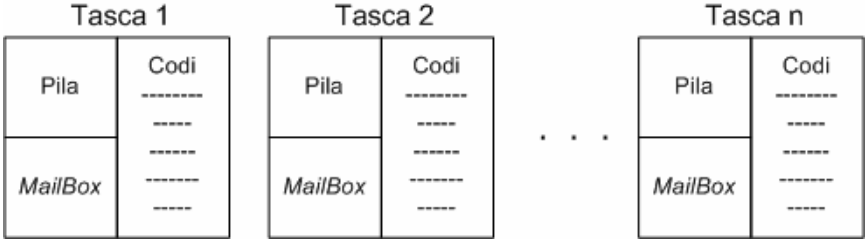



**Fig. A.1.** Composició de les tasques.

Per tant, cada tasca conté:

- Codi que aporta la funcionalitat bàsica de la tasca.
- Una pila, o *stack*, que manté el context.
- Una bústia, o *mailbox*, que permet que una tasca pugui comunicar-se amb altres tasques.

S'ha de notar que és possible, i algunes vegades molt útil, el crear múltiples tasques des de la mateixa funció. El que fa que cada tasca actuï per separat i de forma distinta és que cadascuna té la seva pròpia pila. Com es pot veure, aquest paradigma s'assembla molt al de la clàssica programació orientada a objectes (OOP).

Els arguments de dades proporciona un mètode per a parametritzar la tasca Això pot ser molt útil si múltiples tasques deriven de la mateixa funció. El que fa única cada tasca és a causa del valor del seu argument.

Una tasca pot començar amb algun tipus d'inicialització, després de la qual, sol entrar en un *loop* infinit. En algun punt del *loop*, normalment en la part alta, la tasca "espera que algun esdeveniment passi". Mentre està esperant, la tasca no s'executa, no utilitza el processador. En aquest cas, alguna altra tasca es pot estar executant i utilitzant el processador.

Si de forma eventual l'esdeveniment que la tasca estava esperant succeeix, llavors la tasca "desperta", i actua segons el seu protocol.

Notar que la raó per la qual el paradigma multitasca funciona és perquè la majoria de les tasques estan la major part del temps esperant que algun esdeveniment succeeixi.

### A.1.2. Planificació

Les tasques operen sota la supervisió d'un *kernel* en temps real que consisteix en:

- Una col·lecció de serveis que implementen funcionalitats com la comunicació entre tasques i la sincronització.
- Un planificador, la funció del qual és la d'assegurar que la tasca de major prioritat en el moment sigui la que s'estigui executant en un determinat instant.

El planificador tracta cada tasca com una màquina d'estats. Mentrestant, cada *kernel* té per si mateix, i normalment més complex, un model d'estats. La següent figura Fig. A.2 mostra una màquina d'estats senzill.

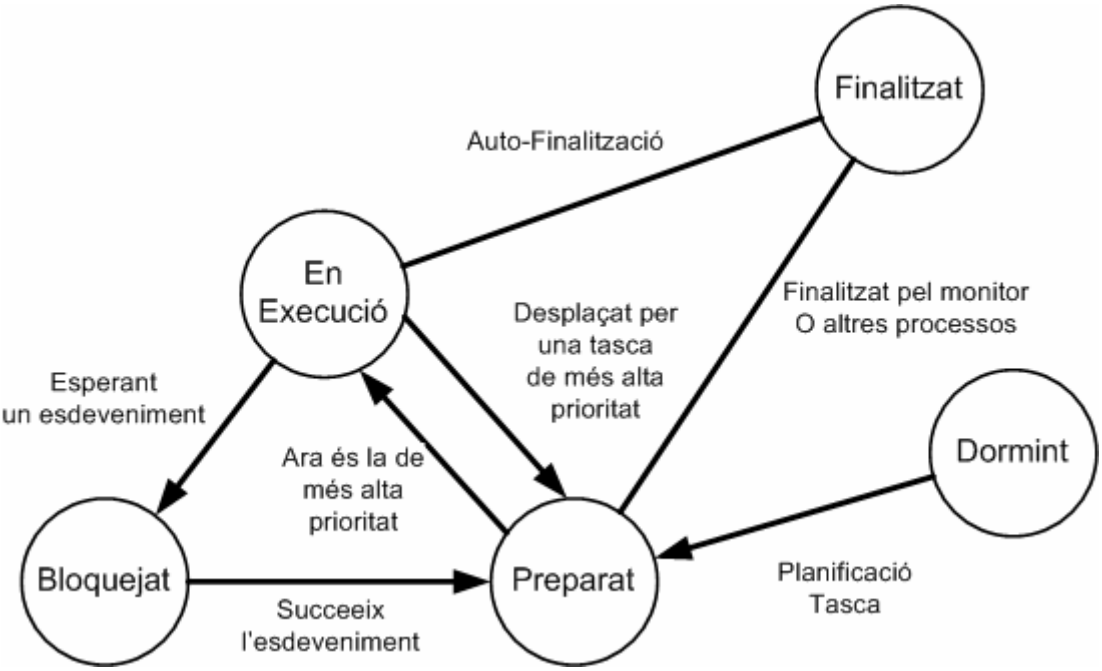



**Fig. A.2.** Estat de les tasques en un Sistema de Temps Real.

- **En execució:** Només una tasca, l'actual tasca en execució, pot estar en l'estat de "En execució". Una tasca pot passar voluntàriament de l'estat d'execució a l'estat de "Blocat" esperant un esdeveniment. En un sistema de prioritats, el planificador pot provocar que la tasca en execució passi a l'estat "Preparat" si una tasca de major prioritat retorna de l'estat preparat.

- **Preparat:** La tasca està preparada, però té una prioritat més baixa que la tasca que s'està executant en aquell instant. La tasca passarà de preparat a en execució quan sigui la tasca de major prioritat en estat preparat.

- **Blocat:** Una tasca blocada està esperant que algun esdeveniment succeeixi. Quan l'esdeveniment succeeix, la tasca passa a l'estat de preparat.

- **Dormint:** La tasca ha estat creada i inicialitzada. No està llesta per a executar-se, això és, en aquest estat, el procés no pot executar-se.

- **Finalitzat:** El procés ha finalitzat la seva execució, o s'ha auto-finalitzat.

### A.1.3. Tipus de Planificacions

Per a poder manegar els Sistemes de Temps Real es necessiten d'uns gestors o planificadors de tasques. En els següents subapartats es realitza una breu descripció dels més comuns.

### *A.1.3.1. Planificació periòdica*

Hi ha moltes tasques que simplement requereixen que siguin despertades periòdicament, realitzar alguna funció i tornar a "dormir-se". Hi ha un parell d'aproximacions a la planificació de tasques periòdiques, tal com es mostra en la següent figura Fig. A.3.

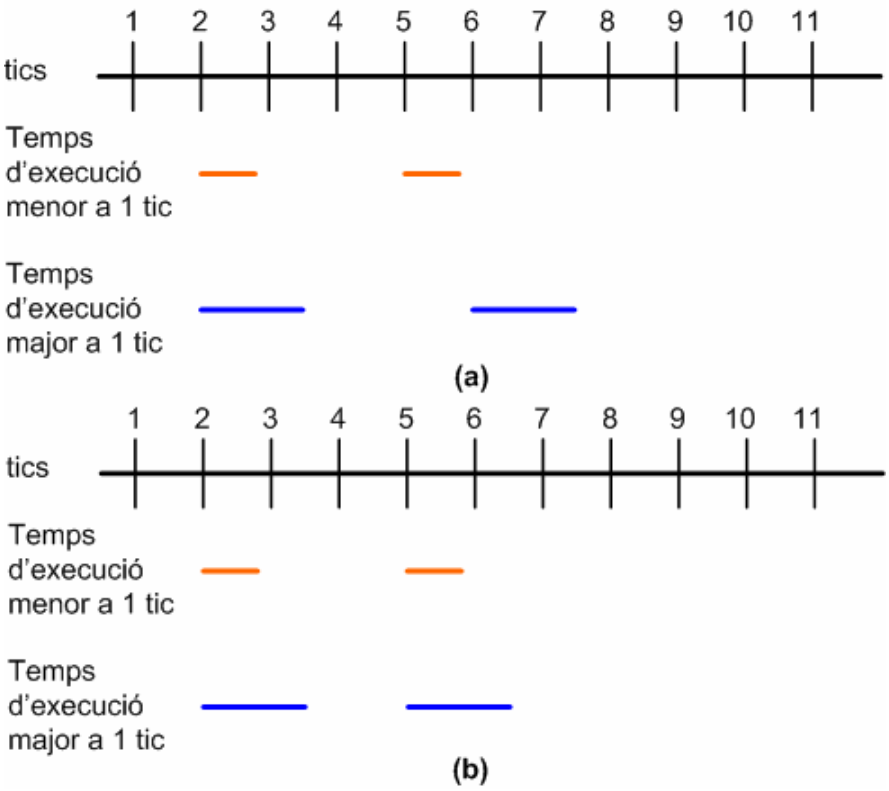



**Fig. A.3.** Tasques periòdiques.

El comportament del sistema depèn del temps d'execució de cada tasca. Si el temps d'execució és menor al temps d'un tic de rellotge, llavors la tasca es desperta cada tres tics de rellotge. No obstant, si el temps d'execució és major a un tic, quan la tasca crida al sistema per retardar-se, estarà encara bloquejat per tres tics. D'aquesta forma, en aquest exemple, la tasca es desperta cada 4 tics, i això no és el que es vol.

Una opció alternativa, que no està disponible en tots els sistemes, és declarar la tasca com a periòdica. En aquest cas, el planificador desperta la tasca en l'interval apropiat sigui quin sigui el temps d'execució de la tasca.

### *A.1.3.2. Planificació no periòdica*

Altres tasques han de respondre a esdeveniments que sorgeixen de forma aleatòria. Un esdeveniment es pot generar al arribar un paquet de xarxa, el tancament d'un commutador, o qualsevol altra acció. Molt sovint, aquests

esdeveniments asíncrons es comuniquen amb el processador mitjançant interrupcions. La rutina de servei d'interrupcions (*Interrupt Service Routine*, ISR) ha de tenir alguna forma per comunicar que una interrupció ha succeït a una tasca que és la responsable de servir l'esdeveniment.

### *A.1.3.3. Planificació per prioritats vs. no prioritats*

Hi ha dues estratègies fonamentals per a la planificació de tasques: per prioritats vs. no prioritats. Considerem dues tasques, on s'està executant en un moment determinat la tasca de més baixa prioritat i la tasca de més alta prioritat està bloquejada (està esperant un esdeveniment que es senyalitzarà mitjançant una interrupció).

En la següent figura Fig. A.4 es mostra què passa en el cas sense prioritats. La ISR provoca que la tasca de major prioritat passi a l'estat *Ready*, però al final del ISR, el control retorna a la tasca de més baixa prioritat que havia estat interrompuda. Més tard, quan la tasca de més baixa prioritat es bloqueja esperant un esdeveniment, la tasca de més alta prioritat passa a l'estat *Running*.

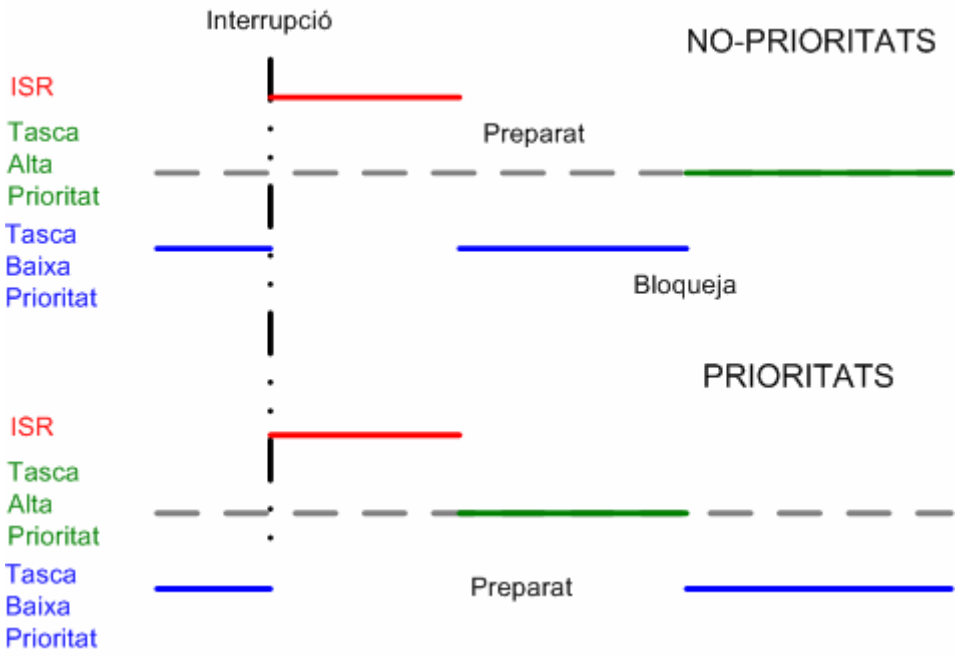



**Fig. A.4.** Planificació per prioritats vs. no-prioritats.

En la figura Fig. A.4 es mostra un cas d'ús amb prioritats. La diferència aquí és que el planificador s'invoca al final del ISR. Aquest determina que la tasca de més alta prioritat està preparada i commuta les tasques segons l'acordat. La tasca de més baixa prioritat és llavors reemplaçada per la de més alta prioritat.

Els sistemes amb prioritats proporcionen temps de resposta més predibles perquè l'esdeveniment d'alta prioritat es serveix immediatament. En essència,

això és el que determina que un sistema sigui de temps real, ja que es garanteix el temps màxim que es pren a l'hora de respondre a un esdeveniment. En el cas que no hi hagi prioritats no es garanteix el temps consumit abans que la tasca en execució deixi lliure el processador. D'altra banda, els sistemes per prioritats estan subjectes a problemes i conflictes en l'ús dels recursos.

Entre tasques de mateixa prioritat es poden utilitzar altres estratègies en la planificació. En la planificació *round robin*, una tasca s'executa fins que es bloqueja esperant per un esdeveniment o deixa voluntàriament el processador. La distinció entre bloquejar-se i cedir el processador és que en l'últim cas la tasca roman en estat *Ready*.

La fragmentació del temps és una variació del *round robin* que assigna un temps màxim, ranura, o *slot* a cada tasca per evitar que la tasca s'allargui molt en el seu temps de processament. Una tasca s'executa fins que es bloqueja, cedeix voluntàriament o expira el seu temps o *slot* d'execució. Depenent de la implementació, el temps fragmentat pot ser el mateix per a totes les tasques o de diferent valors segons cada tasca.

# ANNEX B. PROCESSADOR POWERPC 405

En aquest annex s'ofereix informació del processador PowerPC 405 integrat dins el xip FPGA Virtex-II Pro 30 de Xilinx. L'annex es centra, principalment, en aportar informació relativa al tractament de Sistemes de Temps Real, concepte molt utilitzat al llarg de la memòria del PFC. Gran part de la informació d'aquest annex es troba disponible en les pàgines web de Xilinx, i també d'IBM (fabricant d'aquests tipus de CPUs).

## B.1. Sistemes de Temps Real en el PowerPC 405

En aquest apartat introduirem part del llenguatge particular del sistemes de temps real aplicats al processador PowerPC 405 d'IBM [17] de tipus RISC.

### B.1.1. Definicions de l'entorn PPC405

#### *B.1.1.1. Excepcions*

Les Excepcions són esdeveniments detectats pel processador que sovint requereixen alguna acció per part del software de sistema. La majoria d'excepcions són inesperades i són el resultat de diverses condicions d'error. Algunes excepcions es poden programar per a que sorgeixin mitjançant l'ús d'instruccions. Algunes excepcions són programades per dispositius externs i comunicades al processador utilitzant senyalització externa.

#### *B.1.1.2. Interrupcions*

Les Interrupcions són transferències de control automàtic que sorgeixen com a resultat a una excepció. Una interrupció succeeix quan el processador suspèn l'execució d'un programa després de detectar una excepció. El processador guarda la màquina d'estats del programa suspès i una adreça de retorn al programa suspès. La informació es guarda en un parell de registres especials, anomenats *save/restore registers* (registres de guarda/restauració). El processador carrega una màquina d'estats, la qual transfereix el control a un manegador d'interrupcions o, *interrupt handler*.

#### *B.1.1.3. Interrupt Handlers*

Un *interrupt handler* és una rutina software de sistema que respon a una interrupció, sovint corregint la condició que genera l'excepció. El software del sistema disposa els *handlers* d'interrupcions en adreces predefinides en la memòria física i el mecanisme d'interrupcions transfereix el control automàticament al *handler* apropiat basant-se en la condició que genera l'excepció.

Una interrupció posiciona el processador en els modes privilegiat i real (la recol·locació de les adreces d'instruccions i dades es deshabilita). Les interrupcions són esdeveniments amb sincronització contextual. Totes les instruccions que precedeixen la instrucció interrompuda es garanteixen que tinguin la seva execució completa quan la interrupció succeeix. Totes les instrucció que segueixen a la instrucció interrompuda (en el flux de programa) es descarten.

El retorn d'un *handler* d'interrupcions a un programa interromput requereix que la màquina d'estats antiga i l'adreça de retorn del programa es restaurin del parell de registres *save/restore*. Això es compleix utilitzant la instrucció *return-from-interrupt*. Com en les interrupcions, les instruccions de la *return-from-interrupt* estan sincronitzades contextualment.

Algunes interrupcions es poden deshabilitar (marcar) o habilitar (desmarcar). La deshabilitació d'interrupcions provoca que es generin les interrupcions quan es detecta la condició d'excepció.

### *B.1.1.4. Interrupcions precises i imprecises*

La majoria d'interrupcions són precises. Una interrupció precisa succeeix en l'ordre del programa i en els límits de la instrucció on l'excepció es reconeix. Una interrupció precisa provoca el següent:

- L'adreça de retorn apunta a la instrucció amb excepcions. Per a excepcions síncrones, l'adreça de retorn apunta tant a la instrucció que ha causat l'excepció o la instrucció que segueix immediatament, dependent de la condició d'excepció. Per a les excepcions asíncrones, l'adreça de retorn apunta a la condició d'instrucció on l'excepció ha estat reconeguda pel processador.

- Totes les instruccions que precedeixen la instrucció d'excepció completen la seva execució abans que la interrupció succeeixi. No obstant, és possible que alguns accessos a emmagatzemament iniciats abans de les instruccions no estiguin completats respecte a l'estat dels dispositius externs.

- Depenent de la condició d'excepció, és possible que la instrucció d'excepció hagi completat la seva execució, s'hagi completat parcialment, o que no hagi començat.

- Cap instrucció que segueixi a la instrucció d'excepció s'executa abans de transferir el control al *handler* d'interrupcions.

Quan una interrupció imprecisa succeeix, la instrucció que provoca l'excepció no està relacionada amb la condició d'excepció. En aquest cas, hi ha una retràs entre el punt on l'excepció és reconeguda pel processador i el temps en què la interrupció succeeix. Una interrupció imprecisa provoca el següent:

- La instrucció d'excepció segueix (en l'ordre del programa) la condició d'instrucció on l'excepció és reconeguda pel processador. El retràs pot incloure diverses instruccions.

- Totes les instruccions que precedeixen la instrucció d'excepció completen la seva execució abans que la interrupció succeeixi. No obstant, és possible que alguns accessos a dades iniciats per les instruccions no hagin finalitzat respecte a l'estat dels dispositius externs.

- És possible que la instrucció d'excepció hagi completat la seva execució, l'hagi completat parcialment o que no hagi començat l'execució.

- Cap instrucció que segueixi la instrucció amb excepció és executada abans de la transferència de control al *handler* d'interrupcions.

En el PPC405, només la interrupció de comprovació de màquina és imprecisa, la qual pot estar causada indirectament per l'execució d'una instrucció. En aquest cas, és possible que el processador executi instruccions addicionals abans de reconèixer que ha succeït una comprovació de màquina.

#### *B.1.1.5. Excepcions crítiques i no crítiques*

El PPC405 suporta excepcions crítiques i no crítiques. Generalment, el processador respon a excepcions crítiques abans que a les excepcions no crítiques (algunes excepcions de depuració es maneguen a baixa prioritat).

Hi ha quatre excepcions (i les seves interrupcions associades) crítiques:

- Excepció d'entrada crítica (*Critical Exception*).
- Excepció de comprovació de màquina (*Machine Check*).
- Excepció de temporitzador de vigilància (*Watchdog Timer*).
- Excepció de depuració (*Debug*).

Les interrupcions crítiques utilitzen un parell de registres de *save/restore* diferents (SRR2 i SRR3) que els utilitzats per les interrupcions no crítiques (SRR0 i SRR1). Això habilita a una interrupció crítica poder interrompre a un *handler* d'interrupcions no crítiques. La interrupció crítica no sobreescriu l'estat guardat per la interrupció no crítica.

En la Taula B.1 es presenta una descripció més detallada de les excepcions que suporta el PowerPC 405 i el seu grau de criticisme.

### B.1.2. Interfície Controladora d'interrupcions Externes

Es pot utilitzar lògica externa al bloc del processador per a provocar interrupcions crítiques i no crítiques. Les fonts d'interrupció externes es poden recol·lectar totes en un *External Interrupt Controller* (EIC, Controlador d'interrupcions externes) i llavors presentar-les al processador. Quan la petició

d'una interrupció externa s'activa, el EIC ha de mantenir el senyal activat fins que el software el desactivi.

El software pot habilitar i deshabilitar interrupcions externes utilitzant els següents bits en el Registre de Màquina d'Estats (MSR):

- Les interrupcions no crítiques es controlen amb el MSR[EE]. Quan s'activa a "1", les interrupcions no crítiques estan habilitades. Quan està a "0", estan deshabilitades.

- Les interrupcions crítiques es controlen amb el MSR[CE]. Quan s'activa a "1", les interrupcions crítiques estan habilitades, i "0" altrament.

### B.1.3. Excepcions del PowerPC 405

A continuació detallem en la Taula B.1 el conjunt d'excepcions que s'usen en el processador PowerPC 405.

**Taula B.1.** Excepcions suportades pel PPC405D5.

| **Excepció** | **Traducció** | ***Vector Offset*** | **Classificació** | **Causa** |
|---|---|---|---|---|
| Critical Input | Entrada Crítica | 0x0100 | Crítica | Senyal d'interrupció externa crítica |
| Machine Check | Comprovació de Màquina | 0x0200 | Crítica | Error de bus extern |
| Data Storage | Emmagatze-mament de Dades | 0x0300 | No crítica | Violació en l'accés a dades |
| Instruction Storage | Emmagatze-mament de | 0x0400 | No crítica | Violació en l'accés a instrucions |
| External | Externa | 0x0500 | No crítica | Senyal d'interrupció externa no crítica |
| Alignment | Alineació | 0x0600 | No crítica | Operand no alineat de *dcread, lwarx, stwcx*.<br>Ús de *dcbz* a memòria no cacheable o d'escriptura |
| Program | Programa | 0x0700 | No crítica | Execució d'instrucció ilegal o impròpia.<br>Execució d'instruccions de *trap*. |
| FPU Unavailable | FPU no disponible | 0x0800 | No crítica | Intent d'executar una instrucció FPU quan el FPU està deshabilitat. |
| System Call | Crida de Sistema | 0x0C00 | No crítica | Execució de la instrucció *sc*. |
| APU | APU no | 0x0F20 | No crítica | Intent d'executar una |

| | | | | |
|---|---|---|---|---|
| Unavailable | disponible | | | instrucció APU quan el APU està deshabilitat. |
| Programmable-Interval Timer | Temporitzador d'interval programables | 0x1000 | No crítica | Finalització del temporitzador d'interval programable. |
| Fixed-Interval Timer | Temporitzador d'interval fix | 0x1010 | No crítica | Finalització del temporitzador d'interval fix. |
| Watchdog Timer | Temporitzador de vigilància | 0x1020 | Crítica | Finalització del temporitzador de vigilància. |
| Data TLB Miss | Pèrdua de TLB de dades | 0x1100 | No crítica | No s'ha trobat la traducció dades-paginació. |
| Instrucció TLB Miss | Pèrdua de TLB d'instruccions | 0x1200 | No crítica | No s'ha trobat la traducció instruccions-paginació. |
| Debug | Depuració | 0x2000 | Crítica | Ocurrència d'un esdeveniment de depuració. |

### B.1.4. Excepcions simultànies i prioritat de les Interrupcions

El mecanisme d'interrupcions del PPC405 respon a les excepcions en sèrie. Si múltiples excepcions estan pendents al mateix temps, les interrupcions associades succeeixen en un ordre consistent i predible. Així i tot, les excepcions crítiques i no crítiques utilitzen parells de registres diferents, i les ocurrències simultànies d'aquestes excepcions també es processen en sèrie.

El PPC405 utilitza les prioritzacions d'interrupcions que es mostra en la Taula B.2 per a manegar les excepcions que succeeixen de forma simultània. Les interrupcions de baixa prioritat succeeixen després de les interrupcions d'alta prioritat.

**Taula B.2.** Prioritat d'interrupcions per a excepcions simultànies.

| Prioritat | Excepció | Causa |
|---|---|---|
| 1 | Machine check – Data | Error de bus extern durant l'accés a dades. |
| 2 | Debug – Instruction-address compare | Esdeveniment de depuració en la comparació d'adreces d'instrucció. |
| 3 | Machine check – Instruccion | Intent d'execució d'una instrucció per la qual un error extern de bu ha succeït durant la recollida d'instrucció. |
| 4 | Debug – Exception | Esdeveniment de depuració d'excepcions (EDE). |
| | Debug – Unconditional | Esdeveniment de depuració incondicional (UDE). |

| | | |
|---|---|---|
| 5 | Critical input | Es confirma el senyal d'entrada d'una interrupció crítica. |
| 6 | Watchdog timer | Timeout del temporitzador de vigilància. |
| 7 | Instruction TLB Miss | Intent d'executar una instrucció des d'una adreça de memòria amb una translació de pàgina no vàlida, carregada en el TLB (només en mode virtual). |
| 8 | Instruction storage – No access | En mode usuari, intent d'execució d'una instrucció des d'una adreça de memòria sense zona de protecció per accesso no permesos (només mode virtual). |
| 9 | Instruction storage – Non-executable | Intent d'execució d'una instrucció des d'una adreça de memòria no executable (només mode virtual). |
| | Instruction storage – Guarded | Intent d'execució d'una instrucció des d'una adreça de memòria vigilada. |
| 10 | Program | Intent d'execució de:<br>o Una instrucció ilegal.<br>o Instruccions de coma flotant no implementades.<br>o Instruccions del processador auxiliar no implementades.<br>o Una instrucció privilegiada des del mode usuari.<br>o Execució d'una instrucció *trap* que satisfà les condicions de *trap*. |
| | System call | Execució de la instrucció **sc**. |
| | FPU unavailable | Intent d'execució d'una instrucció en coma flotant quan el MSR[FP] = 0. No implementat en el PPC405D5 |
| | APU unavailable | Intent d'execució d'una instrucció en coma flotant quan el MSR[AP] = 0. No implementat en el PPC405D5 |
| 11 | Data TLB Miss | Intent d'accés a dades des d'una adreça amb una traducció de pàgina no vàlida carregada en el TLB (només mode virtual). |
| 12 | Data storage – No access | En mode usuari, intent d'accés a dades des d'una adreça de memòria amb zona protegida d'accés no permès (només mode virtual). |
| 13 | Data storage – Read-only | Intent d'escriptura de dades en una adreça de memòria de només lectura (només mode virtual). |
| | Data storage – User defined | Intent d'escriptura de dades en una adreça de memòria amb l'atribut d'emmagatzematge U0 configurat a "1", quan les excepcions U0 estan habilitades. |
| 14 | Alignment | Intent d'execució de:<br>o **Dcbz** a una adreça no cacheable o d'escriptura cacheable.<br>o **lwarx** o **stwcx** a una adreça no alineada a paraula.<br>o **dcread** a una adreça que no està alineada a paraula (només mode privilegiat). |
| 15 | Debug – branch taken | Esdeveniment de depuració de presa de bifurcació (BT). |
| | Debun – Data- | Esdeveniment de depuració de comparació |

| | | |
|---|---|---|
| | address compare | d'adreces de dades (DAC). |
| | Debug – Data-value compare | Esdeveniment de depuració de comparació de valors de dades (DAC). |
| | Debug – Instructions completion | Esdeveniment de depuració de finalització d'instrucció (IC). |
| | Debug – Trap instruction | Esdeveniment de depuració d'instrucció de *trap* (TDE). |
| 16 | External | Es confirma el senyal d'entrada d'una interrupció no crítica. |
| 17 | Fixed-interval timer | Finalització del temporitzador d'interval Fix. |
| 18 | Programmable-interval timer | Finalització del temporitzador d'interval Programable. |

### B.1.5. Instruccions executades parcialment

Algunes instruccions poden provocar una excepció d'alineament o d'emmagatzemament de dades durant la seva execució. Quan una interrupció succeeix, algun estat visible pel software es pot actualitzar per a reflexar l'execució parcial de la instrucció amb excepcions. Les instruccions i l'efecte que les interrupcions tenen en l'execució parcial són:

- Instruccions de càrrega múltiple i càrrega de *strings*. És possible que alguns dels registres estiguin actualitzats quan l'excepció d'emmagatzemament de dades o d'alineament succeeix. Quan la instrucció es reinicia, els registres modificats són actualitzats un altre cop.

- Instruccions d'emmagatzemament múltiple o emmagatzemament de *strings*. És possible que alguns dels bytes en memòria estiguin actualitzats quan l'excepció d'emmagatzemament de dades o alineament succeeix. Quan les instruccions es reinicien, les localitzacions de memòria modificades s'actualitzen un altre cop.

- Instruccions de càrrega escalar que cobreixen una condició de paraula. És possible que alguns bytes de memòria hagin estat accedits (llegits) quan l'excepció d'emmagatzemament de dades o d'alineament succeeix. No obstant, els registres estan actualitzats.

- Instruccions d'emmagatzemament escalar que cobreixen condicions de paraula. És possible que alguns bytes en memòria estiguin actualitzats quan l'excepció d'emmagatzemament de dades o alineament succeeix. Si la instrucció és una forma d'actualització, el registre d'actualització no està actualitzat. Quan la instrucció es reinicia, la memòria modificada s'actualitza un altre cop.

En els casos anteriors, l'execució parcial d'una instrucció no viola mai la protecció de memòria. Cap altre instrucció actualitza l'estat visible del software si una excepció succeeix com a part de l'execució.

Per prevenir les instruccions de càrrega i emmagatzematge de ser interrompudes i reiniciades, només les instruccions escalars haurien de ser utilitzades per a referenciar memòries. Igualment, alguna de les següents regles han de seguir-se per a evitar aquests problemes:

- L'operand de memòria ha d'estar alineat amb el límit de grandària dels operands.
- La localització d'accés a memòria ha d'estar protegit per l'atribut de prohibició d'emmagatzematge.

## B.2. Flux d'atenció a les interrupcions

El flux d'atenció a les interrupcions està compost per dos subfluxos, un per a carregar el *handler* d'interrupcions, i un altre per a retornar d'aquest al flux normal del programa.

### B.2.1. Transferència de control als Handlers d'interrupcions

La següent figura Fig. B.1 mostra quines són les accions executades pel processador quan una interrupció succeeix. El procediment és el que es descriu a continuació:

1. Guardar l'adreça d'interrupció de retorn (adreça efectiva).
   Generalment, l'adreça de retorn és tant de la instrucció que ha provocat la excepció, o de la instrucció següent que hauria d'haver-se executat si no s'hagués produït cap excepció. Es guarda en un o dos registres *save/restore*, depenent del tipus d'interrupció:
   - Les interrupcions crítiques carreguen el SRR2 amb l'adreça de retorn.
   - Les interrupcions no crítiques carreguen el SRR0 amb l'adreça de retorn.

2. Guardar l'estat del programa interromput.
   El contingut del registre de la màquina d'estats (MSR) es copien en un o dos registres *save/restore*, depenent del tipus d'interrupció:
   - Les interrupcions crítiques carreguen el SRR3 amb una còpia del MSR.
   - Les interrupcions no crítiques carreguen el SRR1 amb una còpia del MSR.

3. Actualitzar el registre d'excepcions síndrome, *exception-syndrome register* (ESR), si aplica.
   Cinc excepcions reporten informació d'estat en el ESR quan el control és transferit al *handler* d'interrupcions (l'ESR no es modifica per les excepcions restants):
   - Comprovació de Màquina.
   - Emmagatzemament de Dades.
   - Emmagatzemament d'Instruccions.
   - Programa.

- Pèrdua de TLB de Dades.

Els *handlers* d'interrupció utilitzen l'ESR per a determinar la causa de l'excepció.

4. Actualitzar el registre d'adreces d'excepcions (DEAR), si aplica.
   Tres excepcions reporten l'adreça d'un accés fallit a dades en el DEAR quan el control és transferit al *handler* d'interrupcions (el DEAR no es modifica per les excepcions restants):
   - Emmagatzemament de Dades.
   - Alineament.
   - Pèrdua de TLB de Dades.

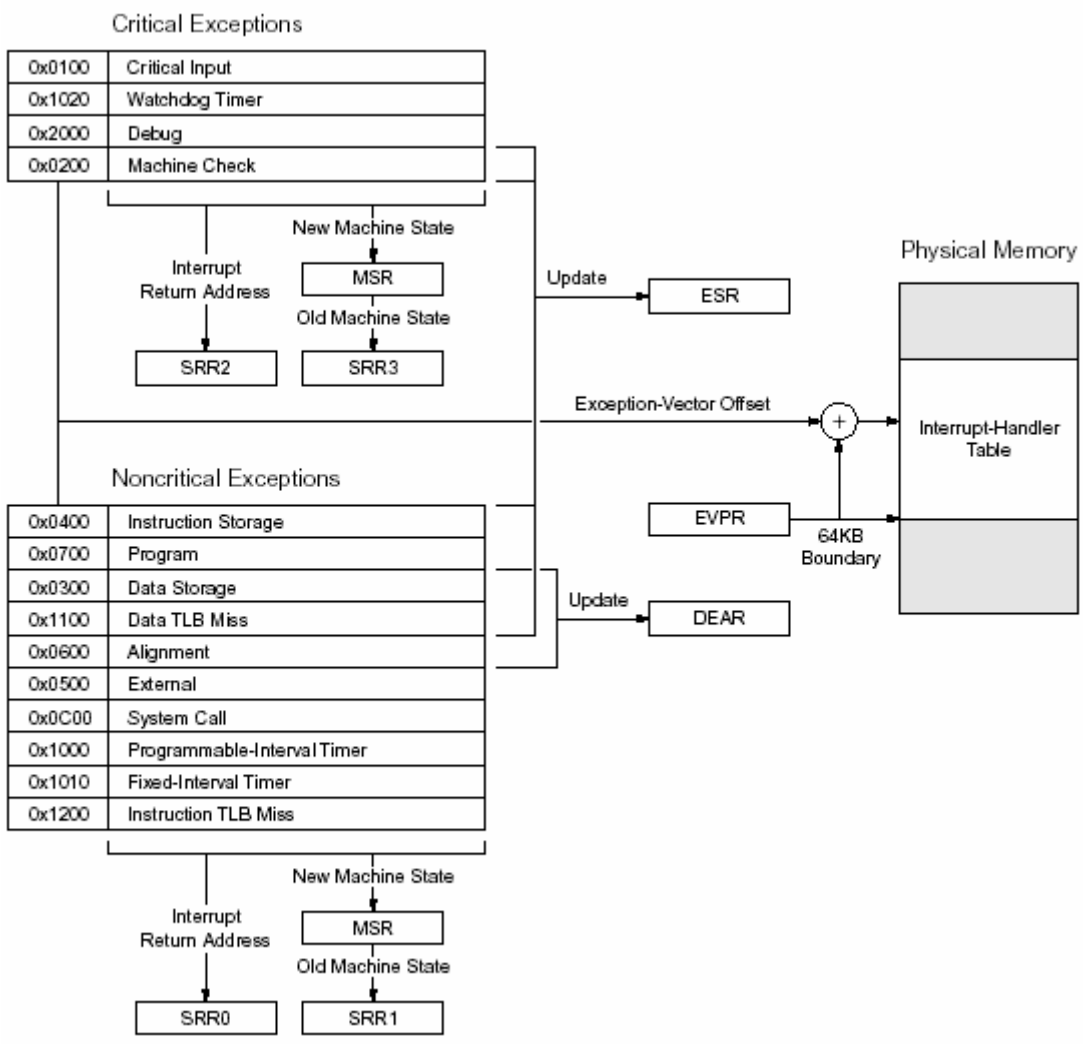



**Fig. B.1.** Mecanisme d'excepcions del PPC405.

5. Carregar el nou estat de programa en el MSR.
   Totes les interrupcions carreguen el nou estat de programa en un MSR. El nou estat posiciona el processador en mode privilegiat. La translació de d'adreces d'instruccions i adreces de dades es deshabiliten,

posicionant el processador en mode real. Algunes interrupcions es deshabiliten, dependent de l'excepció.

6. Sincronitzar el context del processador.
   Totes les interrupcions estan sincronitzades amb el context. El processador treu i executa la primera instrucció en el *handler* d'interrupcions en el context establert pel nou contingut del MSR.

7. Transferir el control al *handler* d'interrupcions.
   S'associa amb cada excepció un *offset* del vector d'excepcions. L'*offset* s'afegeix a una adreça base alineada a 64KB localitzada en el registre prefix del vector d'excepcions (EVPR). La suma representa una adreça física que apunta a la primera instrucció del *handler* d'interrupcions. Els *handlers* d'interrupcions estan localitzats en una taula de *handlers* d'interrupcions.

### B.2.2. Retorn dels Handlers d'interrupcions

El software de sistema surt d'un *handler* d'interrupcions utilitzant una de les dues instruccions. Els *handlers* d'interrupcions no crítiques retornen a un programa interromput utilitzant la instrucció *return-from-interrupt* (rfi). Els *handlers* d'interrupcions crítiques retornen al programa interromput utilitzant la instrucció *retorn-from-critical-interrupt* (rfci). Ambdues instruccions operen de forma similar, amb la única diferència del parell de registres *save/*restore utilitzats per a restaurar l'estat del programa interromput. Les instruccions **rfi** i **rfci** realitzen les següents funcions:

1. Totes les instruccions prèvies completen l'execució en el context en què varen ser emeses (privilegiat, protecció, i mode de translació d'adreces).

2. Totes les instruccions prèvies es completen en el punt en el qual no poden provocar una excepció.

3. El processador carrega el MSR amb l'estat del programa interromput des d'un dels dos registres *save/restore*, depenent de la instrucció:
   - **rfi** copia SRR1 en el MSR.
   - **rfci** copia SRR3 en el MSR.

4. El context de processador es sincronitza.
   Ambdues instruccions es sincronitzen contextualment. El processador pren i executa la instrucció en l'adreça de retorn en el context del programa interromput.

5. El processador comença la recollida i execució d'instruccions des del programa interromput:
   - Les instruccions es recullen des de l'adreça en SRR0 un cop completada la instrucció **rfi**.
   - Les instruccions es recullen des de l'adreça SRR2 un completada la instrucció **rfci**.

# ANNEX C. FULL D'ESPECIFICACIONS DEL VIRTEX-II PRO DE XILINX

XILINX®

## Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Introduction and Overview

DS083-1 (v4.5) October 10, 2005 **Product Specification**

### Summary of Virtex-II Pro™ / Virtex-II Pro X Features

- High-Performance Platform FPGA Solution, Including
  - Up to twenty RocketIO™ or RocketIO X embedded Multi-Gigabit Transceivers (MGTs)
  - Up to two IBM PowerPC™ RISC processor blocks
- Based on Virtex-II™ Platform FPGA Technology
  - Flexible logic resources
  - SRAM-based in-system configuration
  - Active Interconnect technology
  - SelectRAM™+ memory hierarchy
  - Dedicated 18-bit x 18-bit multiplier blocks
  - High-performance clock management circuitry
  - SelectI/O™-Ultra technology
  - XCITE Digitally Controlled Impedance (DCI) I/O

Virtex-II Pro / Virtex-II Pro X family members and resources are shown in Table 1.

*Table 1:* **Virtex-II Pro / Virtex-II Pro X FPGA Family Members**

| Device[1] | RocketIO Transceiver Blocks | PowerPC Processor Blocks | Logic Cells[2] | CLB (1 = 4 slices = max 128 bits) | | 18 X 18 Bit Multiplier Blocks | Block SelectRAM+ | | DCMs | Maximum User I/O Pads |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Slices | Max Distr RAM (Kb) | | 18 Kb Blocks | Max Block RAM (Kb) | | |
| XC2VP2 | 4 | 0 | 3,168 | 1,408 | 44 | 12 | 12 | 216 | 4 | 204 |
| XC2VP4 | 4 | 1 | 6,768 | 3,008 | 94 | 28 | 28 | 504 | 4 | 348 |
| XC2VP7 | 8 | 1 | 11,088 | 4,928 | 154 | 44 | 44 | 792 | 4 | 396 |
| XC2VP20 | 8 | 2 | 20,880 | 9,280 | 290 | 88 | 88 | 1,584 | 8 | 564 |
| XC2VPX20 | 8[4] | 1 | 22,032 | 9,792 | 306 | 88 | 88 | 1,584 | 8 | 552 |
| XC2VP30 | 8 | 2 | 30,816 | 13,696 | 428 | 136 | 136 | 2,448 | 8 | 644 |
| XC2VP40 | 0[3], 8, or 12 | 2 | 43,632 | 19,392 | 606 | 192 | 192 | 3,456 | 8 | 804 |
| XC2VP50 | 0[3] or 16 | 2 | 53,136 | 23,616 | 738 | 232 | 232 | 4,176 | 8 | 852 |
| XC2VP70 | 16 or 20 | 2 | 74,448 | 33,088 | 1,034 | 328 | 328 | 5,904 | 8 | 996 |
| XC2VPX70 | 20[4] | 2 | 74,448 | 33,088 | 1,034 | 308 | 308 | 5,544 | 8 | 992 |
| XC2VP100 | 0[3] or 20 | 2 | 99,216 | 44,096 | 1,378 | 444 | 444 | 7,992 | 12 | 1,164 |

**Notes:**
1. -7 speed grade devices are not available in Industrial grade.
2. Logic Cell ≈ (1) 4-input LUT + (1)FF + Carry Logic
3. These devices can be ordered in a configuration without RocketIO transceivers. See Table 3 for package configurations.
4. Virtex-II Pro X devices equipped with RocketIO X transceiver cores.

### RocketIO X Transceiver Features (XC2VPX20 and XC2VPX70 Only)

- Variable-Speed Full-Duplex Transceiver (XC2VPX20) Allowing 2.488 Gb/s to 6.25 Gb/s Baud Transfer Rates.
  - Includes specific baud rates used by various standards, as listed in Table 4, Module 2.
- Fixed-Speed Full-Duplex Tranceiver (XC2VPX70) Operating at 4.25 Gb/s Baud Transfer Rate.
- Eight or Twenty Transceiver Modules on an FPGA, Depending upon Device
- Monolithic Clock Synthesis and Clock Recovery
  - Eliminates the need for external components
- Automatic Lock-to-Reference Function
- Programmable Serial Output Differential Swing
  - 200 mV to 1600 mV, peak-peak
  - Allows compatibility with other serial system voltage levels
- Programmable Pre-emphasis Levels 0 to 500%
- Telecom/Datacom Support Modes
  - "x8" and "x10" clocking/data paths
  - 64B/66B clocking support

- Programmable Receiver Equalization
- Internal AC Coupling
- On-Chip 50Ω Termination
  - Eliminates the need for external termination resistors
- Pre- and Post-Driver Serial and Parallel TX-to-RX Internal Loopback Modes for Testing Operability
- Programmable Comma Detection
  - Allows for any protocol
  - Allows for detection of any 10-bit character
- 8B/10B and 64B/66B Encoding Blocks

## RocketIO Transceiver Features (All Except XC2VPX20 and XC2VPX70)

- Full-Duplex Serial Transceiver (SERDES) Capable of Baud Rates from 600 Mb/s to 3.125 Gb/s
- 100 Gb/s Duplex Data Rate (20 Channels)
- Monolithic Clock Synthesis and Clock Recovery (CDR)
- Fibre Channel, 10G Fibre Channel, Gigabit Ethernet, 10 Gb Attachment Unit Interface (XAUI), and Infiniband-Compliant Transceivers
- 8-, 16-, or 32-bit Selectable Internal FPGA Interface
- 8B/10B Encoder and Decoder (optional)
- 50Ω/75Ω on-chip Selectable Transmit and Receive Terminations
- Programmable Comma Detection
- Channel Bonding Support (from 2 to 20 Channels)
- Rate Matching via Insertion/Deletion Characters
- Four Levels of Selectable Pre-Emphasis
- Five Levels of Output Differential Voltage
- Per-Channel Internal Loopback Modes
- 2.5V Transceiver Supply Voltage

## PowerPC RISC Processor Block Features (All Except XC2VP2)

- Embedded 300+ MHz Harvard Architecture Block
- Low Power Consumption: 0.9 mW/MHz
- Five-Stage Data Path Pipeline
- Hardware Multiply/Divide Unit
- Thirty-Two 32-bit General Purpose Registers
- 16 KB Two-Way Set-Associative Instruction Cache
- 16 KB Two-Way Set-Associative Data Cache
- Memory Management Unit (MMU)
  - 64-entry unified Translation Look-aside Buffers (TLB)
  - Variable page sizes (1 KB to 16 MB)
- Dedicated On-Chip Memory (OCM) Interface
- Supports IBM CoreConnect™ Bus Architecture
- Debug and Trace Support
- Timer Facilities

## Virtex-II Pro Platform FPGA Technology (All Devices)

- SelectRAM+ Memory Hierarchy
  - Up to 8 Mb of True Dual-Port RAM in 18 Kb block SelectRAM+ resources
  - Up to 1,378 Kb of distributed SelectRAM+ resources
  - High-performance interfaces to external memory
- Arithmetic Functions
  - Dedicated 18-bit x 18-bit multiplier blocks
  - Fast look-ahead carry logic chains
- Flexible Logic Resources
  - Up to 88,192 internal registers/latches with Clock Enable
  - Up to 88,192 look-up tables (LUTs) or cascadable variable (1 to 16 bits) shift registers
  - Wide multiplexers and wide-input function support
  - Horizontal cascade chain and Sum-of-Products support
  - Internal 3-state busing
- High-Performance Clock Management Circuitry
  - Up to twelve Digital Clock Manager (DCM) modules
    - Precise clock de-skew
    - Flexible frequency synthesis
    - High-resolution phase shifting
  - 16 global clock multiplexer buffers in all parts
- Active Interconnect Technology
  - Fourth-generation segmented routing structure
  - Fast, predictable routing delay, independent of fanout
  - Deep sub-micron noise immunity benefits
- SelectIO™-Ultra Technology
  - Up to 1,164 user I/Os
  - Twenty-two single-ended standards and ten differential standards
  - Programmable LVCMOS sink/source current (2 mA to 24 mA) per I/O
  - XCITE Digitally Controlled Impedance (DCI) I/O
  - PCI/PCI-X support (1)
  - Differential signaling
    - 840 Mb/s Low-Voltage Differential Signaling I/O (LVDS) with current mode drivers
    - On-chip differential termination
    - Bus LVDS I/O

1. Refer to **XAPP653** for more information.

        - HyperTransport (LDT) I/O with current driver buffers
        - Built-in DDR input and output registers
    - Proprietary high-performance SelectLink technology for communications between Xilinx devices
        - High-bandwidth data path
        - Double Data Rate (DDR) link
        - Web-based HDL generation methodology
- SRAM-Based In-System Configuration
    - Fast SelectMAP™ configuration
    - Triple Data Encryption Standard (DES) security option (bitstream encryption)
    - IEEE 1532 support
    - Partial reconfiguration
    - Unlimited reprogrammability
    - Readback capability
- Supported by Xilinx Foundation™ and Alliance Series™ Development Systems
    - Integrated VHDL and Verilog design flows
    - ChipScope™ Integrated Logic Analyzer
- 0.13 µm Nine-Layer Copper Process with 90 nm High-Speed Transistors
- 1.5V ($V_{CCINT}$) core power supply, dedicated 2.5V $V_{CCAUX}$ auxiliary and $V_{CCO}$ I/O power supplies
- IEEE 1149.1 Compatible Boundary-Scan Logic Support
- Flip-Chip and Wire-Bond Ball Grid Array (BGA) Packages in Standard 1.00 mm Pitch.
- Wire-Bond BGA Devices Available in Pb-Free Packaging (**www.xilinx.com/pbfree)**
- Each Device 100% Factory Tested

## General Description

The Virtex-II Pro and Virtex-II Pro X families contain platform FPGAs for designs that are based on IP cores and customized modules. The family incorporates multi-gigabit transceivers and PowerPC CPU blocks in Virtex-II Pro Series FPGA architecture. It empowers complete solutions for telecommunication, wireless, networking, video, and DSP applications.

The leading-edge 0.13 µm CMOS nine-layer copper process and Virtex-II Pro architecture are optimized for high performance designs in a wide range of densities. Combining a wide variety of flexible features and IP cores, the Virtex-II Pro family enhances programmable logic design capabilities and is a powerful alternative to mask-programmed gate arrays.

## Architecture

### Array Overview

Virtex-II Pro and Virtex-II Pro X devices are user-programmable gate arrays with various configurable elements and embedded blocks optimized for high-density and high-performance system designs. Virtex-II Pro devices implement the following functionality:

- Embedded high-speed serial transceivers enable data bit rate up to 3.125 Gb/s per channel (RocketIO) or 6.25 Gb/s (RocketIO X).
- Embedded IBM PowerPC 405 RISC processor blocks provide performance up to 400 MHz.
- SelectIO-Ultra blocks provide the interface between package pins and the internal configurable logic. Most popular and leading-edge I/O standards are supported by the programmable IOBs.
- Configurable Logic Blocks (CLBs) provide functional elements for combinatorial and synchronous logic, including basic storage elements. BUFTs (3-state buffers) associated with each CLB element drive dedicated segmentable horizontal routing resources.
- Block SelectRAM+ memory modules provide large 18 Kb storage elements of True Dual-Port RAM.
- Embedded multiplier blocks are 18-bit x 18-bit dedicated multipliers.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division, and coarse- and fine-grained clock phase shifting.

A new generation of programmable routing resources called Active Interconnect Technology interconnects all these elements. The general routing matrix (GRM) is an array of routing switches. Each programmable element is tied to a switch matrix, allowing multiple connections to the general routing matrix. The overall programmable interconnection is hierarchical and supports high-speed designs.

All programmable elements, including the routing resources, are controlled by values stored in static memory cells. These values are loaded in the memory cells during configuration and can be reloaded to change the functions of the programmable elements.

### Features

This section briefly describes Virtex-II Pro / Virtex-II Pro X features. For more details, refer to Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Functional Description.

#### *RocketIO / RocketIO X MGT Cores*

The RocketIO and RocketIO X Multi-Gigabit Transceivers are flexible parallel-to-serial and serial-to-parallel embedded transceiver cores used for high-bandwidth interconnection between buses, backplanes, or other subsystems.

Multiple user instantiations in an FPGA are possible, providing up to 100 Gb/s (RocketIO) or 170 Gb/s (RocketIO X) of full-duplex raw data transfer. Each channel can be operated at a maximum data transfer rate of 3.125 Gb/s (RocketIO) or 6.25 Gb/s (RocketIO X).

Each RocketIO or RocketIO X core implements the following technology:

- Serializer and deserializer (SERDES)
- Monolithic clock synthesis and clock recovery (CDR)
- 10 Gigabit Attachment Unit Interface (XAUI) Fibre Channel (3.1875 Gb/s XAUI), Infiniband, PCI Express, Aurora, SXI-5 (SFI-5,/SPI-5), and OC-48 compatibility(1)
- 8/16/32-bit (RocketIO) or 8/16/32/64-bit (RocketIO X) selectable FPGA interface
- 8B/10B (RocketIO) or 8B/10B and 64B/66B (RocketIO X) encoder and decoder with bypassing option on each channel
- Channel bonding support (two to twenty channels)
  - Elastic buffers for inter-chip deskewing and channel-to-channel alignment
- Receiver clock recovery tolerance of up to 75 non-transitioning bits
- 50Ω (RocketIO X) or 50Ω /75Ω selectable (RocketIO) on-chip transmit and receive terminations
- Programmable comma detection and word alignment
- Rate matching via insertion/deletion characters
- Automatic lock-to-reference function
- Programmable pre-emphasis support
- Per-channel serial and parallel transmitter-to-receiver internal loopback modes
- Optional transmit and receive data inversion
- Cyclic Redundancy Check support (RocketIO only)

### *PowerPC 405 Processor Block*

The PPC405 RISC CPU can execute instructions at a sustained rate of one instruction per cycle. On-chip instruction and data cache reduce design complexity and improve system throughput.

The PPC405 features include:

- PowerPC RISC CPU
  - Implements the PowerPC User Instruction Set Architecture (UISA) and extensions for embedded applications
  - Thirty-two 32-bit general purpose registers (GPRs)
  - Static branch prediction
  - Five-stage pipeline with single-cycle execution of most instructions, including loads/stores
  - Unaligned and aligned load/store support to cache, main memory, and on-chip memory
  - Hardware multiply/divide for faster integer arithmetic (4-cycle multiply, 35-cycle divide)
  - Enhanced string and multiple-word handling
  - Big/little endian operation support
- Storage Control
  - Separate instruction and data cache units, both two-way set-associative and non-blocking
  - Eight words (32 bytes) per cache line
  - 16 KB array Instruction Cache Unit (ICU), 16 KB array Data Cache Unit (DCU)
  - Operand forwarding during instruction cache line fill
  - Copy-back or write-through DCU strategy
  - Doubleword instruction fetch from cache improves branch latency
- Virtual mode memory management unit (MMU)
  - Translation of the 4 GB logical address space into physical addresses
  - Software control of page replacement strategy
  - Supports multiple simultaneous page sizes ranging from 1 KB to 16 MB
- OCM controllers provide dedicated interfaces between Block SelectRAM+ memory and processor block instruction and data paths for high-speed access
- PowerPC timer facilities
  - 64-bit time base
  - Programmable interval timer (PIT)
  - Fixed interval timer (FIT)
  - Watchdog timer (WDT)
- Debug Support
  - Internal debug mode
  - External debug mode
  - Debug Wait mode
  - Real Time Trace debug mode
  - Enhanced debug support with logical operators
  - Instruction trace and trace-back support
  - Forward or backward trace
- Two hardware interrupt levels support
- Advanced power management support

### *Input/Output Blocks (IOBs)*

IOBs are programmable and can be categorized as follows:

- Input block with an optional single data rate (SDR) or double data rate (DDR) register
- Output block with an optional SDR or DDR register and an optional 3-state buffer to be driven directly or through an SDR or DDR register
- Bidirectional block (any combination of input and output configurations)

These registers are either edge-triggered D-type flip-flops or level-sensitive latches.

IOBs support the following single-ended I/O standards:

- LVTTL, LVCMOS (3.3V,(2) 2.5V, 1.8V, and 1.5V)
- PCI-X compatible (133 MHz and 66 MHz) at 3.3V(3)
- PCI compliant (66 MHz and 33 MHz) at 3.3V(3)
- GTL and GTLP

1. Refer to Table 4, Module 2 for detailed information about RocketIO and RocketIO X transceiver compatible protocols.
2. Refer to **XAPP659** for more information.
3. Refer to **XAPP653** for more information.

- HSTL (1.5V and 1.8V, Class I, II, III, and IV)
- SSTL (1.8V and 2.5V, Class I and II)

The DCI I/O feature automatically provides on-chip termination for each single-ended I/O standard.

The IOB elements also support the following differential signaling I/O standards:

- LVDS and Extended LVDS (2.5V)
- BLVDS (Bus LVDS)
- ULVDS
- LDT
- LVPECL (2.5V)

Two adjacent pads are used for each differential pair. Two or four IOBs connect to one switch matrix to access the routing resources. On-chip differential termination is available for LVDS, LVDS Extended, ULVDS, and LDT standards.

### *Configurable Logic Blocks (CLBs)*

CLB resources include four slices and two 3-state buffers. Each slice is equivalent and contains:

- Two function generators (F & G)
- Two storage elements
- Arithmetic logic gates
- Large multiplexers
- Wide function capability
- Fast carry look-ahead chain
- Horizontal cascade chain (OR gate)

The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16-bit shift registers, or as 16-bit distributed SelectRAM+ memory.

In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches.

Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources.

### *Block SelectRAM+ Memory*

The block SelectRAM+ memory resources are 18 Kb of True Dual-Port RAM, programmable from 16K x 1 bit to 512 x 36 bit, in various depth and width configurations. Each port is totally synchronous and independent, offering three "read-during-write" modes. Block SelectRAM+ memory is cascadable to implement large embedded storage blocks. Supported memory configurations for dual-port and single-port modes are shown in Table 2.

*Table 2:* **Dual-Port and Single-Port Configurations**

| | | |
|---|---|---|
| 16K x 1 bit | 4K x 4 bits | 1K x 18 bits |
| 8K x 2 bits | 2K x 9 bits | 512 x 36 bits |

### *18 X 18 Bit Multipliers*

A multiplier block is associated with each SelectRAM+ memory block. The multiplier block is a dedicated 18 x 18-bit 2s complement signed multiplier, and is optimized for operations based on the block SelectRAM+ content on one port. The 18 x 18 multiplier can be used independently of the block SelectRAM+ resource. Read/multiply/accumulate operations and DSP filter structures are extremely efficient.

Both the SelectRAM+ memory and the multiplier resource are connected to four switch matrices to access the general routing resources.

### *Global Clocking*

The DCM and global clock multiplexer buffers provide a complete solution for designing high-speed clock schemes.

Up to twelve DCM blocks are available. To generate deskewed internal or external clocks, each DCM can be used to eliminate clock distribution delay. The DCM also provides 90-, 180-, and 270-degree phase-shifted versions of its output clocks. Fine-grained phase shifting offers high-resolution phase adjustments in increments of $^{1}/_{256}$ of the clock period. Very flexible frequency synthesis provides a clock output frequency equal to a fractional or integer multiple of the input clock frequency. For exact timing parameters, see Virtex-II Pro and Virtex-II Pro X Platform FPGAs: DC and Switching Characteristics.

Virtex-II Pro devices have 16 global clock MUX buffers, with up to eight clock nets per quadrant. Each clock MUX buffer can select one of the two clock inputs and switch glitch-free from one clock to the other. Each DCM can send up to four of its clock outputs to global clock buffers on the same edge. Any global clock pin can drive any DCM on the same edge.

### *Routing Resources*

The IOB, CLB, block SelectRAM+, multiplier, and DCM elements all use the same interconnect scheme and the same access to the global routing matrix. Timing models are shared, greatly improving the predictability of the performance of high-speed designs.

There are a total of 16 global clock lines, with eight available per quadrant. In addition, 24 vertical and horizontal long lines per row or column, as well as massive secondary and local routing resources, provide fast interconnect. Virtex-II Pro buffered interconnects are relatively unaffected by net fanout, and the interconnect layout is designed to minimize crosstalk.

Horizontal and vertical routing resources for each row or column include:

- 24 long lines
- 120 hex lines
- 40 double lines
- 16 direct connect lines (total in all four directions)

### *Boundary Scan*

Boundary-scan instructions and associated data registers support a standard methodology for accessing and configuring Virtex-II Pro devices, complying with IEEE standards

1149.1 and 1532. A system mode and a test mode are implemented. In system mode, a Virtex-II Pro device will continue to function while executing non-test boundary-scan instructions. In test mode, boundary-scan test instructions control the I/O pins for testing purposes. The Virtex-II Pro Test Access Port (TAP) supports BYPASS, PRELOAD, SAMPLE, IDCODE, and USERCODE non-test instructions. The EXTEST, INTEST, and HIGHZ test instructions are also supported.

### *Configuration*

Virtex-II Pro / Virtex-II Pro devices are configured by loading the bitstream into internal configuration memory using one of the following modes:

- Slave-serial mode
- Master-serial mode
- Slave SelectMAP mode
- Master SelectMAP mode
- Boundary-Scan mode (IEEE 1532)

A Data Encryption Standard (DES) decryptor is available on-chip to secure the bitstreams. One or two triple-DES key sets can be used to optionally encrypt the configuration data.

The Xilinx System Advanced Configuration Enviornment (System ACE) family offers high-capacity and flexible solution for FPGA configuration as well as program/data storage for the processor. See **DS080,** *System ACE CompactFlash Solution* for more information.

### *Readback and Integrated Logic Analyzer*

Configuration data stored in Virtex-II Pro / Virtex-II Pro configuration memory can be read back for verification. Along with the configuration data, the contents of all flip-flops and latches, distributed SelectRAM+, and block SelectRAM+ memory resources can be read back. This capability is useful for real-time debugging.

The Xilinx ChipScope Integrated Logic Analyzer (ILA) cores and Integrated Bus Analyzer (IBA) cores, along with the ChipScope Pro Analyzer software, provide a complete solution for accessing and verifying user designs within Virtex-II Pro devices.

## IP Core and Reference Support

Intellectual Property is part of the Platform FPGA solution. In addition to the existing FPGA fabric cores, the list below shows some of the currently available hardware and software intellectual properties specially developed for Virtex-II Pro / Virtex-II Pro X by Xilinx. Each IP core is modular, portable, Real-Time Operating System (RTOS) independent, and CoreConnect compatible for ease of design migration. Refer to **www.xilinx.com/ipcenter** for the latest and most complete list of cores.

### Hardware Cores

- Bus Infrastructure cores (arbiters, bridges, and more)
- Memory cores (DDR, Flash, and more)
- Peripheral cores (UART, IIC, and more)
- Networking cores (ATM, Ethernet, and more)

### Software Cores

- Boot code
- Test code
- Device drivers
- Protocol stacks
- RTOS integration
- Customized board support package

## Virtex-II Pro / Virtex-II Pro X Device/Package Combinations and Maximum I/Os

Offerings include ball grid array (BGA) packages with 1.0 mm pitch. In addition to traditional wire-bond interconnect (FG/FGG packages), flip-chip interconnect (FF packages) is used in some of the BGA offerings. Flip-chip interconnect construction supports more I/Os than are possible in wire-bond versions of similar packages, providing a high pin count and excellent power dissipation.

The device/package combination table (Table 3) details the maximum number of user I/Os and RocketIO / RocketIO X MGTs for each device and package using wire-bond or flip-chip technology.

The FF1148 and FF1696 packages have no RocketIO transceivers bonded out. Extra SelectIO-Ultra resources occupy available pins in these packages, resulting in a higher user I/O count. These packages are available for the XC2VP40, XC2VP50, and XC2VP100 devices only.

The I/Os per package count includes all user I/Os except the 15 control pins (CCLK, DONE, M0, M1, M2, PROG_B, PWRDWN_B, TCK, TDI, TDO, TMS, HSWAP_EN, DXN, DXP, and RSVD), VBATT, and the RocketIO / RocketIO X transceiver pins.

*Table 3:* **Virtex-II Pro Device/Package Combinations and Maximum Number of Available I/Os**

| Package(1) | FG256/ FGG256 | FG456/ FGG456 | FG676 | FF672 | FF896 | FF1152 | FF1148 | FF1517 | FF1704 | FF1696 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Pitch (mm)** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| **Size (mm)** | **17 x 17** | **23 x 23** | **26 x 26** | **27 x 27** | **31 x 31** | **35 x 35** | **35 x 35** | **40 x 40** | **42.5 x 42.5** | **42.5 x 42.5** |
| XC2VP2 | 140/4 | 156/4 | | 204/4 | | | | | | |
| XC2VP4 | 140/4 | 248/4 | | 348/4 | | | | | | |
| XC2VP7 | | 248/8 | | 396/8 | 396/8 | | | | | |
| XC2VP20 | | | 404/8 | | 556/8 | 564/8 | | | | |
| XC2VPX20 | | | | | 552/8(2) | | | | | |
| XC2VP30 | | | 416/8 | | 556/8 | 644/8 | | | | |
| XC2VP40 | | | 416/8 | | | 692/12 | 804/0(3) | | | |
| XC2VP50 | | | | | | 692/16 | 812/0(3) | 852/16 | | |
| XC2VP70 | | | | | | | | 964/16 | 996/20 | |
| XC2VPX70 | | | | | | | | | 992/20(2) | |
| XC2VP100 | | | | | | | | | 1,040/20 | 1,164/0(3) |

**Notes:**
1. Wirebond packages FG256, FG456, and FG676 are also available in Pb-free versions FGG256, FGG456, and FGG676. See Virtex-II Pro Ordering Examples for details on how to order.
2. Virtex-II Pro X device is equipped with RocketIO X transceiver cores.
3. The RocketIO transceivers in devices in the FF1148 and FF1696 packages are not bonded out to the package pins.

## Maximum Performance

Maximum performance of the RocketIO / RocketIO X transceiver and the PowerPC processor block varies, depending on package style and speed grade. See Table 4 for details. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: DC and Switching Characteristics contains the rest of the FPGA fabric performance parameters.

*Table 4:* **Maximum RocketIO / RocketIO X Transceiver and Processor Block Performance**

| Device | Speed Grade | | | Units |
|---|---|---|---|---|
| | -7(1) | -6 | -5 | |
| RocketIO X Transceiver FlipChip (FF) | N/A | 6.25(3) | 4.25(3) | Gb/s |
| RocketIO Transceiver FlipChip (FF) | 3.125 | 3.125 | 2.0 | Gb/s |
| RocketIO Transceiver Wirebond (FG) | 2.5 | 2.5 | 2.0 | Gb/s |
| PowerPC Processor Block | 400(2) | 350(2) | 300 | MHz |

**Notes:**
1. -7 speed grade devices are not available in Industrial grade.
2. **IMPORTANT!** When CPMC405CLOCK runs at speeds greater than 350 MHz in -7 Commercial grade dual-processor devices, or greater than 300 MHz in -6 Industrial grade dual-processor devices, users must implement the technology presented in **XAPP755**, "PowerPC 405 Clock Macro for -7(C) and -6(I) Speed Grade Dual-Processor Devices." Refer to Table 1 to identify dual-processor devices.
3. XC2VPX70 is only available at fixed 4.25 Gb/s baud rate.

## Virtex-II Pro Ordering Examples

Virtex-II Pro ordering examples are shown in Figure 1 (flip-chip package) and Figure 2 (Pb-free wire-bond package).

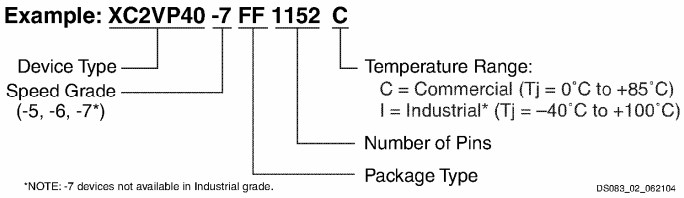



*Figure 1:* **Virtex-II Pro Ordering Example, Flip-Chip Package**

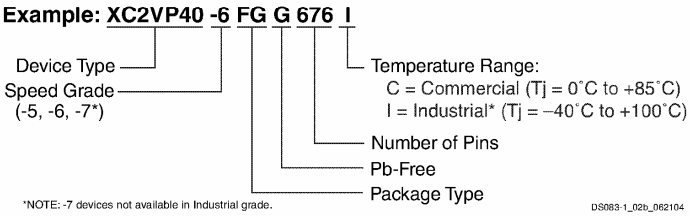



*Figure 2:* **Virtex-II Pro Ordering Example, Pb-Free Wire-Bond Package**

## Virtex-II Pro X Ordering Example

A Virtex-II Pro X ordering example is shown in Figure 3.

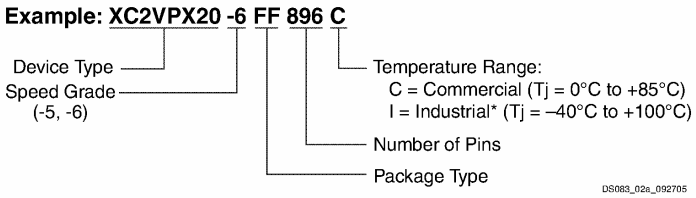



*Figure 3:* **Virtex-II Pro X Ordering Example, Flip-Chip Package**

## Revision History

This section records the change history for this module of the data sheet.

| Date | Version | Revision |
|---|---|---|
| 01/31/02 | 1.0 | Initial Xilinx release. |
| 06/13/02 | 2.0 | New Virtex-II Pro family members. New timing parameters per speedsfile **v1.62**. |
| 09/03/02 | 2.1 | Updates to Table 1 and Table 3. Processor Block information added to Table 4. |
| 09/27/02 | 2.2 | In Table 1, correct max number of XC2VP30 I/Os to 644. |
| 11/20/02 | 2.3 | Add bullet items for 3.3V I/O features. |
| 01/20/03 | 2.4 | • In Table 3, add FG676 package option for XC2VP20, XC2VP30, and XC2VP40.<br>• Remove FF1517 package option for XC2VP40. |
| 03/24/03 | 2.4.1 | • Correct number of single-ended I/O standards from 19 to 22.<br>• Correct minimum RocketIO serial speed from 622 Mbps to 600 Mbps. |
| 08/25/03 | 2.4.2 | • Add footnote referring to XAPP659 to callout for 3.3V I/O standards on page 4. |
| 12/10/03 | 3.0 | • XC2VP2 through XC2VP70 speed grades -5, -6, and -7, and XC2VP100 speed grades -5 and -6, are released to **Production status**. |
| 02/19/04 | 3.1 | • Table 1: Corrected number of RocketIO transceiver blocks for XC2VP40.<br>• Section Virtex-II Pro Platform FPGA Technology (All Devices): Updated number of differential standards supported from six to ten.<br>• Section Input/Output Blocks (IOBs): Added text stating that differential termination is available for LVDS, LVDS Extended, ULVDS, and LDT standards.<br>• Figure 1: Added note stating that -7 devices are not available in Industrial grade. |
| 03/09/04 | 3.1.1 | • Recompiled for backward compatibility with Acrobat 4 and above. No content changes. |
| 06/30/04 | 4.0 | Merged in DS110-1 (Module 1 of Virtex-II Pro X data sheet). Added information on available Pb-free packages. |
| 11/17/04 | 4.1 | *No changes in Module 1 for this revision.* |
| 03/01/05 | 4.2 | Table 3: Corrected number of RocketIO transceivers for XC2VP7-FG456. |
| 06/20/05 | 4.3 | *No changes in Module 1 for this revision.* |
| 09/15/05 | 4.4 | • Changed all instances of 10.3125 Gb/s (RocketIO transceiver maximum bit rate) to 6.25 Gb/s.<br>• Changed all instances of 412.5 Gb/s (RocketIO X transceiver maximum multi-channel raw data transfer rate) to 250 Gb/s. |
| 10/10/05 | 4.5 | • Changed XC2VPX70 variable baud rate specification to fixed-rate operation at 4.25 Gb/s.<br>• Changed maximum performance for -7 Virtex-II Pro X MGT (Table 4) to N/A. |

## Virtex-II Pro Data Sheet

The Virtex-II Pro Data Sheet contains the following modules:

- Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Introduction and Overview (Module 1)
- Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Functional Description (Module 2)
- Virtex-II Pro and Virtex-II Pro X Platform FPGAs: DC and Switching Characteristics (Module 3)
- Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Pinout Information (Module 4)

# ANNEX D. FULL D'ESPECIFICACIONS DE LA PLACA FPGA AVNET

productbrief

AvnetAvenue

## Xilinx Virtex-II Pro™ Development Kit

The Virtex-II Pro™ Development Kit from Avnet Design Services offers system level development on a highly advanced hardware platform. The Xilinx Virtex-II Pro family, based on the highly successful Virtex-II architecture, provides a unique environment for developing high-performance microprocessor and I/O intensive applications. The embedded 32-bit IBM PowerPC™ RISC processor delivers over 420 Dhrystone MIPS at 300 MHz, with a high-bandwidth interface to on-chip programmable logic. Additionally, Rocket I/O™ 3.125 Gbps transceivers support emerging connectivity standards. Time to market is greatly accelerated for implementation of advanced peripherals, IP and high-speed protocols such as 10 Gig Ethernet, Fibre Channel, OC-192 and InfiniBand.

**The Xilinx Virtex-II Pro Development Kit includes:**

- Virtex-II Pro development board with power supply daughtercard
- User's guide
- Bill of materials
- Schematics
- AC/DC power supply module
- Custom serial cable

Not included:

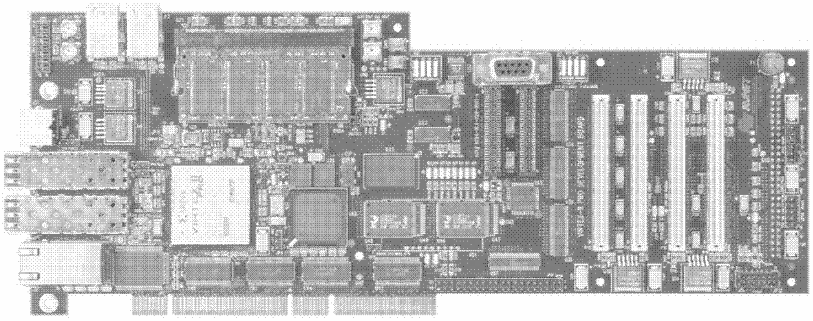
- XPAK module and holder
- SFP modules

### Target Applications:

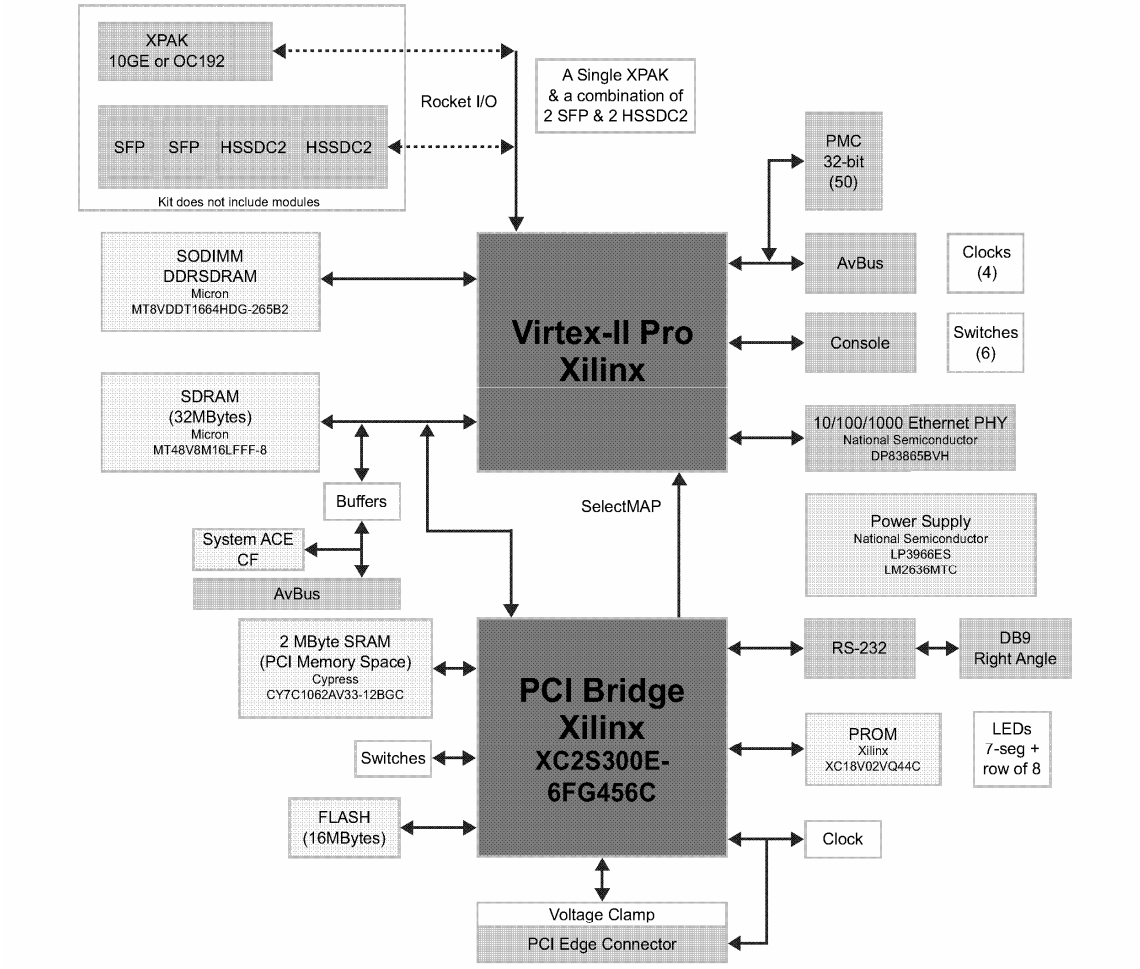
- Packet switching
- Network security
- Storage area networks
- Servers and super computers
- Video computing/transmission
- High-speed serial interfaces (RapidIO, FiberChannel and Hypertransport)

### Features:

- FPGA
  - Xilinx Virtex-II Pro™ XC2VP7-FF896 or XC2VP20-FF896
- High-speed serial communication
  - Two HSSDC2 connectors (InfiniBand, user may replace with Fibre Channel)
  - Pads for an XPAK module (10Gb Ethernet, OC-192)
  - Receptacles for 2 SFP modules (Gigabit Ethernet, Fibre Channel, InfiniBand)
- Board I/O connectors
  - 32-bit PCI Mezzanine Card (PMC) connectors
  - Four 140-pin general purpose I/O expansion connectors (AvBus)
- Memory
  - Micron DDR SDRAM SODIMM (128MB expandable to 1GB)
  - Micron Mobile SDRAM (two 8Mbit x 16 devices, 32MB total)
  - Cypress asynchronous SRAM (512Kbit x 32, 2MB total)
  - Intel StrataFlash® (16MB total)
  - Compact FLASH card
- Communication
  - National 10/100/1000 MBit/s Ethernet PHY
  - RS-232 serial ports
- PCI
  - PCI bridge - Xilinx Spartan™-IIE XC2S300E-FG456 FPGA
  - Windows based GUI interface
  - Configuration or file transfer & system control
  - Universal PCI connector (32-bit or 64-bit slot compatible)
  - Support for both 3.3V and 5.0V PCI signaling
- Power
  - Power supply daughter card (+3.3V and +2.5V rails @ 14A total)
  - 50 Watt AC/DC +5.0V power supply
  - National linear regulators
- Configuration
  - Bridge (Spartan-IIE)
    - Xilinx PROM XC18V02-VQ44
  - In-system programmable PROM
  - Target (Virtex-II Pro)
    - Xilinx System ACE™ CF
    - FLASH or SRAM via PCI and Windows application



AVNET electronics marketing




**Supporting suppliers:**

**Ordering Information:**

| Part Number | Hardware | Resale |
|---|---|---|
| ADS-XLX-V2PRO-DEVP7-5 | Xilinx Virtex-II Pro Development Kit with XC2VP7, -5 speed grade | $1,995.00 USD |
| ADS-XLX-V2PRO-DEVP7-6 | Xilinx Virtex-II Pro Development Kit with XC2VP7, -6 speed grade | $2,495.00 USD |
| ADS-XLX-V2PRO-DEVP20-5 | Xilinx Virtex-II Pro Development Kit with XC2VP20, -5 speed grade | $2,995.00 USD |

**Contact Information:**

North America
2211 South 47th St.
Phoenix, Arizona 85034
United States
Tel: +1 800 585 1602

Europe
Gruber Str. 60
85586 Poing
Germany
Tel: +49 (0) 8121 777279

Asia
7 Changi South St. 2
#01-00
Singapore, 486415
Tel: +6580 6000

**www.ads.avnet.com**

# ANNEX E. GENERADOR DE PAQUETS EN LINUX

El present annex ofereix la descripció de com s'han modificat les llibreries de Libnet per a poder crear trames d'Ethernet amb paquets de control de tipus *Setup* (com els definits en l'apartat 3.4.2 de la memòria). També s'ofereix el codi utilitzat com a client OBS en les proves presentades en el CAPÍTOL 5.

## E.1. Introducció

Com a màquina generadora de trames OBS s'ha utilitzat un PC equipat amb el sistema operatiu Linux. En Linux es poden utilitzar de forma molt senzilla dues llibreries que ens permeten generar i rebre trames/paquets segons els formats que nosaltres desitgem. Per a generar i transmetre trames utilitzem la llibreria LIBNET, i per a rebre trames i processar-les es pot utilitzar la llibreria LIBPCAP.

## E.2. Libnet

El LIBNET [35][36] és una llibreria per a generar i transmetre trames segons varis formats definits per estàndards, o segons qualsevol format que es necessiti. Aquest últim és el nostre cas, ja que com no hi ha un format estàndard de les trames de control d'OBS, hem hagut de generar les estructures de dades per a poder processar el format de trames escollit.

Per a poder utilitzar el format desitjat cal modificar les llibreries, compilar-les i tornar a crear la llibreria per al Linux [37]. Els fitxers modificats, i el contingut dels mateixos s'especifica a continuació.

### 1) libnet-headers.h

Aquest fitxer conté les capçaleres dels diferents tipus de paquets i datagrames que conté la llibreria.

```
#define LIBNET_OBS_H                0x08  /**< OBS header: 8 bytes */

/*
 * OBS header
 * Static header size: 8 bytes
 */
struct libnet_obs_hdr
{
      u_int16_t   oh_nodda;   /* destination node address */
      u_int16_t   oh_nodsa;   /* origin node address */
      u_int16_t   oh_idburst; /* burst ID */
      u_int8_t    oh_type;    /* burst Type */
      u_int8_t    oh_len;     /* message length */
};
```

### 2) libnet-functions.h

Aquest fitxer conté les funcions declarades per a poder crear el tipus de datagrama que es vol, en el nostre cas, construir un paquet de control OBS.

```
libnet_ptag_t
libnet_build_obs(u_int16_t nodda, u_int16_t nodsa, u_int16_t idburst,
u_int8_t type, u_int8_t len, u_int8_t *payload, u_int32_t payload_s,
libnet_t *l, libnet_ptag_t ptag);
```

### 3) libnet-structures.h

El fitxer libnet-structures.h conté informació per identificar els tipus PBlock segons el tipus de datagrama.

```
#define LIBNET_PBLOCK_OBS_H         0x40  /* OBS header */
```

### 4) libnet_build_obs.c

Aquest fitxer és de creació pròpia, i conté la funció declarada abans per a crear un paquet de control OBS.

```
#if (HAVE_CONFIG_H)
#include "../include/config.h"
#endif
#if (!(_WIN32) || (__CYGWIN__))
#include "../include/libnet.h"
#else
#include "../include/win32/libnet.h"
#endif

libnet_ptag_t
libnet_build_obs(u_int16_t nodda, u_int16_t nodsa, u_int16_t idburst,
      u_int8_t  type,  u_int8_t  len,  u_int8_t  *payload,  u_int32_t
payload_s,
      libnet_t *l, libnet_ptag_t ptag)
{
      u_int32_t n;
      libnet_pblock_t *p;
      struct libnet_obs_hdr obs_hdr;

      if(l == NULL){
            return (-1);
      }

      n = LIBNET_OBS_H + payload_s;

      p = libnet_pblock_probe(l, ptag, n, LIBNET_PBLOCK_OBS_H);
      if(p == NULL) {
            return (-1);
      }

      memset(&obs_hdr, 0, sizeof(obs_hdr));
      obs_hdr.oh_nodda  = htons(nodda);
```

```
      obs_hdr.oh_nodsa  = htons(nodsa);
      obs_hdr.oh_idburst      = htons(idburst);
      obs_hdr.oh_type   = type;
      obs_hdr.oh_len          = len;


      n    =    libnet_pblock_append(l,    p,    (u_int8_t    *)&obs_hdr,
LIBNET_OBS_H);
      if(n == -1) {
            goto bad;
      }

      if ((payload && !payload_s) || (!payload && payload_s)) {
        snprintf(l->err_buf, LIBNET_ERRBUF_SIZE,
                      "%s(): payload inconsistency\n", __func__);
        goto bad;
    }
    if (payload && payload_s) {
        n = libnet_pblock_append(l, p, payload, payload_s);
        if (n == -1)
        {
            goto bad;
        }
    }

      return(ptag    ?    ptag    :    libnet_pblock_update(l,    p,    0,
LIBNET_PBLOCK_OBS_H));

bad:
      libnet_pblock_delete(l, p);
      return (-1);
}
```

## E.3. Libpcap

La llibreria LIBPCAP [38] és la més utilitzada a l'hora de processar trames i paquets en els sistemes Linux [39]. Ofereix facilitats a l'hora de capturar els paquets, ja sigui en forma promiscua, a ràfegues, segons certs filtres definits, etc. En el nostre cas, la llibreria s'utilitza per a capturar i processar la trama de resposta que genera el node OBS.

## E.4. Programa client OBS

A continuació proporcionem el codi utilitzat com a client OBS en les proves efectuades en el cos de la memòria.

```
/*******************************
      OBS Client Application
      Author: Joan Triay
*******************************/

#include <string.h>
#include <stdio.h>
#include <libnet.h>
```

```
#include <stdlib.h>
#include <pcap.h>
#include <time.h>
#include <sys/types.h>

/*
 * Main Program
 */
int main(int argc, char **argv)
{
	// Declaration of variables
	static libnet_t *l = NULL;	// Libnet context
	u_short payload_s;		// payload length of the OBS Frame
	u_int16_t *burst_id;		// Local Burst ID
	int nframes, result, i, j;
	float randoffset, randlength; // Random float values
	int randoffset2, randlength2; // Random values

	// Initialized variables
	// Origin Ethernet Address (e.g. the PC's physical address
	char or_eth_add[6] = {0x00, 0x14, 0x38, 0x05, 0xE7, 0x3C};
	// Destination Ethernet Address (e.g. the OBS Node)
	char des_eth_add[6] = {0x06, 0x05, 0x04, 0x03, 0x02, 0x01};
	char *device = NULL;	// Ethernet physical device
	char errbuf[LIBNET_ERRBUF_SIZE];	// Error buffer
	char payload[1024];			// payload of the OBS Frame

	// Variables for storing time values
	struct timeval tvbuf_s, tvbuf_e;
	struct timezone tzbuf_s, tzbuf_e;

	// OBS variables used to construct the control packet
	char offset[4];
	char len_burst[4];


	// Start of the application
	printf("OBS Client\n");
	if(argc != 5) {
		printf("usage:\t%s\t\t [-i interface] [-n frames]\n",
argv[0]);
		return 0;
	}
	else {
		// Set the device and other variables
		device = malloc(strlen(argv[2]));
		strcpy(device, argv[2]);
		nframes = atoi(argv[4]);
		printf("Device: %s\n", device);
		printf("Number of Frames to transmit: %d\n", nframes);
	}

	// Initialize the burst_id
	burst_id = malloc(sizeof(u_int16_t));
	*burst_id = 0x0001;

	for(j=0; j<3; j++){
		offset[j] = 0x00;
		len_burst[j] = 0x00;
	}
```

```
      // Randomized offsets and burst length;
      srand((unsigned)time(NULL));


      gettimeofday(&tvbuf_s, &tzbuf_s);
      for(i = 0; i<nframes; i++) {
            // Generate random numbers
            randoffset = (float) (rand()) / (RAND_MAX);
            randoffset2 = 100*randoffset;
            randlength = (float) rand() / (RAND_MAX);
            randlength2 = 10*randlength;

            /* Libnet context: Link Injection Type, device network
interface and error buffer */
            l = libnet_init(LIBNET_LINK, device, errbuf);
            if(l == NULL) {
                  printf("libnet: %s\n", errbuf);
                  goto bad;
            }

            // Fill the offset and len_burst variables, and then
            // the payload_s.
            offset[3] = randoffset2 & 0xFF;
            len_burst[3] = randlength2 & 0xFF;

            payload_s = snprintf(payload, sizeof(payload),
"%c%c%c%c%c%c%c%c",offset[0], offset[1], offset[2],
offset[3],len_burst[0], len_burst[1], len_burst[2],
len_burst[3],0x02);

            // declare all ptag I need
            libnet_ptag_t eth = 0, obs = 0;

            // Fill OBS data
            obs = libnet_build_obs(
                  0x1234,           // Destination OBS Address
                  0x5678,           // Origin OBS Address
                  *burst_id,        // Burst ID
                  0x00,             // Type
                  0x09,             // Len of PDU
                  payload,          // PDU
                  payload_s,        // length of PDU
                  l,                // Libnet handle
                  0);               // libnet id
            if(obs == -1) {
                  printf("Can't build OBS packet: %s\n",
libnet_geterror(l));
                  goto bad;
            }

            // Fill the Ethernet Data
            eth = libnet_build_ethernet(
                  des_eth_add,      // Destination Ethernet Address
                  or_eth_add,       // Origin Ethernet Address
                  LIBNET_OBS_H + payload_s,     // Length
                  NULL,             // Null payload --> autocomputed
                  0,                // payload length --> autocomputed
                  l,                // Libnet handle
                  0);               // Libnet ID
            if(eth == -1) {
```

```
                  printf("Can't build Eth packet: %s\n",
libnet_geterror(l));
                  goto bad;
            }

            // Transmit the packet - write it to the wire
            result = libnet_write(l);
            if(result == -1) {
                  printf("Write error: %s\n", libnet_geterror(l));
                  goto bad;
            }

            // Regenerate the libnet instance
            libnet_destroy(l);

            // Update the burst_id
            *burst_id = *burst_id + 0x0001;

      }
      gettimeofday(&tvbuf_e, &tzbuf_e);

      // Calculate the amount of time taken to carry out the
      // generation and transmission of packets
      long int diff_time_2_s = tvbuf_e.tv_sec - tvbuf_s.tv_sec;
      long int diff_time_2_us = tvbuf_e.tv_usec - tvbuf_s.tv_usec;
      if(diff_time_2_us < 0) {
            diff_time_2_s++;
            diff_time_2_us = tvbuf_e.tv_usec;
      }
      printf("%d en %ld s %ld usec\n", nframes, diff_time_2_s,
diff_time_2_us);

      return(EXIT_SUCCESS);

// If something goes wrong
bad:
      libnet_destroy(l);
      return(EXIT_FAILURE);
}
```

# ANNEX F. DISSENY DEL SOFTWARE

Aquest annex descriu una part important del disseny del programa de control realitzat en el CAPÍTOL 3, els diagrames de flux de les principals tasques del sistema.

## F.1. Diagrames de flux

A continuació realitzarem una breu descripció dels principals fluxOS del OCPS, associats, principalment, al processament dels paquets de control, de les peticions de transmissió de ràfega, a les reserves de recursos i al control del temporitzador global del sistema que defineix el seguiment al vector ranurat SRV (*Slotted Resource Vector*).

### F.1.1. Recepció de ràfega

El diagrama de flux del processament del paquet de control, i la posterior recepció de la ràfega es mostra en la Fig. F.1.

El flux comença amb el processament del paquet de control, i es determina, primer de tot, si el paquet de control té per camp *node de destinació* l'adreça del node que realitza aquest processament. Si és així, el que s'ha de fer és la recepció de la ràfega. En cas contrari es realitzaria la commutació d'aquesta, que s'explica en l'apartat F.1.2.

Un cop es determina la recepció de la ràfega, es comprova si hi ha suficient espai en el *buffer* de ràfegues per a emmagatzemar la informació relativa a la ràfega que s'ha de rebre. Si no hi ha espai s'elimina la petició i es surt del present flux, per passar a executar alguna altra tasca. En cas contrari, es realitza el càlcul del temps actual per a comparar-lo amb el temps de recepció real del paquet de control. Aquest període permet actualitzar el valor de l'*offset* que conté el paquet i determinar si aquest ja està en temps negatiu, el que voldrà dir que la ràfega, òpticament ja s'hauria d'estar rebent, i que per tant, s'ha de descartar, ja que no s'ha realitzat la reserva en el temps adient.

Si l'*offset* segueix sent positiu significa llavors que encara tindrem temps de processar la reserva. En funció del valor anterior, s'actualitza el punter del SRV i es col·loca en la posició on s'hauria de començar a rebre la ràfega (menys un cert valor de ranures de guarda).

A continuació es realitza la comprovació de si el node actua com a frontera, o també com a node central. Si actua de les dues formes, es necessiten reservar recursos tant per a la commutació de la ràfega en el OXC com per a la recepció òptica, ORX. Si la comprovació de recursos, és a dir, en les ranures sol·licitades, igual a la longitud de la ràfega (+2 ranures, de guarda), és satisfactòria, llavors es realitza la reserva dels recursos, s'actualitzen els valors en el SRV i s'afegeix la informació de la ràfega en el *buffer*. Si no és així,

llavors simplement es descarta la petició. En ambdós casos, el punt final del procés és passar a realitzar alguna altra tasca segons l'ordre de l'algoritme principal del programa, o restar en espera d'alguna altra acció.

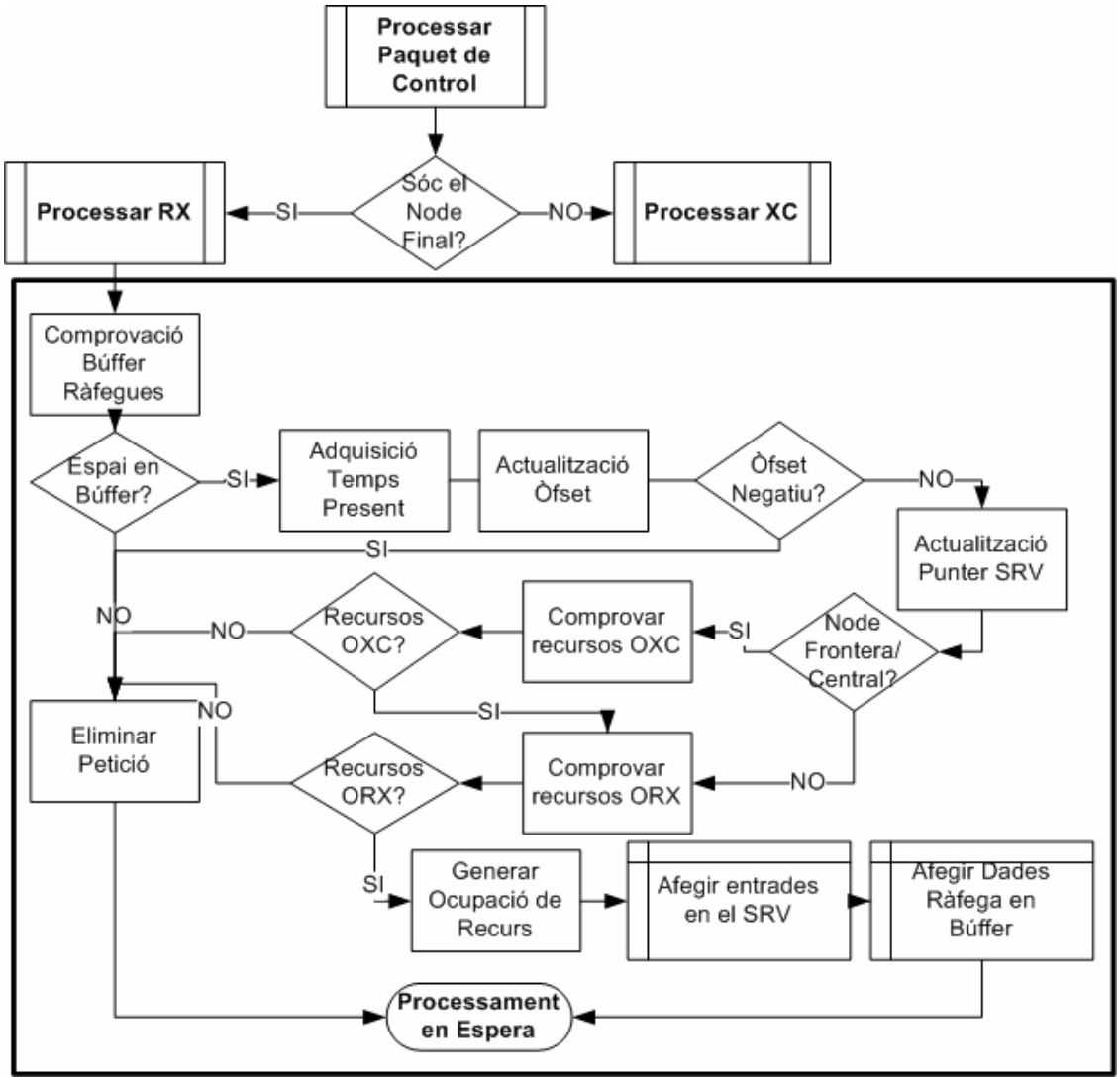



**Fig. F.1.** Diagrama de flux de la recepció de ràfega.

## F.1.2. Commutació de ràfega

La següent figura Fig. F.2 es correspon al diagrama de flux del processament de reserves per a la commutació d'una ràfega òptica. Com es pot observar el flux s'assembla molt al de la recepció d'una ràfega. En aquest cas, però, es determina per la lectura de les dades del paquet de control, que el node que processa actualment el paquet de control no és el node destinació, i per tant, s'ha de realitzar sols una commutació.

Les principals diferències respecte l'anterior flux són:

- Després de realitzar el càlcul del nou *offset*, i determinar que segueix sent positiu, es realitzen les consultes a les taules d'encaminament i commutació. La primera per a determinar el següent node en el camí, tant per al paquet de control com per a la ràfega òptica, i la segona consulta per a determinar segons els canals d'entrada del commutador i la destinació de la ràfega, quines són les configuracions del commutador que permeten realitzar aquesta commutació.

- En aquest flux només es comproven els recursos òptics del commutador, ja que el node no necessita rebre la ràfega.

- Un cop realitzada la reserva, s'han d'actualitzar els camps del paquet de control, i transmetre'l al següent node de la xarxa OBS en el camí cap al node destí, on es desensamblarà la ràfega. Això implica recalcular l'*offset*.

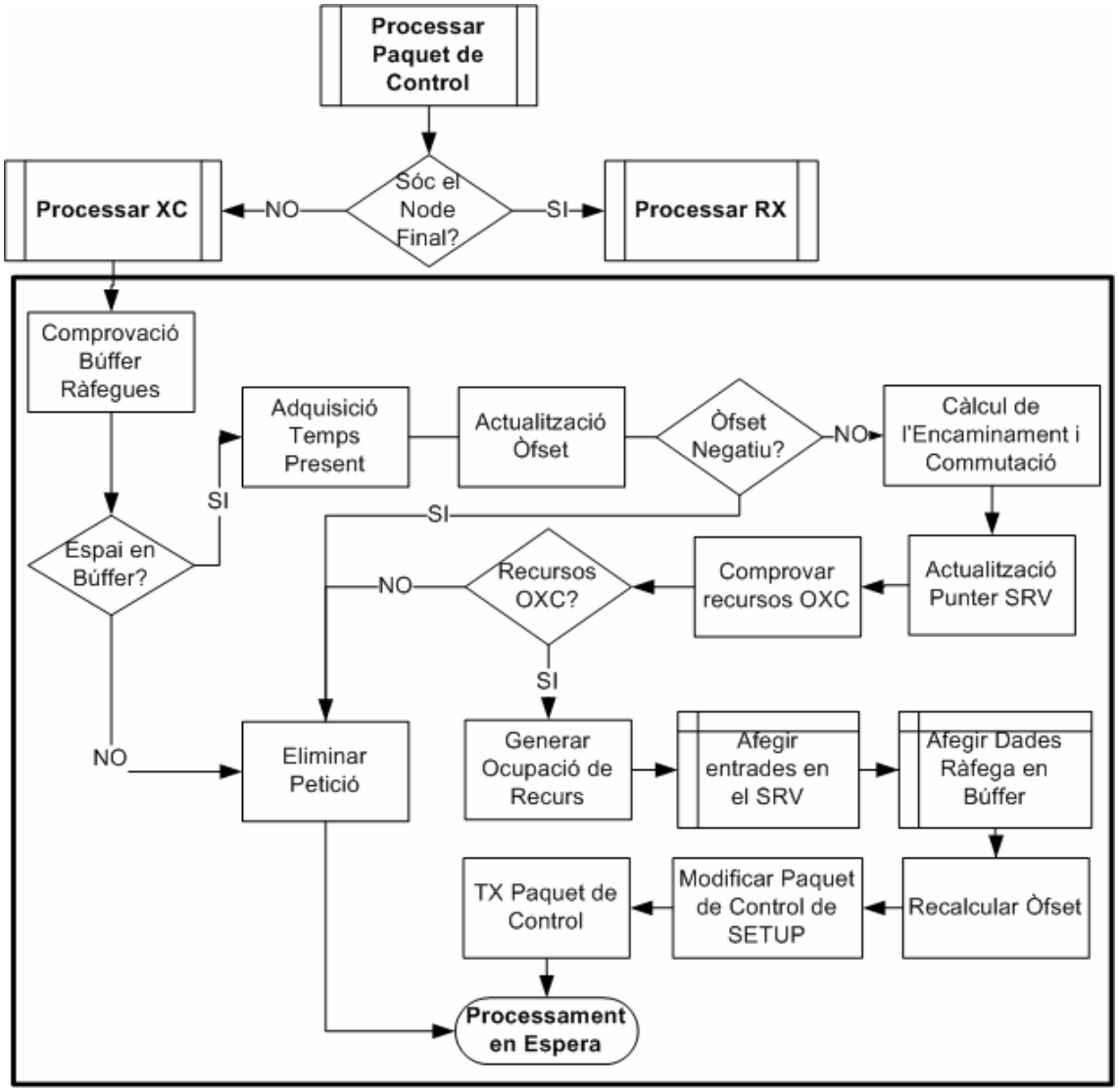



**Fig. F.2.** Diagrama de flux de commutació de ràfega.

### F.1.3. Transmissió de ràfega

La figura Fig. F.3 mostra el flux corresponent al processament dels recursos necessaris per a la transmissió d'una ràfega. Aquest flux és anàleg al de la recepció d'una ràfega, però presenta algunes diferències que comentarem.

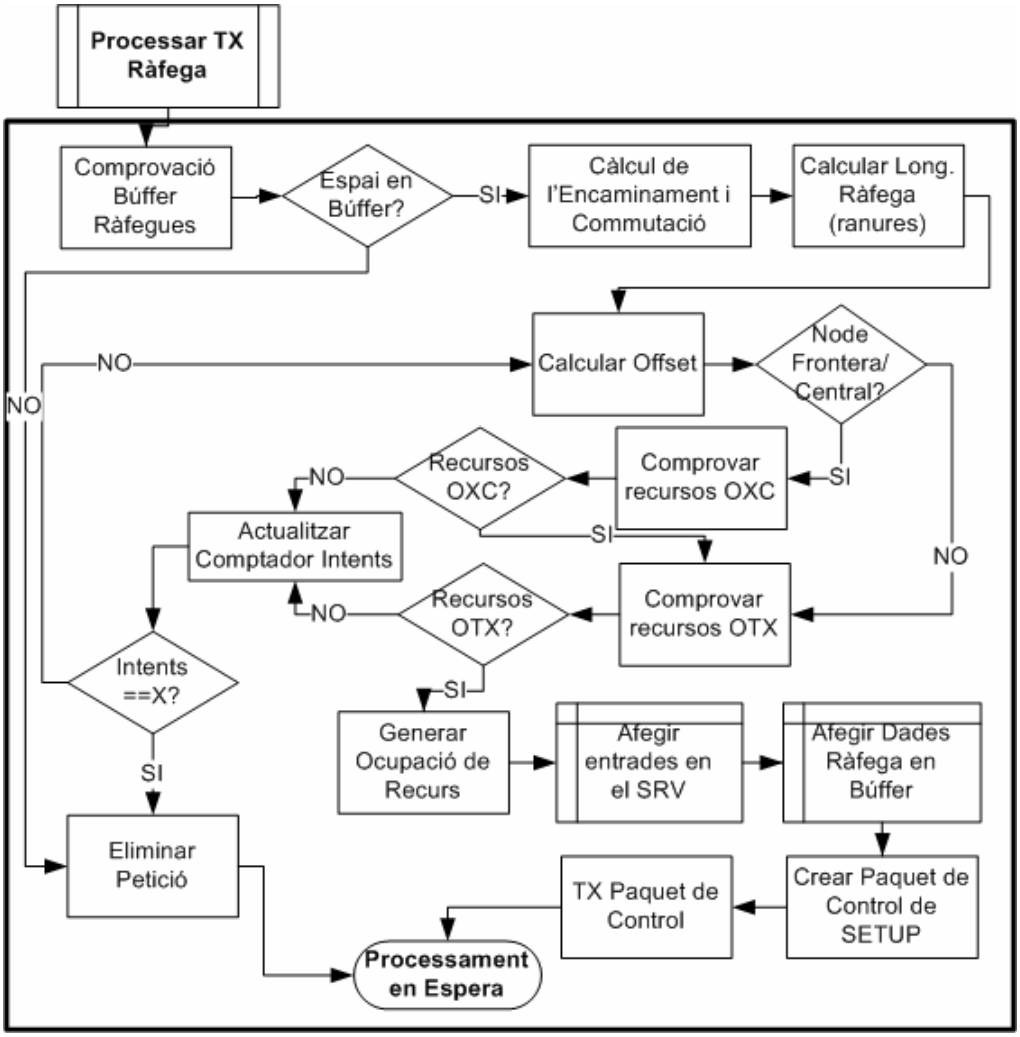



**Fig. F.3.** Diagrama de flux de transmissió de ràfega.

Després de revisar que hi hagi espai per emmagatzemar la informació de la ràfega que es vol transmetre en el *buffer*, es realitza el càlcul de l'encaminament i la commutació per a determinar per quins ports s'ha d'enviar el paquet de control, i per quina canal/port la ràfega òptica. Tot seguit es realitza el càlcul de la longitud de la ràfega i s'extreu, així, el nombre de ranures que ocupa.

A continuació es calcula l'*offset* tal com s'ha comentat en l'apartat 3.4.3. Segons el valor dels ports de sortida, longitud de la ràfega i *offset* es realitza la comprovació de si hi ha recursos lliures per a realitzar la transmissió. Al igual que en el cas de la recepció de la ràfega, s'ha de determinar també si el node

actua tant de node frontera, com de central, perquè si és així, també s'han de comprovar el recursos del commutador òptic que integra. Si la reserva no és possible, s'augmenta un comptador d'intents, i es comprova si el nombre d'intents ha assolit un cert valor "X". Si no és així, es torna a calcular un nou *offset*, i es tornen a comprovar els recursos.

Per altra banda, si la reserva sí ha estat possible, es genera l'ocupació del recurs en el SRV, s'afegeixen la informació de la ràfega en el *buffer*, i es crea el paquet de control de *Setup*.

Finalment es transmet el paquet de control i es passa a executar la següent tasca o el programa es posa en espera de processar alguna altra interrupció.

### F.1.4. Processar petició de transmissió de ràfega

En la següent figura Fig. F.4 es mostra el flux per a processar les peticions de transmissió de ràfega. Les peticions de transmissió de ràfega són excepcions externes que generen una interrupció al processament normal del programa. La funció que atén la interrupció s'encarrega d'agafar les dades en memòria sobre la ràfega, comprovar si hi ha espai en el *buffer* d'entrada de peticions suficient, i si és així, emmagatzema la petició.

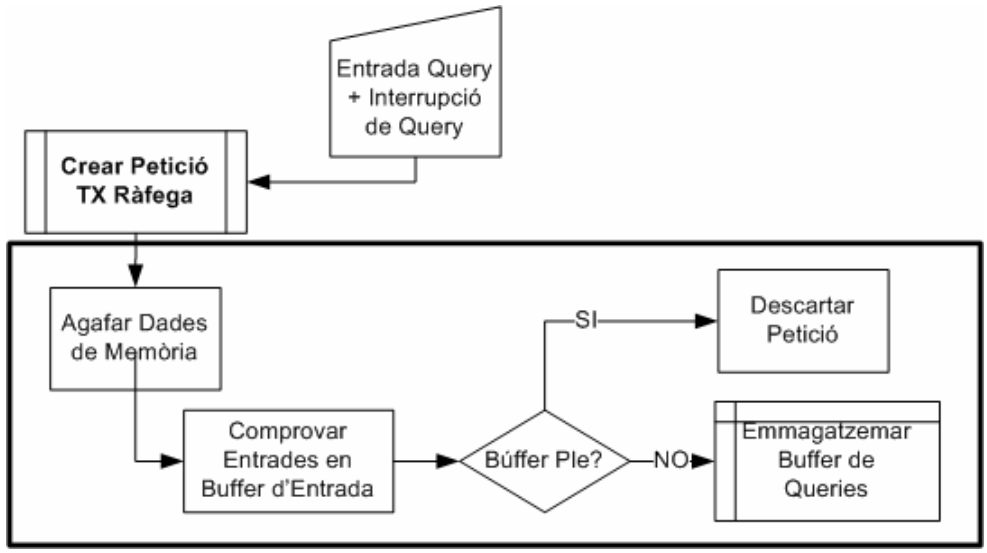



**Fig. F.4.** Diagrama de flux de recepció de petició per a la transmissió d'una ràfega.

Posteriorment, segons el flux de l'algoritme principal del programa, es comprova si hi ha alguna petició en aquest *buffer* d'entrada de peticions, i si és així, es realitza el processament de transmissió de ràfega.

### F.1.5. Processar recepció paquet de control

El flux de processament i recepció d'un paquet de control es mostra en la figura Fig. F.5. Aquest s'assembla molt al flux de recepció d'una petició de transmissió de ràfega, però aquí, el que es reben són paquets de control. Al igual que en el flux anterior, si els *buffers* d'entrada de paquets estan plens, el paquet en qüestió es descarta. Altrament, s'emmagatzema en el *buffer* per al seu posterior processament segons l'algoritme principal del programa.

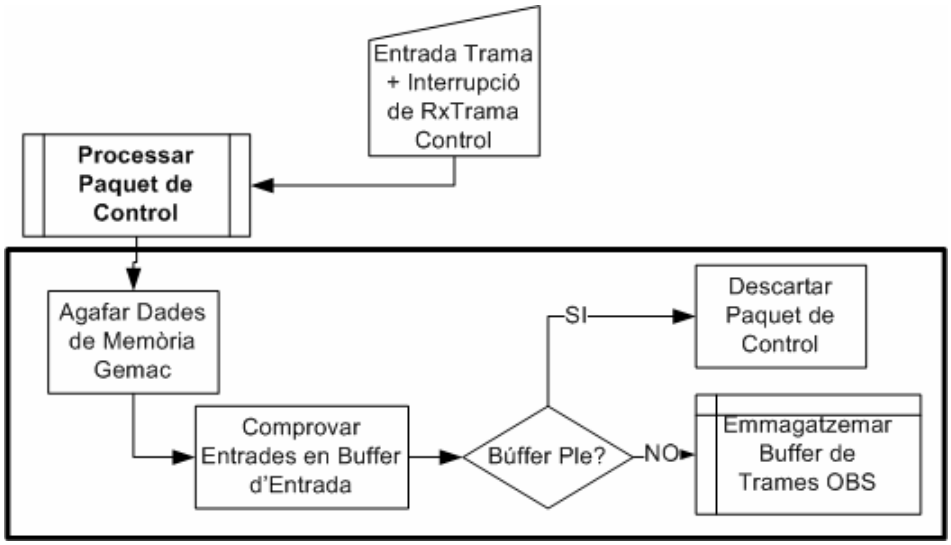



**Fig. F.5.** Diagrama de flux de la recepció d'un paquet de control.

## F.1.6. Processament temporitzador global i senyalització

La figura Fig. F.6 mostra com es processen les interrupcions que marquen el pas de ranura del vector SRV. En la mateixa funció d'atenció a la interrupció, es realitza el processament de la senyalització segons la informació continguda en la ranura actual del SRV.

Així, primer de tot es llegeix la posició actual en el SRV i es comprova el tipus d'ocupació. Si en la ranura no està actiu el bit d'ocupació, llavors simplement es regenera la informació de la ranura als seus valors per defecte, i s'avança la posició del vector.

Per altra banda, si està ocupat, vol dir que hi ha recursos assignats que cal processar i senyalitzar. En aquest sentit es realitza la comprovació del tipus d'ocupació. Si la ranura marca una *Ocupació Inicial*, es comprova de quin tipus és l'acció, i es senyalitza al Pla de Dades per a la TX o RX si escau, i si no al OXC per a fer la commutació. Així mateix, si el node actua tant de frontera com de central, a la vegada que s'haurà de senyalitzar al Pla de Dades, també s'haurà de senyalitzar la commutació.

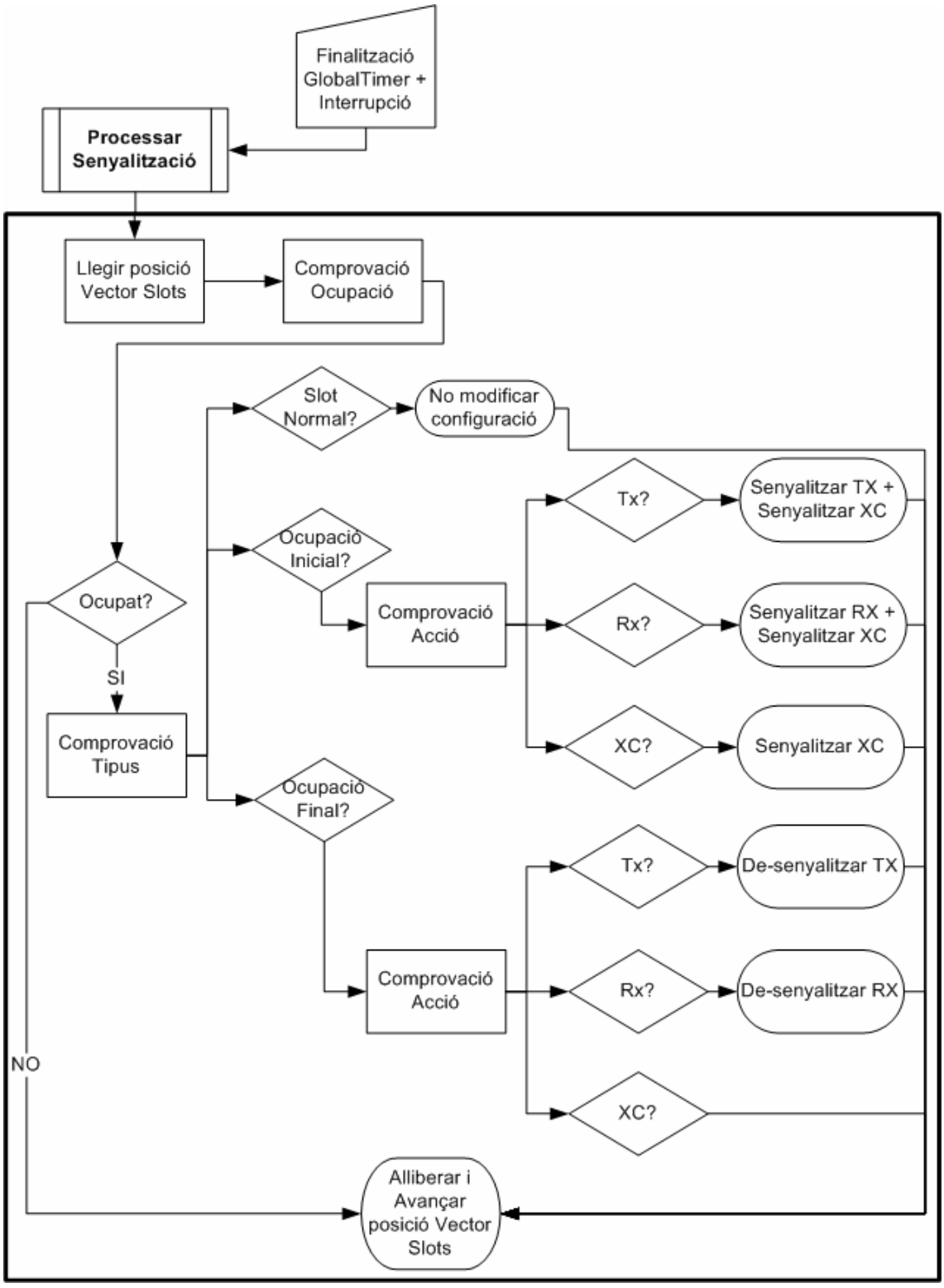



**Fig. F.6.** Diagrama de flux del processament del temporitzador global associat al SRV.

En cas que la marca sigui d'*Ocupació final*, es contempla solament la de-senyalització al Pla de Dades, és a dir, se l'informa a aquest que aturi la

transmissió o recepció de la ràfega òptica. Finalment, si la ranura és de tipus *Normal*, no es realitza cap senyalització.

En tots els casos anteriors, un cop realitzada la senyalització, o no senyalització, s'allibera la informació de la ranura i s'actualitza el punter del SRV.

Així mateix, en aquest flux, just en el moment d'atendre la interrupció, es reinicialitza el temporitzador *GlobalTimer*, per a que marqui en el temps configurat una nova interrupció per a processar la següent ranura del SRV.

### F.1.7. Algoritme principal del programa

Finalment tenim el flux de l'algoritme principal del programa, que es mostra en la figura Fig. F.7. Tal com s'ha descrit en l'apartat 3.5.2, hi ha dues tasques de baixa prioritat, en relació amb l'atenció al *GlobalTimer*, la de processament de peticions, i la de processament de paquets de control. Ambdues realitzen la comprovació de si hi ha peticions o paquets a processar, i si és així, es prossegueix amb l'execució dels fluxos descrits en els apartats anteriors (F.1.1. Recepció de ràfega, F.1.2. Commutació de ràfega i F.1.3. Transmissió de ràfega).

El nombre de cops que es processa cada tasca depèn de la prioritat amb la que es vol processar en relació amb l'altra.

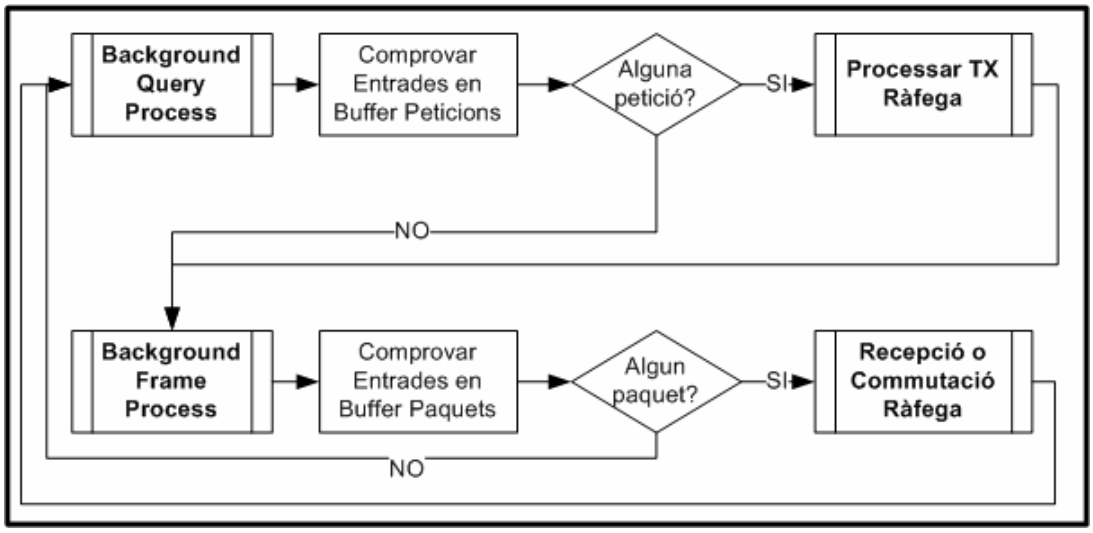



**Fig. F.7.** Diagrama de flux de l'algoritme principal del programa.

# ANNEX G. DISSENY DEL HARDWARE

El present annex descriu algunes característiques importants en el disseny del sistema hardware sobre la placa de desenvolupament FPGA. També es descriu la configuració necessària per adequar les interfícies Ethernet i SFP físiques de la placa als senyals de rellotge.

## G.1. Disseny general del hardware

### G.1.1. Llistat de dispositius

La següent Taula G.1 proporciona un llistat dels diferents components que componen el sistema hardware. S'especifica per a cadascun el tipus de disposiutiu i la versió d'*Intellectual Property* utilitzada en la versió final del programa. La present taula complementa la Taula 4.2 presentada en el CAPÍTOL 4.

**Taula G.1.** Llistat de dispositius i versions IP del sistema hardware.

| Nom Component | Tipus de Dispositiu | Tipus IP | Versió IP |
|---|---|---|---|
| ppc405_0 | Processador PowerPC 405 | ppc405 | 2.00.c |
| ppc405_1 | Processador PowerPC 405 | ppc405 | 2.00.c |
| ppc_plb | Bus PLB | plb_v34 | 1.02.a |
| ppc_opb | Bus OPB | opb_v20 | 1.10.c |
| ppc_dcr | Bus DCR | dcr_v20 | 1.00.a |
| ppc_plb2opb | Pont entre busos | plb2opb_bridge | 1.01.a |
| jtagppc_0 | Controlador JTAG | jtagppc_cntrl | 2.00.a |
| ppc_plbbram_cntlr | Controlador Memòria BRAM en PLB | plb_bram_if_cntlr | 1.00.b |
| data_plbbram_cntlr | Controlador Memòria BRAM en PLB | plb_bram_if_cntlr | 1.00.b |
| ppc_bram | Memòria BRAM | bram_block | 1.00.a |
| data_bram | Memòria BRAM | bram_block | 1.00.a |
| sram | Memòria SRAM | plb_emc | 2.00.a |
| sram_util_bus_split_0 | Utilitat | util_bus_split | 1.00.a |
| dcm_system | Gestor de Rellotge Digital | dcm_module | 1.00.a |
| dcm_dcr | Gestor de Rellotge Digital | dcm_module | 1.00.a |
| reset_block | Bloc de reinicialització | proc_sys_reset | 1.00.a |
| ppc_uart | Port sèrie UART | opb_uartlite | 1.00.b |
| ppc_gemac | MAC Gigabit Ethernet | plb_gemac | 1.01.a |
| ppc_intc | Controlador d'Interrupcions | opb_intc | 1.00.c |
| ppc_mii_clock_ management | Gestor de rellotge MII | mii_clock_ management | sense ver. |
| opb_timer_0 | Temporitzador | opb_timer | 1.00.b |
| opb_timer_1 | Temporitzador | opb_timer | 1.00.b |

## G.1.2. Diagrama hardware del sistema (programa XPS)

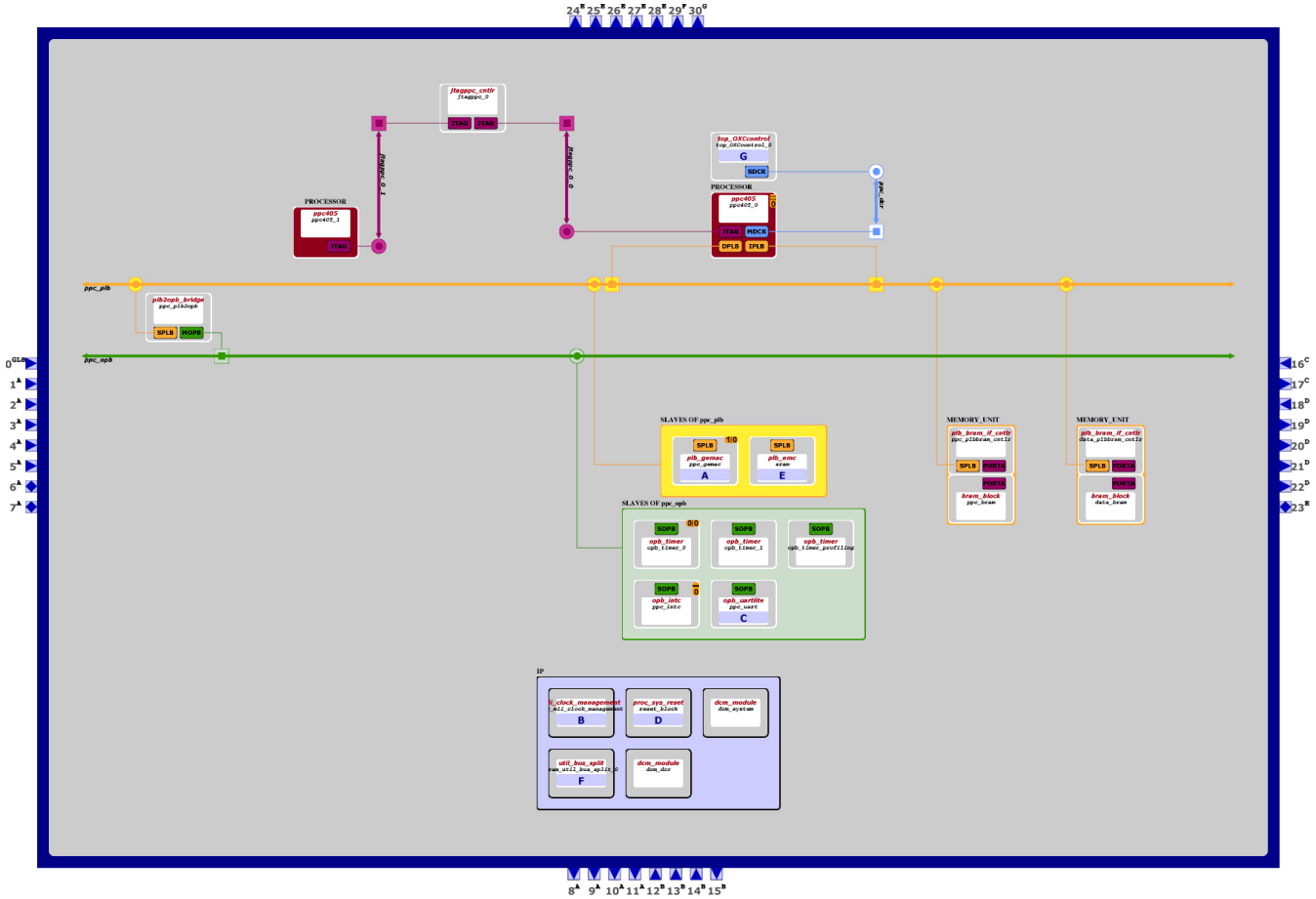



**Fig. G.1.** Diagrama del sistema (extret del programa XPS).

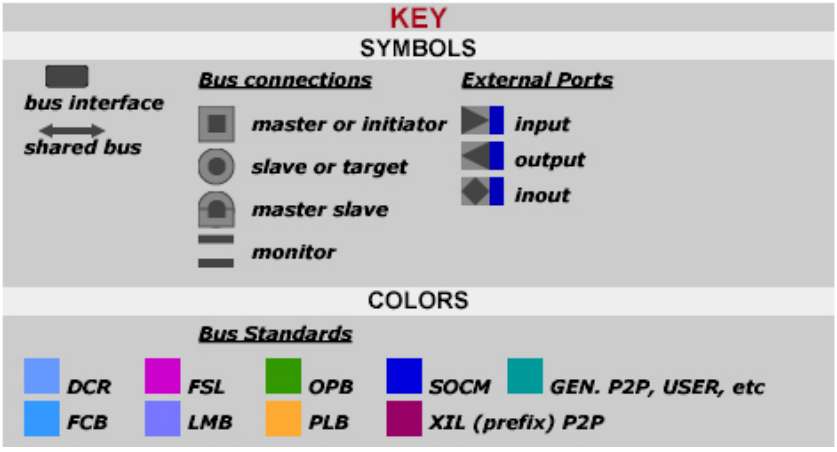



**Fig. G.2.** Llegenda del diagrama Fig. G.1.

## G.1.3. Fitxers de configuració del Xilinx Platform Studio

Amb els següents fitxers es pot crear totalment el sistema hardware en la FPGA Virtex-II Pro 30. Aquestes configuracions són les utilitzades durant els tests realitzats en el CAPÍTOL 5.

### *G.1.3.1. Fitxer system.mss*

```
BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER stdout = ppc_uart
 PARAMETER stdin = ppc_uart
END

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_0
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
 PARAMETER CORE_CLOCK_FREQ_HZ = 300000000
 PARAMETER EXTRA_COMPILER_FLAGS = -g -O3
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_1
 PARAMETER COMPILER = powerpc-eabi-gcc
```

```
 PARAMETER ARCHIVER = powerpc-eabi-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = jtagppc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plbarb
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = ppc_plb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = ppc_opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plb2opb
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc_plb2opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc_plbbram_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = ppc_uart
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gemac
 PARAMETER DRIVER_VER = 1.00.f
 PARAMETER HW_INSTANCE = ppc_gemac
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
 PARAMETER HW_INSTANCE = ppc_intc
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = opb_timer_0
 PARAMETER int_handler = TimeHandler0, int_port = Interrupt
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
```

```
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = data_plbbram_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = emc
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = sram
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = top_OXCcontrol_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = opb_timer_1
 PARAMETER int_handler = TimeHandler1, int_port = Interrupt
END
```

### *G.1.3.2. Fitxer system.mhs*

```
# Parameters
 PARAMETER VERSION = 2.1.0


# Global Ports
 PORT sys_clk = dcm_clk_s, DIR = I
 PORT system_reset = system_reset, DIR = I
 PORT rx = rx, DIR = I
 PORT tx = tx, DIR = O
 PORT clk_125_p = clk_125_p, DIR = I
 PORT clk_125_n = clk_125_n, DIR = I
 PORT gmii_col = gmii_col, DIR = I
 PORT gmii_crs = gmii_crs, DIR = I
 PORT gmii_txd = gmii_txd, DIR = O, VEC = [7:0]
 PORT gmii_tx_en = gmii_tx_en, DIR = O
 PORT gmii_tx_er = gmii_tx_er, DIR = O
 PORT gmii_tx_clk = gmii_tx_clk, DIR = O
 PORT gmii_rxd = gmii_rxd, DIR = I, VEC = [7:0]
 PORT gmii_rx_dv = gmii_rx_dv, DIR = I
 PORT gmii_rx_er = gmii_rx_er, DIR = I
 PORT gmii_rx_clk = gmii_rx_clk, DIR = I
 PORT gmii_rst_n = gmii_rst_n, DIR = O
 PORT PHY_Mii_clk = PHY_Mii_clk, DIR = IO
 PORT PHY_Mii_data = PHY_Mii_data, DIR = IO
 PORT  fpga_0_SRAM_Mem_DQ_pin  =  fpga_0_SRAM_Mem_DQ,  DIR  =  IO,  VEC  =
[0:31]
 PORT fpga_0_SRAM_Mem_A_pin = fpga_0_SRAM_Mem_A, DIR = O, VEC = [8:29]
 PORT  fpga_0_SRAM_Mem_BEN_pin  =  fpga_0_SRAM_Mem_BEN,  DIR  =  O,  VEC  =
[0:3]
 PORT fpga_0_SRAM_emc_disable_sdram_pin = net_vcc, DIR = O
 PORT fpga_0_SRAM_Mem_WEN_pin = fpga_0_SRAM_Mem_WEN, DIR = O
 PORT  fpga_0_SRAM_Mem_OEN_pin  =  fpga_0_SRAM_Mem_OEN,  DIR  =  O,  VEC  =
[0:0]
 PORT  fpga_0_SRAM_Mem_CEN_pin  =  fpga_0_SRAM_Mem_CEN,  DIR  =  O,  VEC  =
[0:0]
```

```
 PORT fpga_0_SRAM_emc_disable_flash_pin = net_vcc, DIR = O
 PORT fpga_0_SRAM_Mem_RPN_pin = fpga_0_SRAM_Mem_RPN, DIR = O
 PORT fpga_0_SRAM_emc_disable_buffer_pin = net_vcc, DIR = O
 PORT fpga_0_SRAM_emc_disable_sysace_pin = net_vcc, DIR = O
 PORT  top_OXCcontrol_0_oxc_switch_pin  =  top_OXCcontrol_0_oxc_switch,
DIR = O


# Sub Components
BEGIN mii_clock_management
 PARAMETER INSTANCE = ppc_mii_clock_management
 PORT gtx_clk_in_p = clk_125_p
 PORT gtx_clk_in_n = clk_125_n
 PORT rx_clk_in = gmii_rx_clk
 PORT rx_clk_out = gmii_rx_clk_int
 PORT tx_clk_out = gmii_tx_clk
 PORT tx_clk_in = gmii_tx_clk_int
 PORT gtx_clk_out = gtx_clk_int
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
 BUS_INTERFACE IPLB = ppc_plb
 BUS_INTERFACE DPLB = ppc_plb
 BUS_INTERFACE MDCR = ppc_dcr
 PORT PLBCLK = sys_clk_s
 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
 PORT RSTC405RESETCORE = RSTC405RESETCORE
 PORT RSTC405RESETSYS = RSTC405RESETSYS
 PORT CPMC405CLOCK = proc_clk_s
 PORT DCRCLK = dcr_bus_clk_proc_s
 PORT EICC405EXTINPUTIRQ = ppc405_0_EICC405EXTINPUTIRQ
 PORT EICC405CRITINPUTIRQ = opb_timer_1_Interrupt
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_1
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_1
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_block
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RST_WIDTH = 20
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Slowest_sync_clk = sys_clk_s
 PORT Ext_Reset_In = system_reset
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
```

```
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = peripheral_rst
 PORT Dcm_locked = net_vcc
END

BEGIN plb_v34
 PARAMETER INSTANCE = ppc_plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
 PORT M_busLock = net_gnd
END

BEGIN opb_v20
 PARAMETER INSTANCE = ppc_opb
 PARAMETER HW_VER = 1.10.c
 PORT OPB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = ppc_plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_CLK_ASYNC = 0
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_RNG0_BASEADDR = 0xF0000000
 PARAMETER C_RNG0_HIGHADDR = 0xF7FFFFFF
 BUS_INTERFACE SPLB = ppc_plb
 BUS_INTERFACE MOPB = ppc_opb
 PORT PLB_Clk = sys_clk_s
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = ppc_plbbram_cntlr
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFFFFFF
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_include_burst_cacheln_support = 0
 BUS_INTERFACE PORTA = porta
 BUS_INTERFACE SPLB = ppc_plb
END

BEGIN bram_block
 PARAMETER INSTANCE = ppc_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = porta
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = ppc_uart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_DATA_BITS = 8
```

```
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BAUDRATE = 19200
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xF0000000
 PARAMETER C_HIGHADDR = 0xF00000FF
 BUS_INTERFACE SOPB = ppc_opb
 PORT OPB_Rst = peripheral_rst
 PORT RX = rx
 PORT TX = tx
END

BEGIN plb_gemac
 PARAMETER INSTANCE = ppc_gemac
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000FFFF
 PARAMETER C_DMA_TYPE = 1
 PARAMETER C_INCLUDE_GMII = 1
 PARAMETER C_INCLUDE_SerDes = 0
 PARAMETER C_INCLUDE_MIIM = 0
 PARAMETER C_INCLUDE_DMA_INTR_COALESCE = 0
 BUS_INTERFACE SPLB = ppc_plb
 PORT gtx_clk = gtx_clk_int
 PORT gmii_col = gmii_col
 PORT IP2INTC_Irpt = gemac_Interrupt
 PORT gmii_crs = gmii_crs
 PORT gmii_txd = gmii_txd
 PORT gmii_tx_en = gmii_tx_en
 PORT gmii_tx_er = gmii_tx_er
 PORT gmii_tx_clk = gmii_tx_clk_int
 PORT gmii_rxd = gmii_rxd
 PORT gmii_rx_dv = gmii_rx_dv
 PORT gmii_rx_er = gmii_rx_er
 PORT gmii_rx_clk = gmii_rx_clk_int
 PORT gmii_rst_n = gmii_rst_n
 PORT PHY_Mii_clk = PHY_Mii_clk
 PORT PHY_Mii_data = PHY_Mii_data
END

BEGIN opb_intc
 PARAMETER INSTANCE = ppc_intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_NUM_INTR_INPUTS = 2
 PARAMETER C_BASEADDR = 0xF0010000
 PARAMETER C_HIGHADDR = 0xF001FFFF
 BUS_INTERFACE SOPB = ppc_opb
 PORT Intr = opb_timer_0_Interrupt & gemac_Interrupt
 PORT Irq = ppc405_0_EICC405EXTINPUTIRQ
END

BEGIN bram_block
 PARAMETER INSTANCE = data_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = data_plbbram_cntlr_PORTA
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_system
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLKFX_MULTIPLY = 3
 PARAMETER C_CLKIN_PERIOD = 10.00000
```

```
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLKFX_BUF = TRUE
 PORT RST = net_gnd
 PORT CLKIN = dcm_clk_s
 PORT CLKFX = proc_clk_s
 PORT CLKFB = sys_clk_s
 PORT CLK0 = sys_clk_s
 PORT LOCKED = dcm_system_LOCKED
END

BEGIN opb_timer
 PARAMETER INSTANCE = opb_timer_0
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0xF0000100
 PARAMETER C_HIGHADDR = 0xF00001FF
 BUS_INTERFACE SOPB = ppc_opb
 PORT Interrupt = opb_timer_0_Interrupt
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = data_plbbram_cntlr
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0xF8000000
 PARAMETER c_highaddr = 0xF801FFFF
 BUS_INTERFACE SPLB = ppc_plb
 BUS_INTERFACE PORTA = data_plbbram_cntlr_PORTA
END

BEGIN util_bus_split
 PARAMETER INSTANCE = sram_util_bus_split_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE_IN = 32
 PARAMETER C_SPLIT = 30
 PARAMETER C_LEFT_POS = 8
 PORT Sig = fpga_0_SRAM_Mem_A_split
 PORT Out1 = fpga_0_SRAM_Mem_A
END

BEGIN plb_emc
 PARAMETER INSTANCE = sram
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_MEM0_BASEADDR = 0x10000000
 PARAMETER C_MEM0_HIGHADDR = 0x100FFFFF
 PARAMETER C_NUM_BANKS_MEM = 1
 PARAMETER C_MAX_MEM_WIDTH = 32
 PARAMETER C_MEM0_WIDTH = 32
 PARAMETER C_INCLUDE_BURST_CACHELN_SUPPORT = 1
 BUS_INTERFACE SPLB = ppc_plb
 PORT PLB_Clk = sys_clk_s
 PORT Mem_A = fpga_0_SRAM_Mem_A_split
 PORT Mem_DQ = fpga_0_SRAM_Mem_DQ
 PORT Mem_BEN = fpga_0_SRAM_Mem_BEN
 PORT Mem_WEN = fpga_0_SRAM_Mem_WEN
 PORT Mem_OEN = fpga_0_SRAM_Mem_OEN
 PORT Mem_CEN = fpga_0_SRAM_Mem_CEN
 PORT Mem_RPN = fpga_0_SRAM_Mem_RPN
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_dcr
```

```
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLKFX_BUF = TRUE
 PARAMETER C_CLKDV_BUF = FALSE
 PARAMETER C_CLKIN_PERIOD = 10.00000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER C_CLKFX_MULTIPLY = 3
 PARAMETER C_CLKFX_DIVIDE = 2
 PORT CLKIN = dcm_clk_s
 PORT CLK0 = dcr_bus_clk_s
 PORT CLKFB = dcr_bus_clk_s
 PORT CLKFX = dcr_bus_clk_proc_s
 PORT RST = net_gnd
 PORT LOCKED = dcm_1_lock
END

BEGIN top_OXCcontrol
 PARAMETER INSTANCE = top_OXCcontrol_0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE SDCR = ppc_dcr
 PORT DCR_RESET_BUS = net_gnd
 PORT DCR_CLOCK_BUS = dcr_bus_clk_proc_s
 PORT oxc_switch = top_OXCcontrol_0_oxc_switch
END

BEGIN dcr_v29
 PARAMETER INSTANCE = ppc_dcr
 PARAMETER HW_VER = 1.00.a
END

BEGIN opb_timer
 PARAMETER INSTANCE = opb_timer_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_ONE_TIMER_ONLY = 1
 PARAMETER C_BASEADDR = 0xF0000200
 PARAMETER C_HIGHADDR = 0xF00002FF
 BUS_INTERFACE SOPB = ppc_opb
 PORT Interrupt = opb_timer_1_Interrupt
END
```

### *G.1.3.3. Fitxer system.ucf*

```
##############################################################
# Clock Period Constraints
##############################################################
NET "sys_clk" PERIOD = 10.00;

NET gmii_rx_clk TNM_NET = clk_rx;
TIMESPEC TS_rx_clk = PERIOD clk_rx 8000 ps HIGH 50 %;
NET gtx_clk_int TNM_NET = clk_tx;
TIMESPEC TS_tx_clk = PERIOD clk_tx 8000 ps HIGH 50 %;
TIMEGRP "tx_clock_grp" = "clk_tx";
INST *FLOW?RX_PAUSE* TNM = FFS flow_grp;
TIMESPEC "TSflow" = FROM flow_grp TO tx_clock_grp 8000 ps;


##################################################
# I/O Signals
```

```
##################################################
NET "sys_clk"           LOC = F15;
NET "system_reset"      LOC = AH8;

#COM2 RS232 Interface
NET "rx"    LOC = "AK3";
NET "tx"    LOC = "AJ3";

#Ethernet
NET "clk_125_p"         LOC = "F16";
NET "clk_125_n"         LOC = "G16";

NET "gmii_tx_clk"       LOC = "D16";
NET "gmii_rx_clk"       LOC = "B16";
NET "gmii_crs"          LOC = "C23";
NET "gmii_rx_dv"        LOC = "E20";
NET "gmii_rxd<0>"       LOC = "D20";
NET "gmii_rxd<1>"       LOC = "H22";
NET "gmii_rxd<2>"       LOC = "H20";
NET "gmii_rxd<3>"       LOC = "G21";
NET "gmii_rxd<4>"       LOC = "G20";
NET "gmii_rxd<5>"       LOC = "G19";
NET "gmii_rxd<6>"       LOC = "E17";
NET "gmii_rxd<7>"       LOC = "E21";
NET "gmii_col"          LOC = "B23";
NET "gmii_rx_er"        LOC = "D23";
NET "gmii_tx_en"        LOC = "G23";
NET "gmii_tx_er"        LOC = "G22";
NET "gmii_txd<0>"       LOC = "E24";
NET "gmii_txd<1>"       LOC = "D24";
NET "gmii_txd<2>"       LOC = "C24";
NET "gmii_txd<3>"       LOC = "F23";
NET "gmii_txd<4>"       LOC = "E23";
NET "gmii_txd<5>"       LOC = "E22";
NET "gmii_txd<6>"       LOC = "F22";
NET "gmii_txd<7>"       LOC = "D21";
NET "PHY_Mii_clk"       LOC = "F24";
NET "gmii_rst_n"        LOC = "H19";
NET "PHY_Mii_data"      LOC = "E25";


#### Module SRAM constraints

Net fpga_0_SRAM_Mem_A_pin<29> LOC=AH24;
Net fpga_0_SRAM_Mem_A_pin<28> LOC=AG24;
Net fpga_0_SRAM_Mem_A_pin<27> LOC=AE22;
Net fpga_0_SRAM_Mem_A_pin<26> LOC=AC22;
Net fpga_0_SRAM_Mem_A_pin<25> LOC=AG21;
Net fpga_0_SRAM_Mem_A_pin<24> LOC=AJ22;
Net fpga_0_SRAM_Mem_A_pin<23> LOC=AD19;
Net fpga_0_SRAM_Mem_A_pin<22> LOC=AH22;
Net fpga_0_SRAM_Mem_A_pin<21> LOC=AC20;
Net fpga_0_SRAM_Mem_A_pin<20> LOC=AC19;
Net fpga_0_SRAM_Mem_A_pin<19> LOC=AF17;
Net fpga_0_SRAM_Mem_A_pin<18> LOC=AD20;
Net fpga_0_SRAM_Mem_A_pin<17> LOC=AE21;
Net fpga_0_SRAM_Mem_A_pin<16> LOC=AE23;
Net fpga_0_SRAM_Mem_A_pin<15> LOC=AG17;
Net fpga_0_SRAM_Mem_A_pin<14> LOC=AC21;
Net fpga_0_SRAM_Mem_A_pin<13> LOC=AF24;
Net fpga_0_SRAM_Mem_A_pin<12> LOC=AK23;
```

```
Net fpga_0_SRAM_Mem_A_pin<11> LOC=AJ23;
Net fpga_0_SRAM_Mem_A_pin<10> LOC=AF23;
Net fpga_0_SRAM_Mem_A_pin<9> LOC=AF22;
Net fpga_0_SRAM_Mem_A_pin<8> LOC=AD22;
Net fpga_0_SRAM_Mem_DQ_pin<31> LOC=AF10;
Net fpga_0_SRAM_Mem_DQ_pin<30> LOC=AE10;
Net fpga_0_SRAM_Mem_DQ_pin<29> LOC=AC12;
Net fpga_0_SRAM_Mem_DQ_pin<28> LOC=AG14;
Net fpga_0_SRAM_Mem_DQ_pin<27> LOC=AC15;
Net fpga_0_SRAM_Mem_DQ_pin<26> LOC=AB15;
Net fpga_0_SRAM_Mem_DQ_pin<25> LOC=AD17;
Net fpga_0_SRAM_Mem_DQ_pin<24> LOC=AE17;
Net fpga_0_SRAM_Mem_DQ_pin<23> LOC=AD12;
Net fpga_0_SRAM_Mem_DQ_pin<22> LOC=AD10;
Net fpga_0_SRAM_Mem_DQ_pin<21> LOC=AF14;
Net fpga_0_SRAM_Mem_DQ_pin<20> LOC=AE14;
Net fpga_0_SRAM_Mem_DQ_pin<19> LOC=AC10;
Net fpga_0_SRAM_Mem_DQ_pin<18> LOC=AD14;
Net fpga_0_SRAM_Mem_DQ_pin<17> LOC=AB16;
Net fpga_0_SRAM_Mem_DQ_pin<16> LOC=AC16;
Net fpga_0_SRAM_Mem_DQ_pin<15> LOC=AK8;
Net fpga_0_SRAM_Mem_DQ_pin<14> LOC=AH9;
Net fpga_0_SRAM_Mem_DQ_pin<13> LOC=AG7;
Net fpga_0_SRAM_Mem_DQ_pin<12> LOC=AF9;
Net fpga_0_SRAM_Mem_DQ_pin<11> LOC=AJ9;
Net fpga_0_SRAM_Mem_DQ_pin<10> LOC=AD9;
Net fpga_0_SRAM_Mem_DQ_pin<9> LOC=AE9;
Net fpga_0_SRAM_Mem_DQ_pin<8> LOC=AG10;
Net fpga_0_SRAM_Mem_DQ_pin<7> LOC=AH7;
Net fpga_0_SRAM_Mem_DQ_pin<6> LOC=AE7;
Net fpga_0_SRAM_Mem_DQ_pin<5> LOC=AJ8;
Net fpga_0_SRAM_Mem_DQ_pin<4> LOC=AF8;
Net fpga_0_SRAM_Mem_DQ_pin<3> LOC=AE8;
Net fpga_0_SRAM_Mem_DQ_pin<2> LOC=AD11;
Net fpga_0_SRAM_Mem_DQ_pin<1> LOC=AC11;
Net fpga_0_SRAM_Mem_DQ_pin<0> LOC=AC9;
Net fpga_0_SRAM_Mem_BEN_pin<3> LOC=AE15;
Net fpga_0_SRAM_Mem_BEN_pin<2> LOC=AH15;
Net fpga_0_SRAM_Mem_BEN_pin<1> LOC=AF16;
Net fpga_0_SRAM_Mem_BEN_pin<0> LOC=AJ15;
Net fpga_0_SRAM_Mem_WEN_pin LOC=AG25;
Net fpga_0_SRAM_Mem_OEN_pin<0> LOC=AH23;
Net fpga_0_SRAM_Mem_CEN_pin<0> LOC=AG16;
Net fpga_0_SRAM_Mem_RPN_pin LOC=C5;
Net fpga_0_SRAM_emc_disable_flash_pin LOC=AD15;
Net fpga_0_SRAM_emc_disable_sdram_pin LOC=AE16;
Net fpga_0_SRAM_emc_disable_sysace_pin LOC=AD16;
Net fpga_0_SRAM_emc_disable_buffer_pin LOC=AG5;

# Bridge to the Optical Switch
Net top_OXCcontrol_0_oxc_switch_pin LOC=H5;
```

## G.2. Configuracions interfícies de xarxa

Un dels punts més importants en un sistema que ha d'actuar com a node d'algun tipus de xarxa és la de proporcionar una interfície per a comunicar-se amb la resta de nodes. La placa de desenvolupament d'Avnet disposa de 2 interfícies sèrie HSSDC2, de dos sòcols SFP per a Gigabit Ethernet i una interfície física 10/100/1000 Ethernet (per a més informació veure l'ANNEX D).

A continuació realitzarem una breu descripció de les configuracions necessàries per a unir aquestes interfícies amb un PLB GEMAC de Xilinx [22].

### G.2.1. Configuració del SFP

La placa de desenvolupament FPGA conté dos sòcols SFP que poden utilitzar-se com a interfícies Gigabit Ethernet, InfiniBand o Fibre Channel. En el present apartat comentarem la seva configuració per a Gigabit Ethernet, unint el port físic a un RocketIO del xip FPGA.

Els MGTs (RocketIOs) de la FPGA Virtex-II Pro estan connectats a dos connectors HSSDC2, dos connectors SFP amb encapsulats EMI. En el cas dels connectors HSSDC2 i SFP, els MGTs es tracten de forma individual.

En el cas concret dels SFP, aquests es poden utilitzar per a implementar algun protocol sèrie d'alta velocitat sobre fibra òptica o coure, depenent del mòdul utilitzat. Aquests connectors suporten taxes de fins a 2,5 Gbps. Els connectors SFP estan etiquetats en la placa com "P4" i "P5". Els *pin-outs* es mostren en la següent Taula G.2.

**Taula G.2.** *Pin-outs* del SFP i la FPGA.

| SFP | Etiqueta | Senyal | Port FPGA |
|---|---|---|---|
| SFP#1 | P4 | TX_FAULT | F8 |
| | | LOS | F7 |
| | | RX_N | A4 |
| | | RX_P | A5 |
| | | TX_P | A6 |
| | | TX_N | A7 |
| SFP#2 | P5 | TX_FAULT | E9 |
| | | LOS | E8 |
| | | RX_N | A11 |
| | | RX_P | A12 |
| | | TX_P | A13 |
| | | TX_N | A14 |

Els SFPs tenen també *pins* de *Transmit Disable*. Aquests estan connectats a dos *jumpers*. Per defecte, aquests *pins* estan en estat *high* per a deshabilitar la transmissió. Per a habilitar la transmissió a través dels SFPs s'han de posar dos *jumpers*, els etiquetats com JP2 i JP3 en la placa.

Per a implementar algun protocol sèrie d'alta velocitat a 2,5 Gbps s'ha d'utilitzar una entrada de rellotge diferencial de 125 MHz com a rellotge de referència en el MGT. Aquest rellotge s'importa a partir d'un oscil·lador de la placa a un gestor de rellotges (BREF). Els noms dels *nets* de rellotge són GIGE_CLK_P i GIGE_CLK_N, amb números de *pin* FPGA F16 i G16, respectivament. Com que el PLL del MGT sempre multiplica per un factor de 20, utilitzant un rellotge

de 125 MHz resulta que s'obté una taxa de transmissió de 2,5 Gbps. Per a taxes menors, s'ha de dividir el rellotge utilitzant un DCM (*Digital Clock Manager*), i passar l'entrada en el rellotge de referència del MGT. Aquesta configuració, però, introdueix un cert *jitter*.

Les següents taules mostren la configuració al utilitzar el SFP#2 de la placa. La Taula G.3 mostra la configuració dels ports externs (en aquest cas s'han substituït els noms de gige_clk_p i gige_clk_n, per brefclk_p i brefclk_n, respectivament).

**Taula G.3.** Ports externs connectats a la FPGA.

| Net | Port (LOC) | Direcció |
|---|---|---|
| brefclk_p | F16 | In |
| brefclk_n | G16 | In |
| TXN | A14 | Out |
| TXP | A13 | Out |
| RXN | A11 | In |
| RXP | A12 | In |

En la següent Taula G.4 s'especifica el connexionat dels *nets* de rellotge amb el DCM, que utilitzem en el nostre cas per a passar d'un rellotge diferencial de 125 MHz, a un rellotge de 62,5 MHz (per així tenir una taxa de transmissió de 1,25 Gbps, 1000Base-X).

**Taula G.4.** Connexions al DCM des de rellotge diferencial (cas SFP).

| Port del DCM | Net | Direcció |
|---|---|---|
| bref_clk_p | brefclk_p | In |
| bref_clk_n | brefclk_n | In |
| resetn | sys_resetn | In |
| bref_clk | brefclk | Out |
| user_clk | userclk | Out |
| user_clk2 | userclk2 | Out |
| dcm_lock | gemac_dcm_lock | Out |
| lock_led | gemac_lock_led | Out |

El DCM anterior és una variació dels DCMs proporcionats pels *Intellectual Properties* de Xilinx. El codi HDL d'aquest DCM s'especifica a continuació.

```
library ieee;
use ieee.std_logic_1164.all;
library IEEE;
use IEEE.std_logic_1164.all;

entity dcm_ip is
  port (
            bref_clk_p        : in std_logic;
```

```
            bref_clk_n        : in std_logic;
            resetn            : in std_logic;
            bref_clk    : out std_logic;
            user_clk    : out std_logic;
            user_clk2   : out std_logic;
            dcm_lock    : out std_logic;
            lock_led    : out std_logic
            );
end entity dcm_ip;

architecture IMP of dcm_ip is
component DCM
-- synthesis translate_off

generic (CLK_FEEDBACK :string := "1X";
            CLKDV_DIVIDE : real := 2.0;
            CLKFX_DIVIDE : integer :=1;
            CLKFX_MULTIPLY : integer := 1;
            CLKIN_DIVIDE_BY_2 : boolean := FALSE;
            CLKOUT_PHASE_SHIFT: string := "NONE";
            DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
            DFS_FREQUENCY_MODE: string := "LOW";
            DLL_FREQUENCY_MODE: string := "LOW";
            DSS_MODE: string := "NONE";
            DUTY_CYCLE_CORRECTION : Boolean := TRUE;
            FACTORY_JF : bit_vector := X"C080";
            PHASE_SHIFT: real := 0;
            STARTUP_WAIT :boolean := FALSE);

-- synthesis translate_on

   port (CLKIN          : in std_logic;
         CLKFB      : in std_logic;
         DSSEN      : in std_logic;
         PSINCDEC   : in std_logic;
         PSEN       : in std_logic;
         PSCLK      : in std_logic;
         RST        : in std_logic;
         CLK0       : out std_logic;
         CLK90      : out std_logic;
         CLK180     : out std_logic;
         CLK270     : out std_logic;
         CLK2X      : out std_logic;
         CLK2X180   : out std_logic;
         CLKDV      : out std_logic;
         CLKFX      : out std_logic;
         CLKFX180   : out std_logic;
         LOCKED     : out std_logic;
         PSDONE     : out std_logic;
         STATUS     : out std_logic_vector(7 downto 0)
            );
end component DCM;

  component BUFG
      port (
        O : out std_logic;
        I : in std_logic
      );
  end component;

  component IBUFGDS_LVDS_25
```

```
      port  (
            I, IB : in std_logic;
            O : out std_logic
            ); end component;


  signal clk0_i, clk2x180_i         : std_logic;
  signal clk0_int, clk2x180_int     : std_logic;
  signal bref_clk_int, dcm_lock_int, dcm_rst              :
std_logic;
  signal clk125_int, clk125_i       : std_logic;

begin
      dcm_rst     <= not resetn;
bref_clk_ibufg : IBUFGDS_LVDS_25
      port map    (I    => bref_clk_p,
                   IB   => bref_clk_n,
                   O    => bref_clk_int
                  );


    DCM_I: DCM
      port map    (
                  CLKIN       => bref_clk_int,
                  CLKFB       => clk125_i,
                  DSSEN       => '0',
                  PSINCDEC    => '0',
                  PSEN        => '0',
                  PSCLK       => '0',
                  RST         => dcm_rst,
                  CLK0        => clk125_int,
                  CLK90       => open,
                  CLK180      => clk2x180_int,
                  CLK270      => open,
                  CLK2X       => open,
                  CLK2X180    => open,
                  CLKDV       => clk0_int,
                  CLKFX       => open,
                  CLKFX180    => open,
                  LOCKED      => dcm_lock_int,
                  PSDONE      => open,
                  STATUS      => open
                  );

  BUFG_I : BUFG
      port map (
        O => clk0_i,
        I => clk0_int);


  BUFG_D : BUFG
      port map (
        O => clk125_i,
        I => clk125_int);

  BUFG_N : BUFG
      port map (
        O => clk2x180_i,
        I => clk2x180_int);

      bref_clk    <= clk0_i;
```

```
      user_clk    <= clk0_i;
      user_clk2 <= clk2x180_i;
      dcm_lock    <= dcm_lock_int;
      lock_led    <= not dcm_lock_int;

end architecture IMP;
```

Bàsicament, la modificació del DCM_IP proporciona tres senyals de rellotge: dos a 62,5 MHz (bref_clk i user_clk), i un a 125 MHz (el user_clk2). Posteriorment, en la configuració del PLB GEMAC, s'especifica que utilitzi el de 62,5 MHz. Aquesta elecció es realitza mitjançant la modificació d'un camp en el fitxer de *User Constraints* (fitxer "system.ucf"), i amb la configuració del paràmetre/port REFCLKSEL = net_gnd (com s'especifica en [40]) i del REF_CLK_V_SEL = 0 (com s'especifica en les següents línies de configuració). Els camps de restriccions del fitxer UCF són:

```
NET gemac_dcm_ip/gemac_dcm_ip/clk2x180_int  PERIOD = 8.0;
NET gemac_dcm_ip/gemac_dcm_ip/clk0_int  PERIOD = 16.0;

#------------------------------------------------------------------------
---------
# System Clock Period Constraints
#------------------------------------------------------------------------
---------
INST gemac_dcm_ip/gemac_dcm_ip/dcm_i  CLKIN_PERIOD = 8.0;
INST gemac_dcm_ip/gemac_dcm_ip/dcm_i  DLL_FREQUENCY_MODE = LOW;
INST gemac_dcm_ip/gemac_dcm_ip/dcm_i  DFS_FREQUENCY_MODE = HIGH;

INST  ppc_gemac/ppc_gemac/u0/gpcs_pma_inst/mgt/mgt  LOC  =  GT_X2Y1;   ##
MGT5
INST ppc_gemac/ppc_gemac/u0/gpcs_pma_inst/mgt/mgt REF_CLK_V_SEL = 0;
```

En la següent Fig. G.3 es mostra la lògica de selecció del rellotge en el MGT.

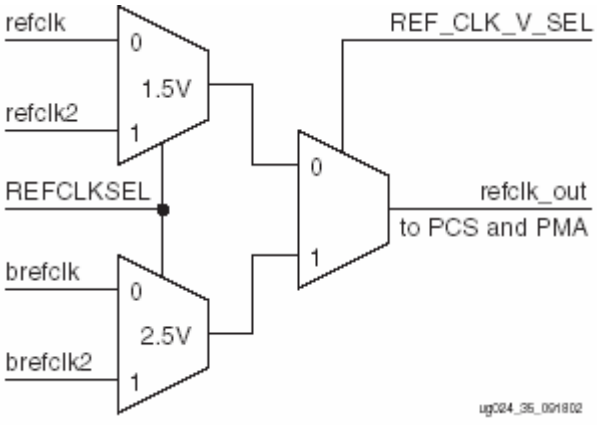



**Fig. G.3.** Lògica de selecció BREFCLK/REFCLK.

La configuració de ports del PLB GEMAC quan en es connecta a un MGT (RocketIO) (sols s'especifiquen els ports que tenen algun tipus de connexió) es mostra en la Taula G.5.

**Taula G.5.** Connexions als ports del PLB GEMAC (cas SFP).

| Port del PLB Gemac | Net | Direcció |
|---|---|---|
| REFCLK | brefclk | In |
| REFCLK2 | net_gnd | In |
| BREFCLK | net_gnd | In |
| BREFCLK2 | net_gnd | In |
| REFCLKSEL | net_gnd | In |
| USERCLK | userclk | In |
| USERCLK2 | userclk2 | In |
| DCM_LOCKED | gemac_dcm_lock | In |
| SIGNAL_DETECT | net_vcc | In |
| TXP | TXP | Out |
| TXN | TXN | Out |
| RXP | RXP | In |
| RXN | RXN | In |
| PLB_Clk | plb_clk | I |

En la següent Taula G.6 s'especifiquen els valors de configuració del PLB GEMAC necessaris per a que funcioni connectat a una interfície sèrie, SERDES (SFP), així com també altres valors utilitzats possibles de configuració.

**Taula G.6.** Configuració del PLB GEMAC (cas SFP).

| Pestanya | Paràmetre | Nom HDL | Valor |
|---|---|---|---|
| User.Misc | Device Block ID | C_DEV_BLK_ID | 1 |
| User.Misc | IPIF Receiver FIFO Depth | C_IPIF_RDFIFO_DEPTH | 131072 |
| User.Misc | IPIF Transmit FIFO Depth | C_IPIF_WRFIFO_DEPTH | 131072 |
| User.Misc | MAX Length and Status FIFO Depth | C_MAC_FIFO_DEPTH | 64 |
| User.PLB/IPIF Interface | Enable MIR Register | C_INCLUDE_DEV_MIR | 0 |
| User.PLB/IPIF Interface | Support Software Reset | C_INCLUDE_RESET | 1 |
| User.PLB/IPIF Interface | Enable Interrupt Device ID Encoder | C_INCLUDE_DEV_PENCODER | 0 |
| User.PLB/IPIF Interface | DMA Mode | C_DMA_TYPE | 1 |
| User.PLB/IPIF Interface | Enable DMA Interrupt Coalescing | C_INCLUDE_DMA_INTR_COALESCE | 0 |

| | Functionality | | |
|---|---|---|---|
| User.GEMAC Features | Support MIIM Interface | C_INCLUDE_MIIM | 0 |
| User.GEMAC Features | MIIM Interface Clock Divider | C_MIIM_CLKDVD | 0b10011 |
| User.GEMAC Features | Enable GMII Interface | C_INCLUDE_GMII | 0 |
| User.GEMAC Features | Enable TBI Interface | C_INCLUDE_TBI | 0 |
| User.GEMAC Features | Enable SERDES | C_INCLUDE_SERDES | 1 |

Finalment, la següent figura Fig. G.4 representa esquemàticament la distribució dels diferents dispositius configurats en les taules anteriors.

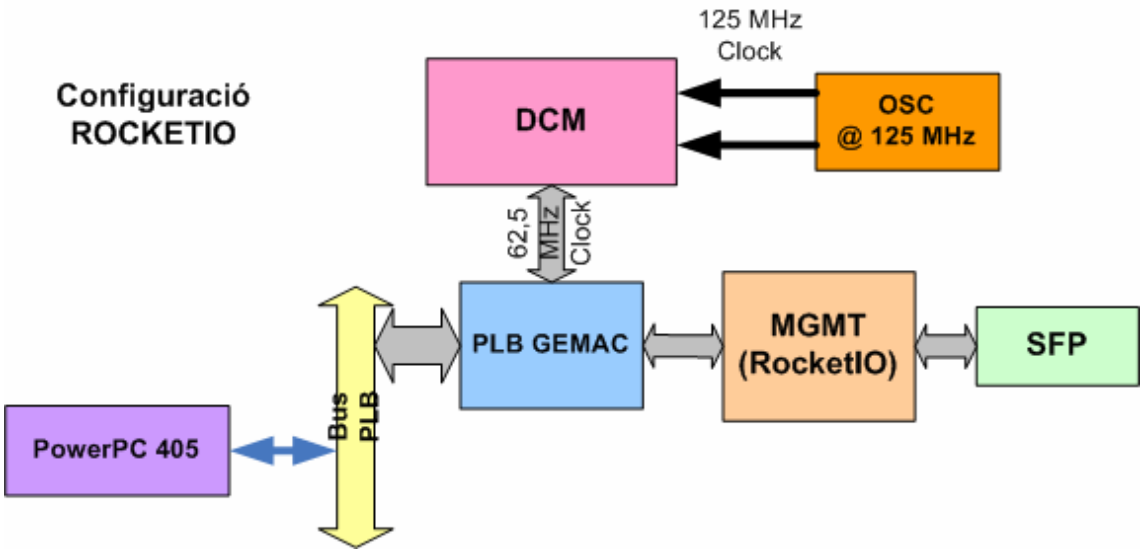



**Fig. G.4.** Diagrama configuració PLB GEMAC (cas SFP).

### G.2.2. Configuració del GMII

En aquest cas, la placa de desenvolupament disposa d'una interfície física 10/100/1000 Mbits/s de National [41]. El PHY està connectat a un connector RJ-45 Pulse. Disposa a més de dos LEDs per a mostrar l'estat de *link* i activitat. Mitjançant lògica externa, la placa disposa també de tres indicadors d'enllaç per a 10, 100 i 1000 Mb/s.

La configuració per defecte de la PHY d'Ethernet són:

- Auto-Negociació habilitat.
- Mode *Full Duplex*
- Advertiment de velocitats 10/100/1000 Mb/s.
- Adreça PHY 0b000001.
- Suport Compatible i No Compatible IEEE.
- Cable Pla amb mode no-MDIX.

- Mode auto-MDIX habilitat.
- Un sol node (NIC).

La configuració física comprèn diferents apartats. Primerament s'ha d'identificar els ports físics externs de la FPGA pels quals s'obtenen els senyals de rellotge externs, i també els que aporten la connectivitat entre la FPGA i el Ethernet PHY. La següent Taula G.7 representa el port extern i el port d'E/S de la FPGA.

**Taula G.7.** Ports externs a la FPGA (cas GMII).

| Net | Port (LOC) | Direcció | Marge |
|---|---|---|---|
| clk_125_p | F16 | In | |
| clk_125_n | G16 | In | |
| gmii_tx_clk | D16 | Out | |
| gmii_rx_clk | B16 | In | |
| gmii_crs | C23 | In | |
| gmii_rx_dv | E20 | In | [7:0] |
| gmii_rxd(<0>-<7>) | D20,H22,H20,G21,G20,G19,E17,E21 | In | |
| gmii_col | B23 | In | |
| gmii_rx_er | D23 | In | |
| gmii_tx_en | G23 | Out | |
| gmii_tx_er | G22 | Out | |
| gmii_txd(<0>-<7>) | E24,D24,C24,F23,E23,E22,F22,D21 | Out | [7:0] |
| PHY_Mii_clk | F24 | In/Out | |
| gmii_rst_n | H19 | Out | |
| PHY_Mii_data | E25 | In/Out | |

Per altra banda, la configuració del mòdul *mii_clock_management* (IP propi d'Avnet) és la que es mostra en la Taula G.8. Els noms dels *Nets* es correspon amb els de la taula anterior.

**Taula G.8.** Configuració de ports del MII_Clock_Management (cas GMII).

| Port del Mòdul | Net | Direcció |
|---|---|---|
| gtx_clk_in_p | clk_125_p | In |
| gtx_clk_in_n | clk_125_n | In |
| rx_clk_in | gmii_rx_clk | In |
| rx_clk_out | gmii_rx_clk_int | Out |
| tx_clk_in | gmii_tx_clk_int | In |
| tx_clk_out | gmii_tx_clk | Out |
| gtx_clk_out | gtx_clk_int | Out |

I la configuració de ports i *Nets* del PLB GEMAC és (només s'especifiquen els *Nets* que tenen connexió) (veure Taula G.9):

**Taula G.9.** Connexions ports del PLB GEMAC (cas GMII).

| Port del PLB Gemac | Net | Direcció | Marge |
|---|---|---|---|
| gtx_clk | gtx_clk_int | I | |
| gmii_col | gmii_col | In | |
| gmii_crs | gmii_crs | In | |
| gmii_txd | gmii_txd | Out | [7:0] |
| gmii_tx_en | gmii_tx_en | Out | |
| gmii_tx_er | gmii_tx_er | Out | |
| gmii_tx_clk | gmii_tx_clk_int | Out | |
| gmii_rxd | gmii_rxd | In | [7:0] |
| gmii_rx_dv | gmii_rx_dv | In | |
| gmii_rx_er | gmii_rx_er | In | |
| gmii_rx_clk | gmii_rx_clk_int | In | |
| gmii_rx_en | No Connection | Out | |
| gmii_rst_n | gmii_rst_n | Out | |
| PHY_Mii_clk | PHY_Mii_clk | In/Out | |
| PHY_Mii_data | PHY_Mii_data | In/Out | |

En la següent Taula G.10 s'especifiquen els valors de configuració del PLB GEMAC necessaris per a que funcioni connectat a una interfície GMII, així com també altres valors utilitzats per a la nostra implementació concreta del Pla de Control en les proves especificades en la memòria del PFC.

**Taula G.10.** Configuració del PLB GEMAC (cas GMII).

| Pestanya | Paràmetre | Nom HDL | Valor |
|---|---|---|---|
| User.Misc | Device Block ID | C_DEV_BLK_ID | 1 |
| User.Misc | IPIF Receiver FIFO Depth | C_IPIF_RDFIFO_DEPTH | 131072 |
| User.Misc | IPIF Transmit FIFO Depth | C_IPIF_WRFIFO_DEPTH | 131072 |
| User.Misc | MAX Length and Status FIFO Depth | C_MAC_FIFO_DEPTH | 64 |
| User.PLB/IPIF Interface | Enable MIR Register | C_INCLUDE_DEV_MIR | 1 |
| User.PLB/IPIF Interface | Support Software Reset | C_INCLUDE_RESET | 1 |
| User.PLB/IPIF Interface | Enable Interrupt Device ID Encoder | C_INCLUDE_DEV_PENCODER | 1 |
| User.PLB/IPIF Interface | DMA Mode | C_DMA_TYPE | 1 |
| User.PLB/IPIF Interface | Enable DMA Interrupt Coalescing Functionality | C_INCLUDE_DMA_INTR_COALESCE | 0 |

| | | | |
|---|---|---|---|
| User.GEMAC Features | Support MIIM Interface | C_INCLUDE_MIIM | 0 |
| User.GEMAC Features | MIIM Interface Clock Divider | C_MIIM_CLKDVD | 0b10011 |
| User.GEMAC Features | Enable GMII Interface | C_INCLUDE_GMII | 1 |
| User.GEMAC Features | Enable TBI Interface | C_INCLUDE_TBI | 0 |
| User.GEMAC Features | Enable SERDES | C_INCLUDE_SERDES | 0 |

Finalment, la següent Fig. G.5 representa esquemàticament la distribució dels diferents dispositius configurats en les taules anteriors.

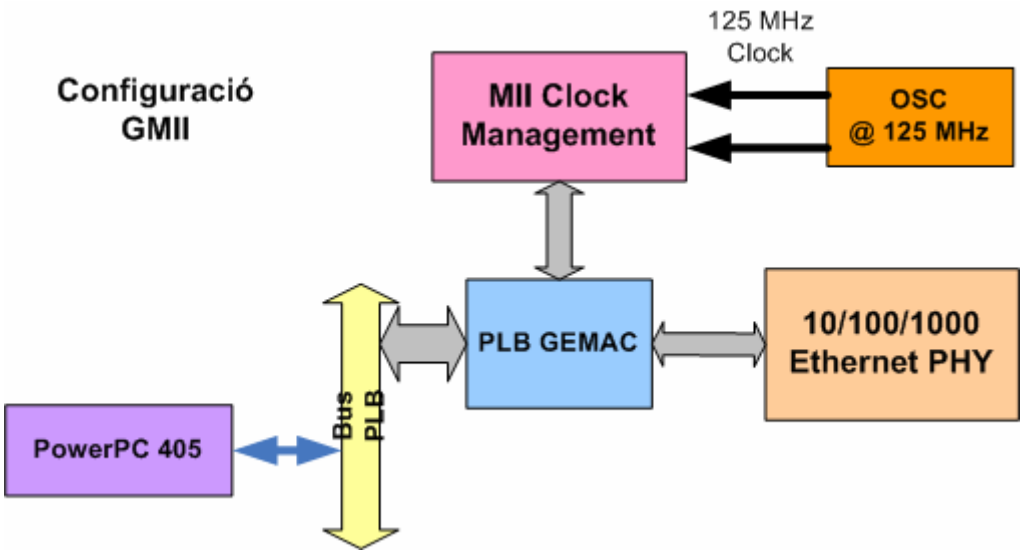



**Fig. G.5.** Diagrama configuració PLB GEMAC (cas GMII).

# ANNEX H. RESULTATS: TAULES I GRÀFIQUES

En aquest annex es presenten els resultats en taules i gràfiques de les diferents proves realitzades en el CAPÍTOL 5.

## H.1. Tests de processament de peticions locals de TX de ràfega

Aquests tests es centren en el processament de les peticions locals de transmissió de ràfega. En aquest cas, el programa de control no processa cap tipus de paquet de control enviat per algun altre node.

### H.1.1. Taules de dades

**Taula H.1.** Temps de procés per funció i totals a diferents taxes i temps de ranura.

| ID Prova | Temps Ranura (µs) | Taxa Peticions (pet/s) | # Clocks Proc. | # Clocks TX Paquet | Temps Proc. (µs) | Temps TX paquet (µs) | Temps Total (µs) |
|---|---|---|---|---|---|---|---|
| 01_1 | 1000 | 10 | 10765 | 12300 | 35,883 | 41,000 | 76,883 |
| 02_1 | 1000 | 100 | 11568 | 12040 | 38,560 | 40,133 | 78,693 |
| 03_1 | 1000 | 200 | 12524 | 12041 | 41,747 | 40,137 | 81,883 |
| 04_1 | 1000 | 400 | 14580 | 12041 | 48,600 | 40,137 | 88,737 |
| 05_1 | 500 | 10 | 11470 | 12303 | 38,233 | 41,010 | 79,243 |
| 06_1 | 500 | 100 | 12326 | 12094 | 41,087 | 49,140 | 90,227 |
| 07_1 | 500 | 200 | 13719 | 12041 | 45,730 | 49,140 | 94,870 |
| 08_1 | 500 | 400 | 14800 | 12041 | 49,333 | 49,140 | 98,473 |
| 09_1 | 100 | 10 | 22287 | 13502 | 74,290 | 49,140 | 123,430 |
| 10_1 | 100 | 100 | 21996 | 13493 | 73,320 | 51,140 | 124,460 |
| 11_1 | 100 | 200 | 21925 | 13630 | 73,083 | 52,140 | 125,223 |
| 12_1 | 100 | 400 | 28052 | 13441 | 93,507 | 53,140 | 146,647 |
| 13_1 | 50 | 10 | 34515 | 15335 | 115,050 | 53,140 | 168,190 |
| 14_1 | 50 | 100 | 37929 | 15362 | 126,430 | 53,140 | 179,570 |
| 15_1 | 50 | 200 | 37509 | 15429 | 125,030 | 53,140 | 178,170 |
| 16_1 | 50 | 400 | 46687 | 15917 | 155,623 | 53,140 | 208,763 |

**Taula H.2.** Quantificació de peticions i ranures reservades a diferents taxes i temps de ranura.

| ID Prova | Temps Ranura (µs) | Taxa Pet. (pet/s) | Pet. | Pet. No Reserv. | Pet. Reserv. | Reser. Inicia. | Reser. Finalitz. | Ranu. TX |
|---|---|---|---|---|---|---|---|---|
| 01_1 | 1000 | 10 | 1024 | 0 | 1024 | 1007 | 1007 | 3010 |
| 02_1 | 1000 | 100 | 1024 | 0 | 1024 | 1003 | 1003 | 3008 |
| 03_1 | 1000 | 200 | 1197 | 173 | 1024 | 997 | 997 | 2824 |
| 04_1 | 1000 | 400 | 1983 | 959 | 1024 | 979 | 979 | 2746 |
| 05_1 | 500 | 10 | 1024 | 0 | 1024 | 1019 | 1019 | 6112 |
| 06_1 | 500 | 100 | 1024 | 0 | 1024 | 1011 | 1011 | 6056 |
| 07_1 | 500 | 200 | 1205 | 181 | 1024 | 996 | 996 | 5616 |
| 08_1 | 500 | 400 | 1964 | 940 | 1024 | 975 | 975 | 5872 |
| 09_1 | 100 | 10 | 1024 | 0 | 1024 | 1009 | 1009 | 30240 |
| 10_1 | 100 | 100 | 1024 | 0 | 1024 | 1009 | 1009 | 30260 |
| 11_1 | 100 | 200 | 1024 | 0 | 1024 | 1007 | 1007 | 30240 |
| 12_1 | 100 | 400 | 1960 | 936 | 1024 | 989 | 989 | 38800 |
| 13_1 | 50 | 10 | 1024 | 0 | 1024 | 1010 | 1010 | 60560 |
| 14_1 | 50 | 100 | 1024 | 0 | 1024 | 1003 | 1003 | 60200 |
| 15_1 | 50 | 200 | 1024 | 0 | 1024 | 1005 | 1004 | 60320 |
| 16_1 | 50 | 400 | 1992 | 968 | 1024 | 997 | 997 | 78560 |

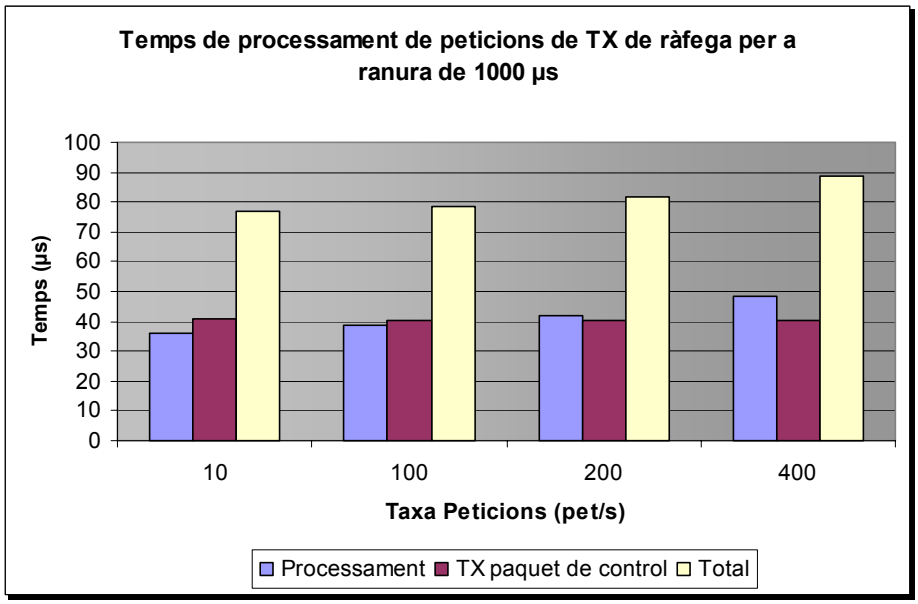
**Taula H.3.** Taxa de reserves i throughput.

| ID Prova | Temps Ranur. (µs) | Taxa Pet. (pet/s) | Bits / ranura | Ranur. / Pet. | Taxa Ràfegues (Mbps) | Taxa Reser. / segon | Taxa Reser. Slots / segon | % Reserva / segon |
|---|---|---|---|---|---|---|---|---|
| 01_1 | 1000 | 10 | 1000000 | 2,99 | 29,891 | 10,00 | 29,9 | 2,99% |
| 02_1 | 1000 | 100 | 1000000 | 3,00 | 299,900 | 100,00 | 299,9 | 29,99% |
| 03_1 | 1000 | 200 | 1000000 | 2,83 | 484,624 | 171,09 | 484,6 | 48,46% |
| 04_1 | 1000 | 400 | 1000000 | 2,80 | 579,369 | 206,56 | 579,4 | 57,94% |
| 05_1 | 500 | 10 | 500000 | 6,00 | 29,990 | 10,00 | 60,0 | 3,00% |
| 06_1 | 500 | 100 | 500000 | 5,99 | 299,505 | 100,00 | 599,0 | 29,95% |
| 07_1 | 500 | 200 | 500000 | 5,64 | 479,160 | 169,96 | 958,3 | 47,92% |
| 08_1 | 500 | 400 | 500000 | 6,02 | 628,015 | 208,55 | 1256,0 | 62,80% |
| 09_1 | 100 | 10 | 100000 | 29,97 | 29,970 | 10,00 | 299,7 | 3,00% |
| 10_1 | 100 | 100 | 100000 | 29,99 | 299,901 | 100,00 | 2999,0 | 29,99% |
| 11_1 | 100 | 200 | 100000 | 30,03 | 600,596 | 200,00 | 6006,0 | 60,06% |
| 12_1 | 100 | 400 | 100000 | 39,23 | 819,859 | 208,98 | 8198,6 | 81,99% |
| 13_1 | 50 | 10 | 50000 | 59,96 | 29,980 | 10,00 | 599,6 | 3,00% |
| 14_1 | 50 | 100 | 50000 | 60,02 | 300,100 | 100,00 | 6002,0 | 30,01% |
| 15_1 | 50 | 200 | 50000 | 60,08 | 600,797 | 200,00 | 12015,9 | 60,08% |
| 16_1 | 50 | 400 | 50000 | 78,80 | 810,115 | 205,62 | 16202,3 | 81,01% |

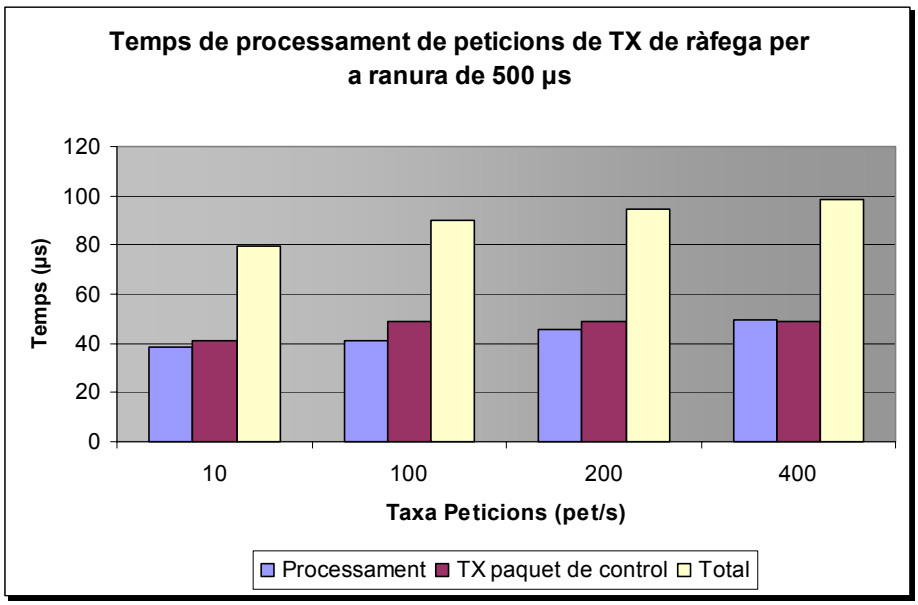
## H.1.2. Gràfiques de temps de processament

En les següents quatre gràfiques (veure Graf H.1, Graf H.2, Graf H.3 i Graf H.4) es pot apreciar l'evolució del temps de procés al decrementar el temps de ranura des dels 1000 µs als 50 µs. Com es pot veure, per a cada cas concret de gràfica, el temps augmenta al augmentar la taxa de peticions. De la mateixa forma, al disminuir el temps de ranura també va augmentant el temps de procés per a un mateix valor de taxa de peticions.

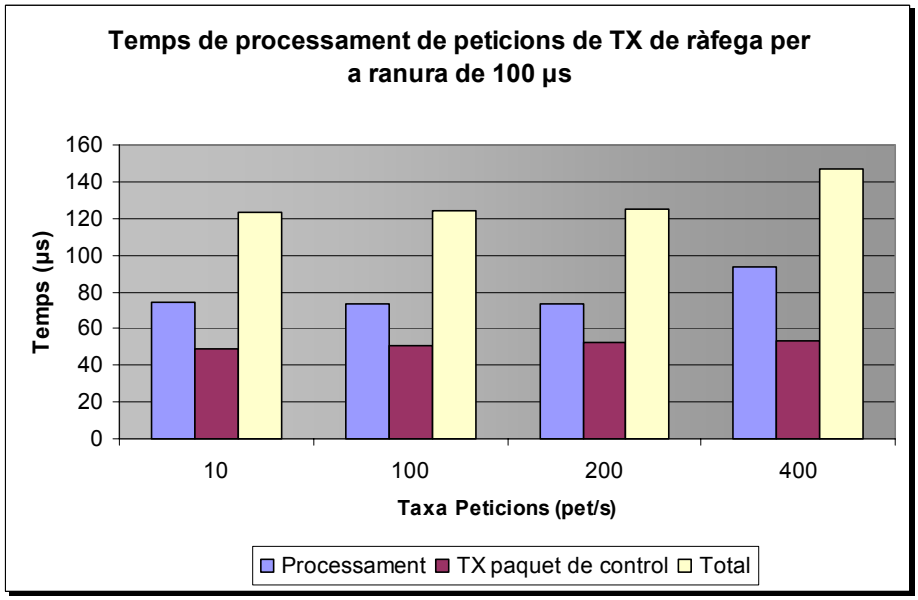
Així mateix, també s'observa com el temps de TX del paquet de control es manté gairebé estable, tant al disminuir el temps de ranura, com al augmentar la taxa de peticions.



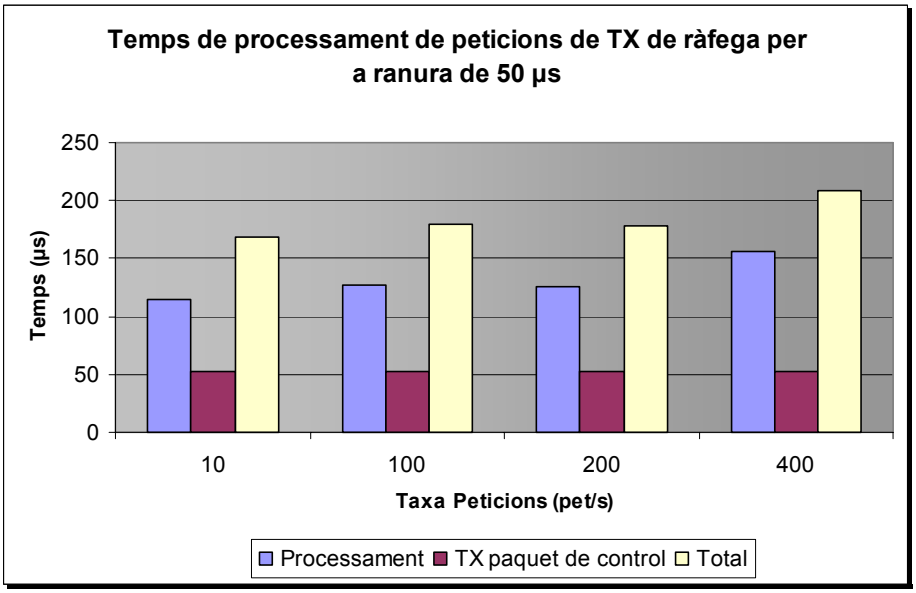

**Graf H.1.** Temps de processament de peticions de TX de ràfega per a ranura de 1000 µs.




**Graf H.2.** Temps de processament de peticions de TX de ràfega per a ranura de 500 µs.

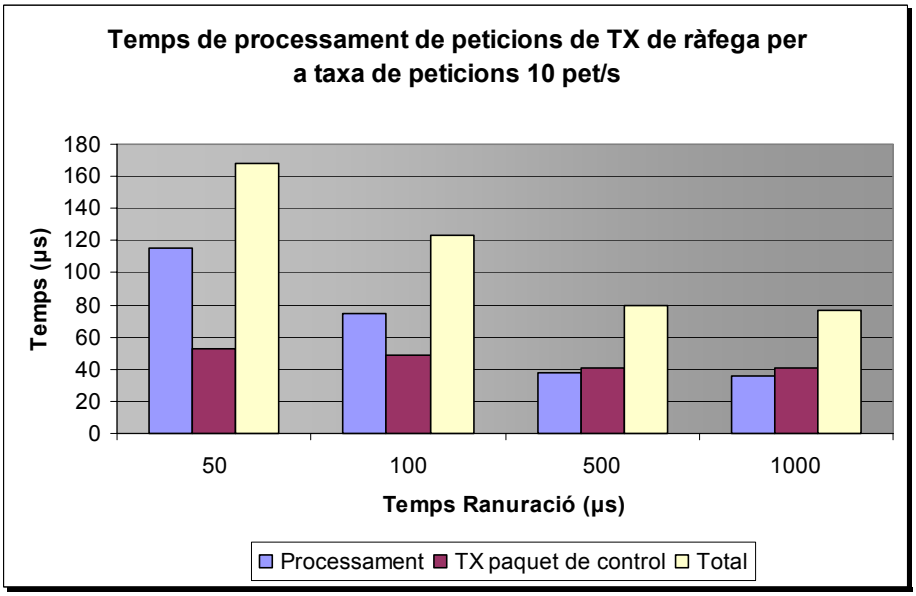



**Graf H.3.** Temps de processament de peticions de TX de ràfega per a ranura de 100 µs.

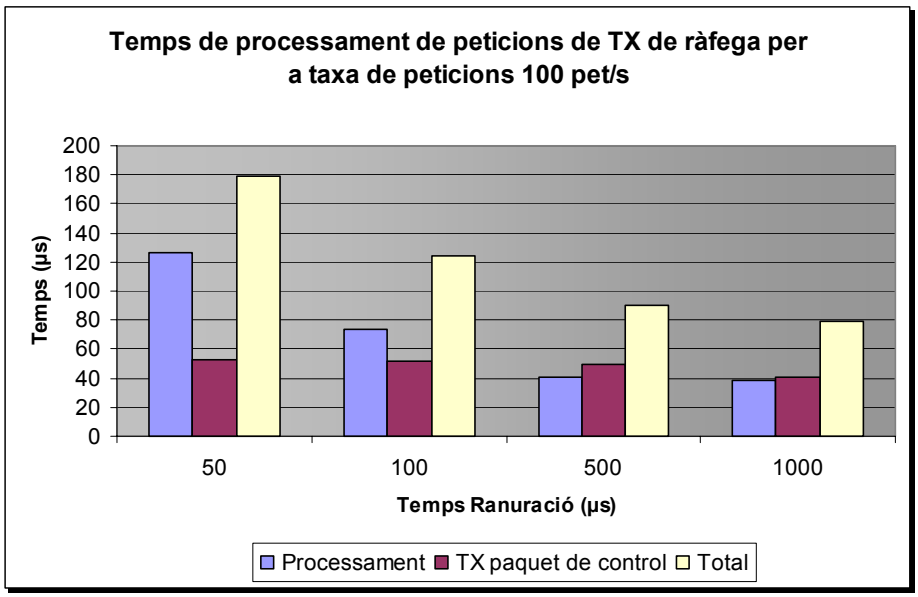



**Graf H.4.** Temps de processament de peticions de TX de ràfega per a ranura de 50 µs.

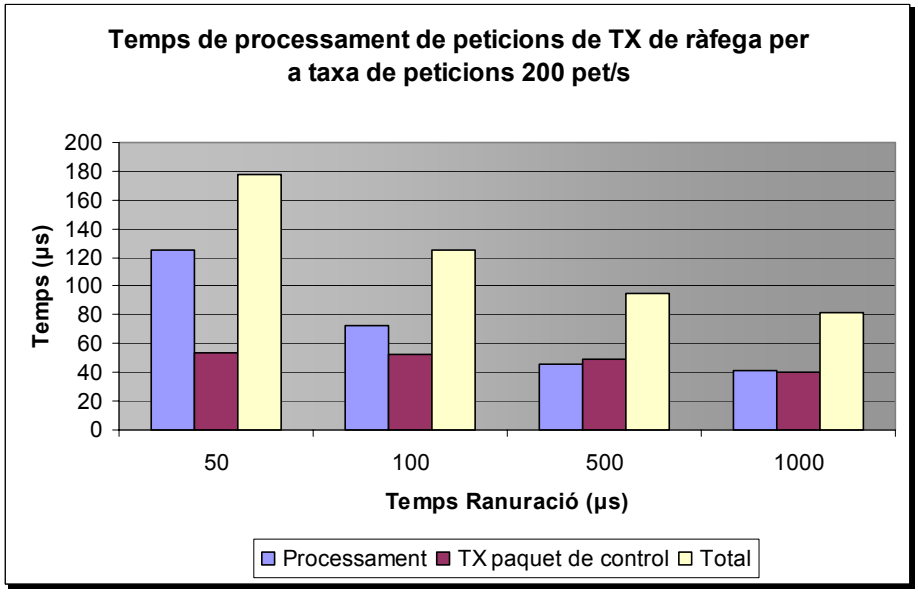
Per altra banda, les següents quatre gràfiques (veure Graf H.5, Graf H.6, Graf H.1 i Graf H.1) proporcionen l'evolució del temps de procés al incrementar la taxa de peticions. Com ja s'ha introduït anteriorment, per a un valor definit de taxa de peticions, al augmentar el temps de ranura, el temps de procés disminueix, principalment per a la funció de "Processament".



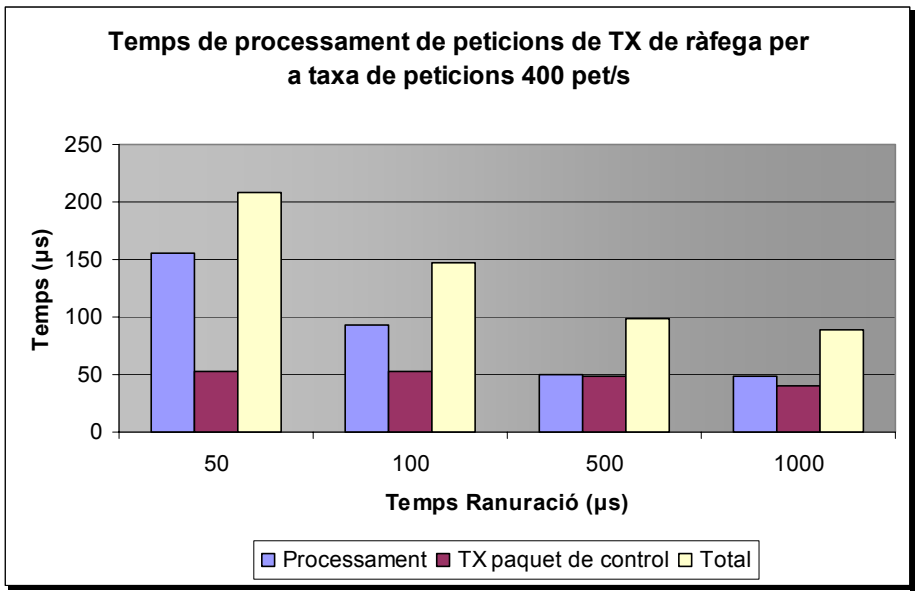

**Graf H.5.** Temps de processament de peticions de TX de ràfega per a una taxa de peticions de 10 pet/s.




**Graf H.6.** Temps de processament de peticions de TX de ràfega per a una taxa de peticions de 100 pet/s.



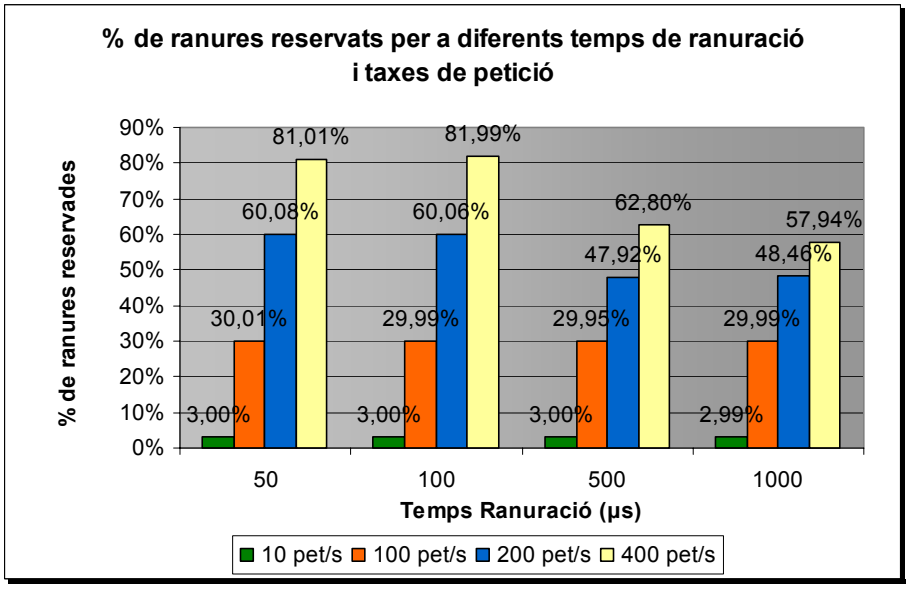

**Graf H.7.** Temps de processament de peticions de TX de ràfega per a una taxa de peticions de 200 pet/s.



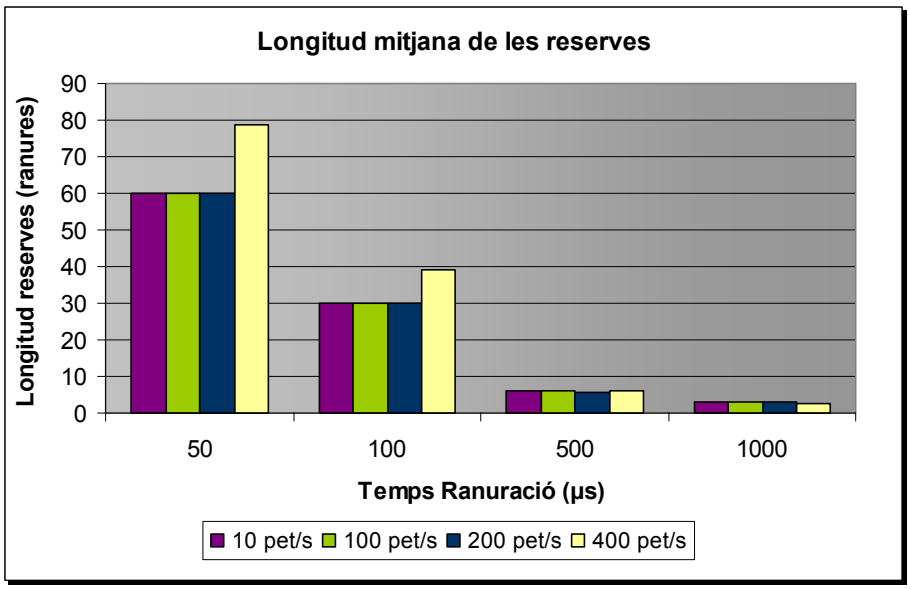

**Graf H.8.** Temps de processament de peticions de TX de ràfega per a una taxa de peticions de 400 pet/s.

## H.1.3. Gràfiques de reserva de ranures

Les dues següents gràfiques (veure Graf H.8 i Graf H.10) estan comentades en el cos de la memòria (veure apartat 5.2.2).




**Graf H.9.** % de ranures reservades per a diferents temps de ranura i taxes de petició.




**Graf H.10.** Longitud mitjana de les reserves per a diferents temps de ranura i taxes de petició.

## H.2. Tests de processament de paquets de control de Setup

En aquests tests el programa de control sols processava peticions de reserva de recursos contingudes en paquets de control de tipus *Setup* enviats pel programa client.

### H.2.1. Taules de dades

**Taula H.4.** Temps de procés per funció a diferents taxes i temps de ranura.

| ID Prova | Temps Ranura (µs) | Taxa Client (pet/s) | # Clocks Proc | Temps Proc. (µs) |
|---|---|---|---|---|
| C02_1 | 1000 | 100 | 10750 | 35,833 |
| C03_1 | 1000 | 200 | 12123 | 40,410 |
| C04_1 | 1000 | 500 | 15750 | 52,500 |
| C05_1 | 1000 | 1000 | 20190 | 67,300 |
| C06_1 | 500 | 100 | 10685 | 35,617 |
| C07_1 | 500 | 200 | 12300 | 41,000 |
| C08_1 | 500 | 500 | 14932 | 49,773 |
| C09_1 | 500 | 1000 | 17033 | 56,777 |
| C10_1 | 100 | 100 | 10130 | 33,767 |
| C11_1 | 100 | 200 | 9874 | 32,913 |
| C12_1 | 100 | 500 | 10879 | 36,263 |
| C13_1 | 100 | 1000 | 11722 | 39,073 |
| C15_1 | 50 | 100 | 11502 | 38,340 |
| C16_1 | 50 | 200 | 11141 | 37,137 |
| C17_1 | 50 | 500 | 12489 | 41,630 |
| C18_1 | 50 | 900 | 11994 | 39,980 |

**Taula H.5.** Quantificació de peticions i ranures reservades a diferents taxes i temps de ranura.

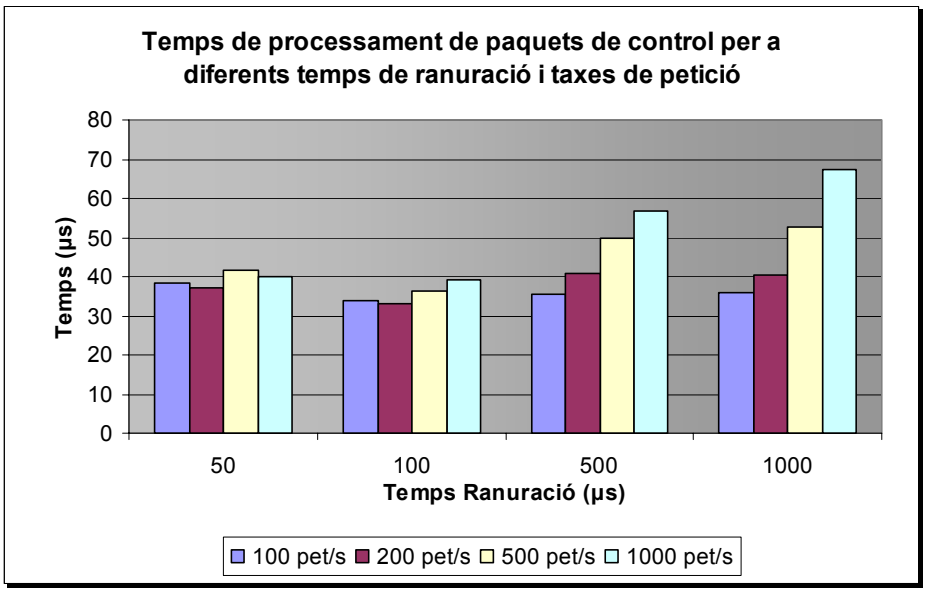
| ID Prova | Temps Ranura (µs) | Taxa Client (pet/s) | Setups Rebuts | Paq. Des-car-tats | Setups No Reserv. | Setups Reserv. | Reser. Inicia. | Reser. Finalit. | Ranu. RX |
|---|---|---|---|---|---|---|---|---|---|
| C02_1 | 1000 | 100 | 1522 | 0 | 498 | 1024 | 949 | 950 | 3910 |
| C03_1 | 1000 | 200 | 2088 | 0 | 1064 | 1024 | 920 | 919 | 3576 |
| C04_1 | 1000 | 500 | 3616 | 6 | 2592 | 1024 | 876 | 876 | 3137 |
| C05_1 | 1000 | 1000 | 5761 | 29 | 4737 | 1024 | 797 | 796 | 2776 |
| C06_1 | 500 | 100 | 1249 | 0 | 225 | 1024 | 959 | 959 | 4165 |
| C07_1 | 500 | 200 | 1553 | 0 | 529 | 1024 | 948 | 948 | 3885 |
| C08_1 | 500 | 500 | 2355 | 0 | 1331 | 1024 | 896 | 896 | 3385 |
| C09_1 | 500 | 1000 | 3542 | 0 | 2518 | 1024 | 855 | 857 | 3054 |
| C10_1 | 100 | 100 | 1040 | 0 | 16 | 1024 | 977 | 977 | 4303 |
| C11_1 | 100 | 200 | 1103 | 0 | 79 | 1024 | 972 | 972 | 4230 |
| C12_1 | 100 | 500 | 1283 | 6 | 259 | 1024 | 965 | 965 | 4063 |
| C13_1 | 100 | 1000 | 1555 | 24 | 531 | 1024 | 952 | 952 | 3941 |
| C15_1 | 50 | 100 | 1025 | 0 | 1 | 1024 | 980 | 980 | 4302 |
| C16_1 | 50 | 200 | 1041 | 0 | 17 | 1024 | 975 | 976 | 4279 |
| C17_1 | 50 | 500 | 1140 | 6 | 116 | 1024 | 971 | 972 | 4271 |
| C18_1 | 50 | 900 | 1325 | 24 | 301 | 1024 | 969 | 969 | 4052 |

**Taula H.6.** Taxa de reserva i throghput.

| ID Prova | Temps Ran. (µs) | Taxa Client (pet/s) | Bits / ranura | Ran. / Pet. | Taxa Ràfe. (Mbps) | Taxa Reser. | Taxa Reser. Ran. | % Reserva / segon |
|---|---|---|---|---|---|---|---|---|
| C02_1 | 1000 | 100 | 1000000 | 4,12 | 276,910 | 67,28 | 276,9 | 27,69% |
| C03_1 | 1000 | 200 | 1000000 | 3,89 | 381,664 | 98,08 | 381,7 | 38,17% |
| C04_1 | 1000 | 500 | 1000000 | 3,58 | 507,051 | 141,59 | 507,1 | 50,71% |
| C05_1 | 1000 | 1000 | 1000000 | 3,49 | 619,881 | 177,75 | 619,9 | 61,99% |
| C06_1 | 500 | 100 | 500000 | 4,34 | 178,034 | 81,99 | 356,1 | 17,80% |
| C07_1 | 500 | 200 | 500000 | 4,10 | 270,216 | 131,87 | 540,4 | 27,02% |
| C08_1 | 500 | 500 | 500000 | 3,78 | 410,676 | 217,41 | 821,4 | 41,07% |
| C09_1 | 500 | 1000 | 500000 | 3,56 | 515,121 | 289,10 | 1030,2 | 51,51% |
| C10_1 | 100 | 100 | 100000 | 4,40 | 43,365 | 98,46 | 433,7 | 4,34% |
| C11_1 | 100 | 200 | 100000 | 4,35 | 80,803 | 185,68 | 808,0 | 8,08% |
| C12_1 | 100 | 500 | 100000 | 4,21 | 168,021 | 399,06 | 1680,2 | 16,80% |
| C13_1 | 100 | 1000 | 100000 | 4,14 | 272,608 | 658,52 | 2726,1 | 27,26% |
| C15_1 | 50 | 100 | 50000 | 4,39 | 21,928 | 99,90 | 438,6 | 2,19% |
| C16_1 | 50 | 200 | 50000 | 4,38 | 43,126 | 196,73 | 862,5 | 4,31% |
| C17_1 | 50 | 500 | 50000 | 4,39 | 98,673 | 449,12 | 1973,5 | 9,87% |
| C18_1 | 50 | 900 | 50000 | 4,18 | 161,585 | 772,83 | 3231,7 | 16,16% |

## H.2.2. Gràfiques de temps de processament

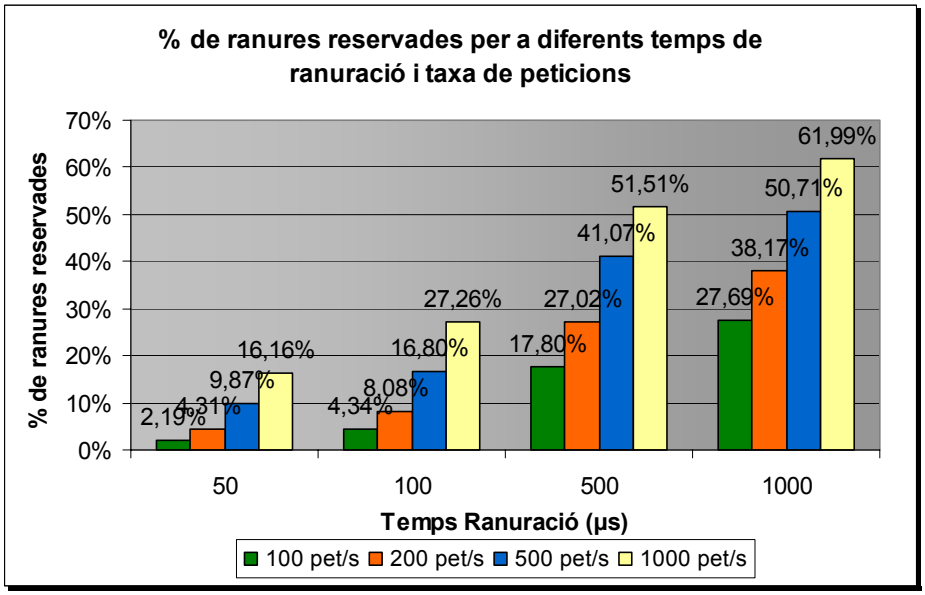
La següent gràfica (Graf H.11) està comentada en l'apartat 5.2.3 del cos de la memòria.



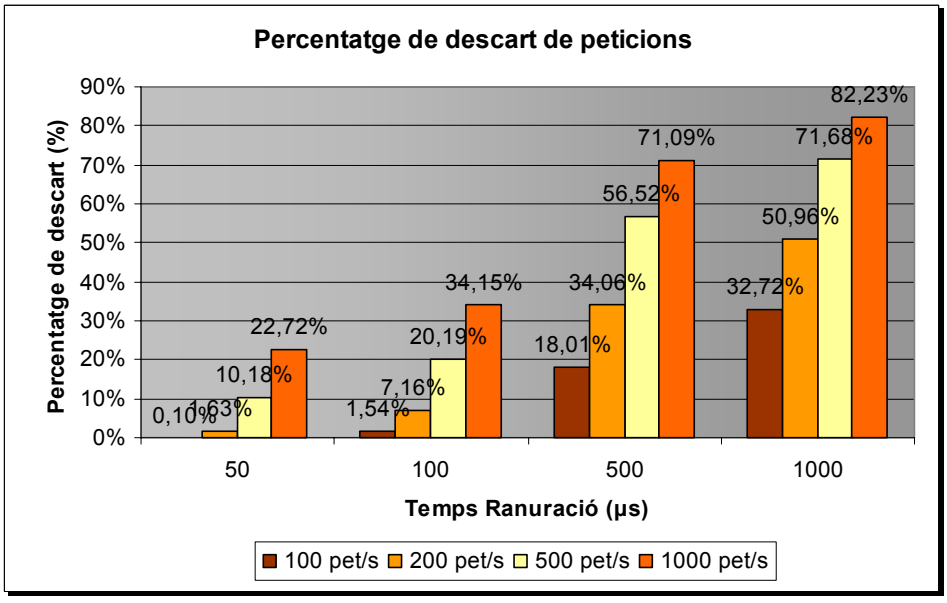

**Graf H.11.** Temps de processament de paquets de control enviats pel programa client per a diferents temps de ranura i taxes de paquet.

## H.2.3. Gràfiques de reserva de ranures

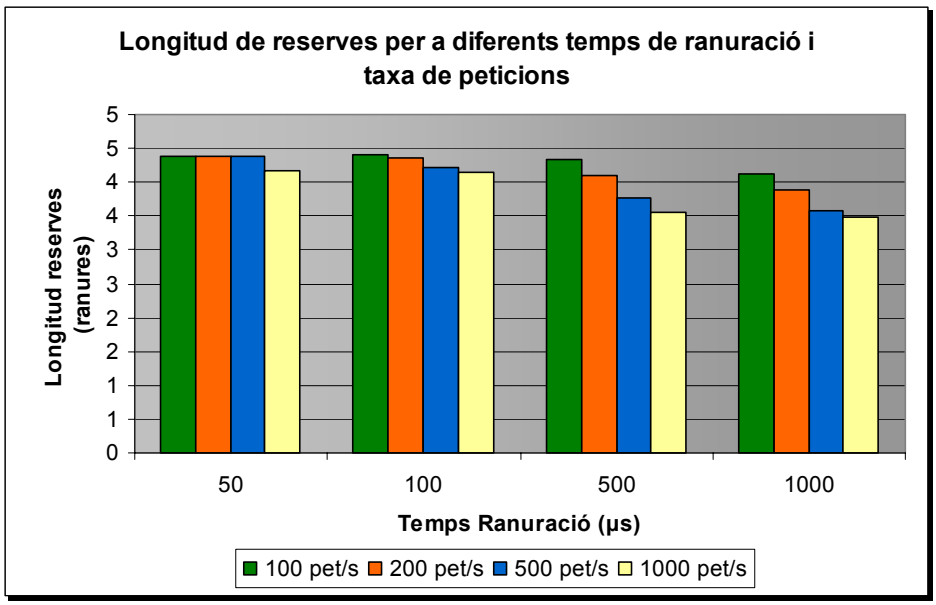
Les gràfiques Graf H.12 i Graf H.13 estàn comentades en el cos de la memoria, apartat 5.2.3. Pel que fa a la longitud de les reserves, el resultat de la seva evolució per a diferents temps de ranura i taxes de petició es pot apreciar en la gràfica Graf H.14. Es pot comprovar com per a un temps de ranura determina, al incrementar la taxa de peticions, la longitud mitjana de reserva disminueix. Aquest valor està correlat amb els de la gràfica Graf H.13, on per a taxes més elevades, la probabilitat de descart augmenta. Per tant, la raó que la longitud de ranura disminueixi és perquè es descarten més peticions que sol·liciten reserves grans enfront de les que sol·liciten reserves menors. Per altra banda, al mantenir la taxa de peticions i incrementar el temps de ranura, la longitud mitjana de reserva també disminueix.




**Graf H.12.** % de ranures reservades per a diferents temps de ranura i taxa de peticions.



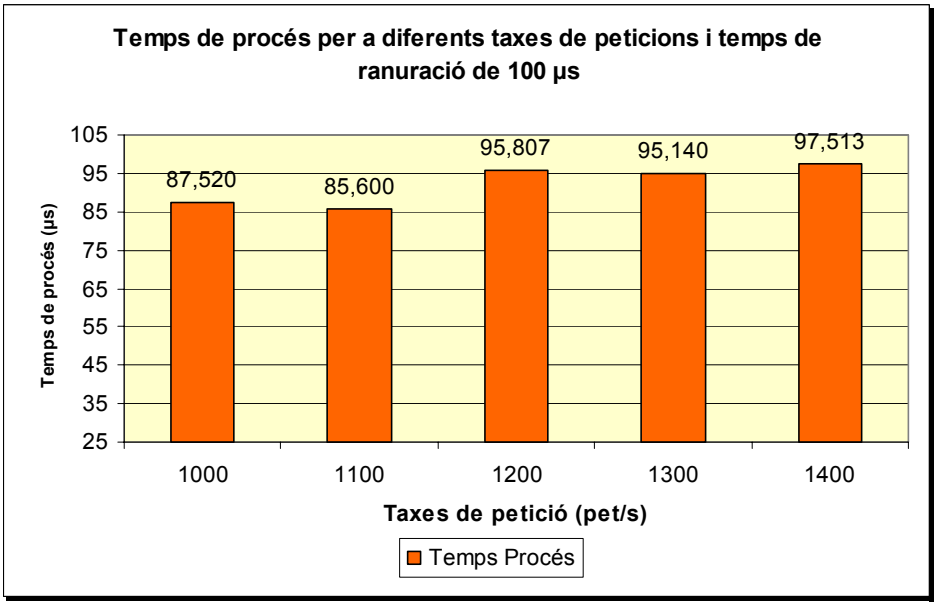

**Graf H.13.** Percentatge de descart de peticions per a diferents temps de ranura i taxes de petició.




**Graf H.14.** Longituds de reserva per a diferents temps de ranura i taxes de petició.

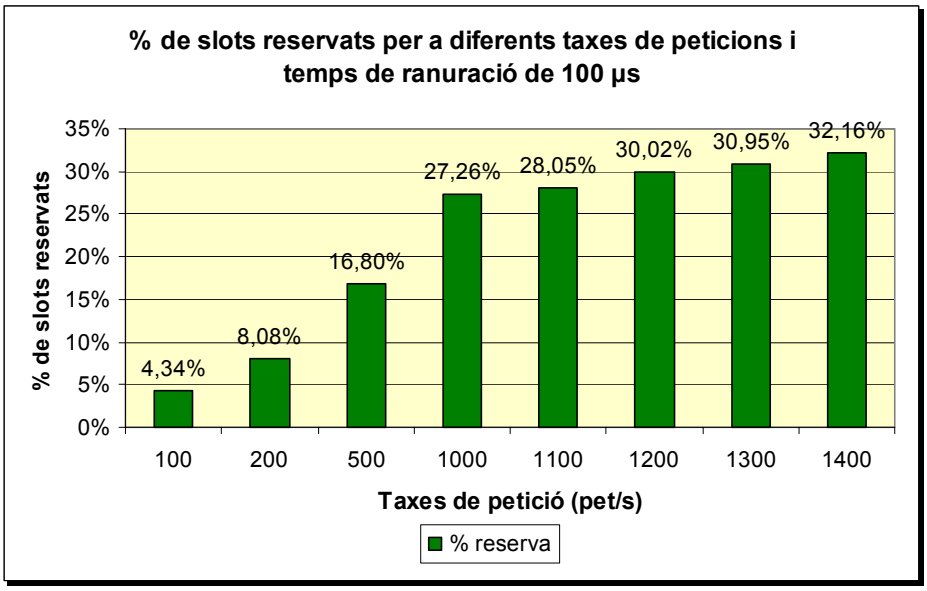
## H.2.4. Comparativa de ranures de 100 µs.

En les següents gràfiques es mostra l'evolució del temps de procés i la taxa de reserva per a un temps de ranura de 100 µs i taxes de petició elevades (més de 1000 pet/s). Com ja s'ha explicat en 5.2.3, al augmentar la taxa de peticions, el temps de procés augmenta. Així i tot s'observa com a taxes extremadament altes, més de 1200 pet/s, el temps tendeix a estabilitzar-se (potser aquest comportament no es compleix per a taxes majors a les analitzades).




**Graf H.15.** Temps de procés per a diferentes taxes de petició i temps de ranura de 100 µs.

També per al cas del % de ranures reservades, al incrementar fins a les 1400 pet/s la taxa de peticions, el % de reserva s'incrementa. Aquest comportament és igual a l'explicat en l'apartat 5.2.3 de la memoria.




**Graf H.16.** % de ranures reservades per a diferents taxes de petició i temps de ranura de 100 µs.

## H.3. Tests globals (peticions + paquets de Setup)

En aquests tests, el programa de control realitza tant el processament de peticions locals de transmissió de ràfega, com el processament d'altres peticions contingudes en paquets de *Setup* enviats pel programa client.

### H.3.1. Taules de dades

**Taula H.7.** Temps de procés per funció a diferents taxes i temps de ranura.

| ID Prova | Temps Ranura (µs) | Taxa Client (pet/s) | Taxa Petitcions (pet/s) | # Clocks Proc. Client | # Clocks Proc. Queries | # Clocks TX paquet | Temps Proc. Client (µs) | Temps Proc. Peticions (µs) | Temps TX paquet (µs) |
|---|---|---|---|---|---|---|---|---|---|
| Q01_1 | 100 | 200 | 100 | 18380 | 21829 | 13537 | 61,267 | 72,763 | 45,123 |
| Q02_1 | 100 | 400 | 100 | 19987 | 24703 | 14002 | 66,623 | 82,343 | 46,673 |
| Q03_1 | 100 | 600 | 100 | 22521 | 25958 | 13572 | 75,070 | 86,527 | 45,240 |
| Q04_1 | 100 | 1000 | 100 | 26710 | 36413 | 16718 | 89,033 | 121,377 | 55,727 |
| Q05_1 | 100 | 200 | 200 | 17793 | 21574 | 13562 | 59,310 | 71,913 | 45,207 |
| Q06_1 | 100 | 400 | 200 | 20664 | 24976 | 13975 | 68,880 | 83,253 | 46,583 |
| Q07_1 | 100 | 600 | 200 | 22109 | 27573 | 14175 | 73,697 | 91,910 | 47,250 |
| Q08_1 | 100 | 1000 | 200 | 26171 | 28827 | 13597 | 87,237 | 96,090 | 45,323 |
| Q09_1 | 100 | 200 | 400 | 16504 | 20638 | 13766 | 55,013 | 68,793 | 45,887 |
| Q10_1 | 100 | 400 | 400 | 18470 | 22437 | 14068 | 61,567 | 74,790 | 46,893 |
| Q11_1 | 100 | 600 | 400 | 21907 | 25257 | 13556 | 73,023 | 84,190 | 45,187 |
| Q12_1 | 100 | 1000 | 400 | 23617 | 27731 | 15098 | 78,723 | 92,437 | 50,327 |

**Taula H.8.** Quantificació de peticions i ranures reservades a diferents taxes i temps de ranura.

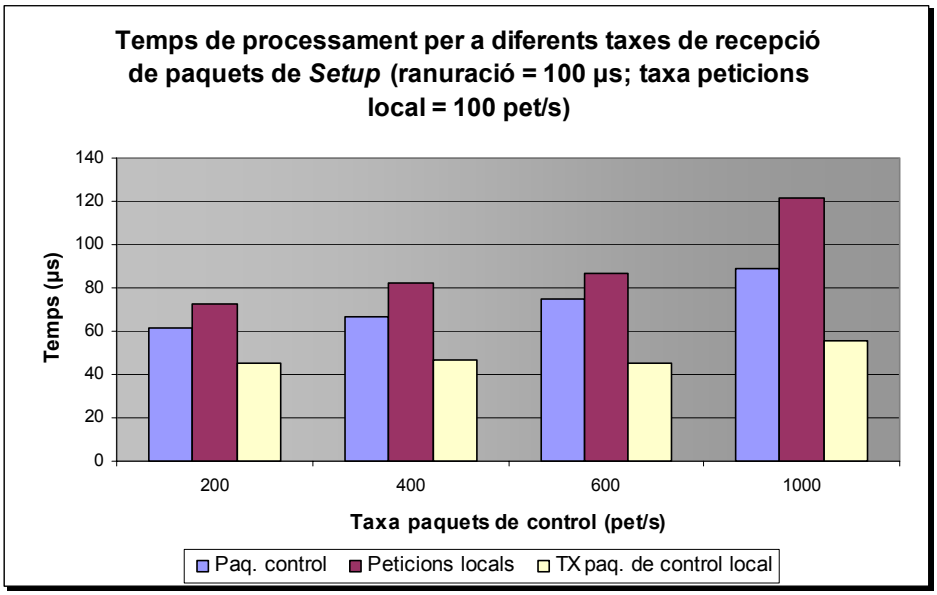
| ID Prova | Tram. Des-car-tades | Setups Rebuts | Setups No Reservats | Setups Reservats | Peticions | Peticions No Reservades | Peticions Reservades | Reserves Iniciades | Reserves Finalitza. | Ranures TX | Ranures RX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Q01_1 | 0 | 2993 | 270 | 2723 | 1547 | 6 | 1541 | 4097 | 4097 | 9104 | 11327 |
| Q02_1 | 6 | 3950 | 669 | 3281 | 1033 | 27 | 1006 | 4097 | 4098 | 5888 | 13529 |
| Q03_1 | 15 | 4724 | 1186 | 3538 | 830 | 38 | 792 | 4097 | 4098 | 4644 | 14134 |
| Q04_1 | 40 | 6150 | 2333 | 3817 | 652 | 79 | 573 | 4098 | 4098 | 3268 | 14742 |
| Q05_1 | 0 | 2218 | 262 | 1956 | 2299 | 10 | 2289 | 4096 | 4097 | 13444 | 8130 |
| Q06_1 | 0 | 3277 | 674 | 2603 | 1717 | 33 | 1684 | 4097 | 4097 | 9848 | 10448 |
| Q07_1 | 0 | 4043 | 1096 | 2947 | 1437 | 69 | 1368 | 4100 | 4100 | 7908 | 11730 |
| Q08_1 | 0 | 5456 | 2086 | 3370 | 1160 | 145 | 1015 | 4102 | 4102 | 5808 | 12852 |
| Q09_1 | 0 | 1450 | 256 | 1194 | 3065 | 31 | 3034 | 4097 | 4097 | 17764 | 4822 |
| Q10_1 | 0 | 2417 | 590 | 1827 | 2520 | 92 | 2428 | 4099 | 4099 | 14204 | 7362 |
| Q11_1 | 0 | 3196 | 1011 | 2185 | 2287 | 173 | 2114 | 4099 | 4099 | 12204 | 8621 |
| Q12_1 | 0 | 4611 | 1921 | 2690 | 1970 | 325 | 1645 | 4100 | 4100 | 9388 | 10232 |

**Taula H.9.** Taxa de reserva i throughput.

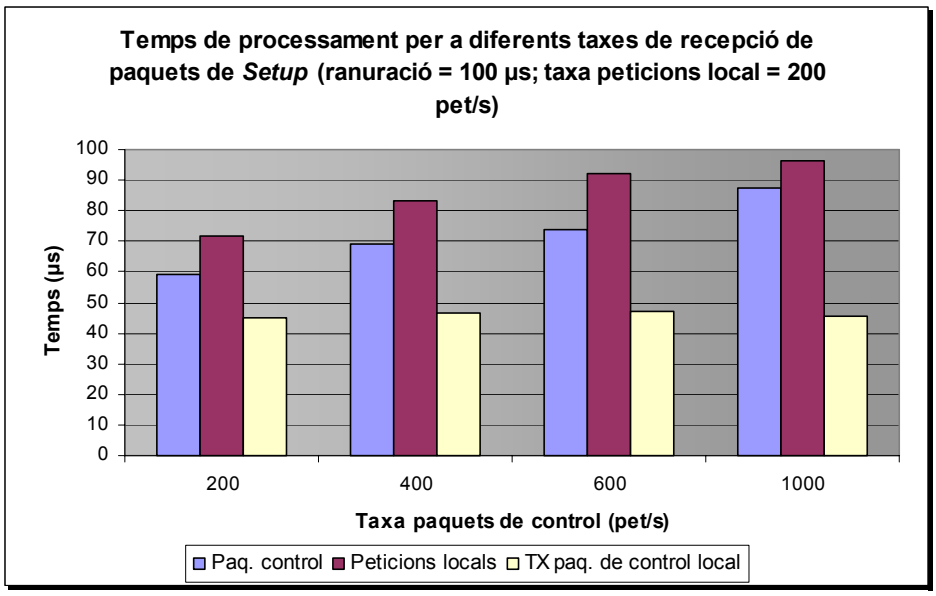
| ID Prova | Bits/slot | Taxa Reser. Client | Taxa Reser. Queries | Taxa Reser. Total | Ranu. / RX | Ranu. / TX | Taxa Reser. Ranur. RX / segon | Taxa Reser. Ranur. TX / segon | Taxa Reser. Ranur. Total / segon | % Reserv. RX (Client) | % Reserv. TX (Queries) | % Reserv. | Taxa Ràfe. (Mbps) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Q01_1 | 100000 | 181,96 | 99,61 | 281,57 | 4,2 | 5,9 | 756,90 | 588,49 | 1345,39 | 7,57% | 5,88% | 13,45% | 134,539 |
| Q02_1 | 100000 | 332,25 | 97,39 | 429,64 | 4,1 | 5,9 | 1370,03 | 569,99 | 1940,02 | 13,70% | 5,70% | 19,40% | 194,002 |
| Q03_1 | 100000 | 449,36 | 95,42 | 544,79 | 4,0 | 5,9 | 1795,17 | 559,52 | 2354,69 | 17,95% | 5,60% | 23,55% | 235,469 |
| Q04_1 | 100000 | 620,65 | 87,88 | 708,53 | 3,9 | 5,7 | 2397,07 | 501,23 | 2898,30 | 23,97% | 5,01% | 28,98% | 289,830 |
| Q05_1 | 100000 | 176,38 | 199,13 | 375,51 | 4,2 | 5,9 | 733,09 | 1169,55 | 1902,64 | 7,33% | 11,70% | 19,03% | 190,264 |
| Q06_1 | 100000 | 317,73 | 196,16 | 513,89 | 4,0 | 5,8 | 1275,31 | 1147,12 | 2422,43 | 12,75% | 11,47% | 24,22% | 242,243 |
| Q07_1 | 100000 | 437,35 | 190,40 | 627,75 | 4,0 | 5,8 | 1740,79 | 1100,63 | 2841,41 | 17,41% | 11,01% | 28,41% | 284,141 |
| Q08_1 | 100000 | 617,67 | 175,00 | 792,67 | 3,8 | 5,7 | 2355,57 | 1001,38 | 3356,95 | 23,56% | 10,01% | 33,57% | 335,695 |
| Q09_1 | 100000 | 164,69 | 395,95 | 560,64 | 4,0 | 5,9 | 665,10 | 2318,30 | 2983,41 | 6,65% | 23,18% | 29,83% | 298,341 |
| Q10_1 | 100000 | 302,36 | 385,40 | 687,76 | 4,0 | 5,9 | 1218,37 | 2254,60 | 3472,97 | 12,18% | 22,55% | 34,73% | 347,297 |
| Q11_1 | 100000 | 410,20 | 369,74 | 779,94 | 3,9 | 5,8 | 1618,46 | 2134,50 | 3752,96 | 16,18% | 21,34% | 37,53% | 375,296 |
| Q12_1 | 100000 | 583,39 | 334,01 | 917,40 | 3,8 | 5,7 | 2219,04 | 1906,19 | 4125,23 | 22,19% | 19,06% | 41,25% | 412,523 |

## H.3.2. Gràfiques de temps de processament

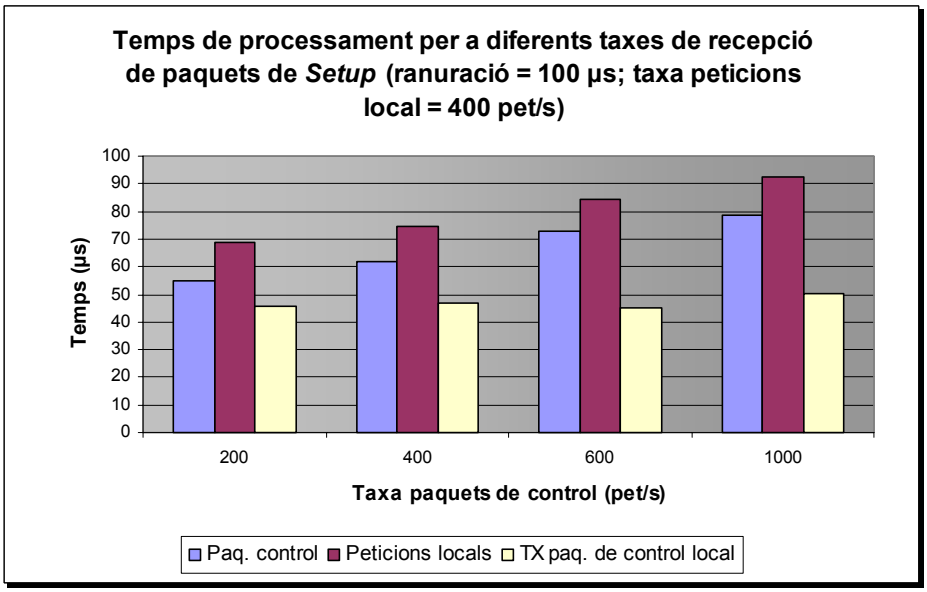
En les següents tres gràfiques (veure Graf H.17, Graf H.18 I Graf H.19) es mostra l'evolució del temps de procés de les diferents funcions analitzades per a una determinada taxa de peticions locals de TX de ràfega. El pas d'una gràfica a l'altra presenta un increment d'aquesta taxa. Com es pot observar, per a una taxa determinada, al incrementar la taxa de recepció de paquets del programa client, el temps de procés augmenta per a les diferents funcions. Aquest comportament ja s'ha explicat en l'apartat 5.2.4 on s'ha inclòs la gràfica Graf H.19.




**Graf H.17.** Temps de processament per a diferents taxes de recepció de paquets (taxa local = 100 pet/s).

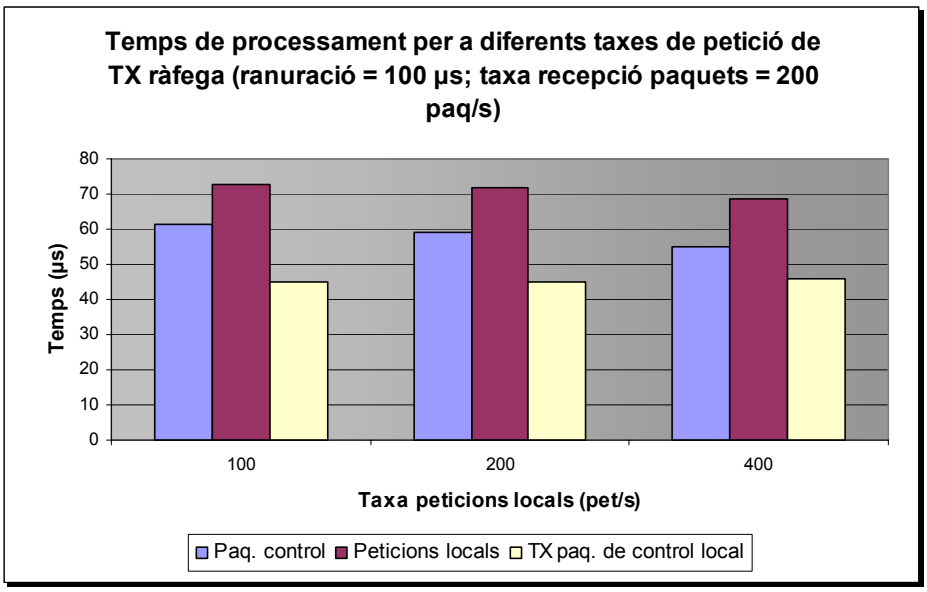



**Graf H.18.** Temps de processament per a diferents taxes de recepció de paquets (taxa local = 200 pet/s).

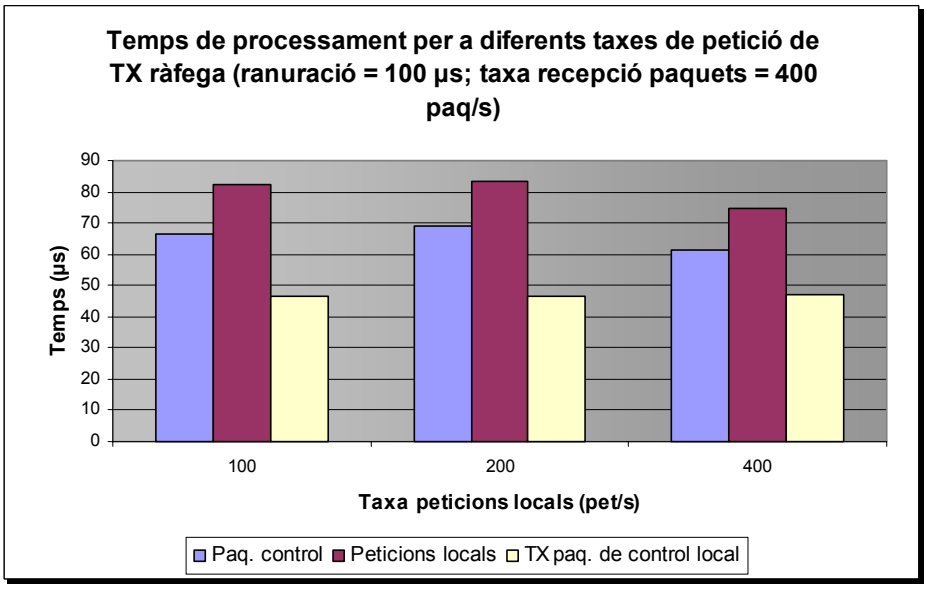



**Graf H.19.** Temps de processament per a diferents taxes de recepció de paquets (taxa local = 400 pet/s).

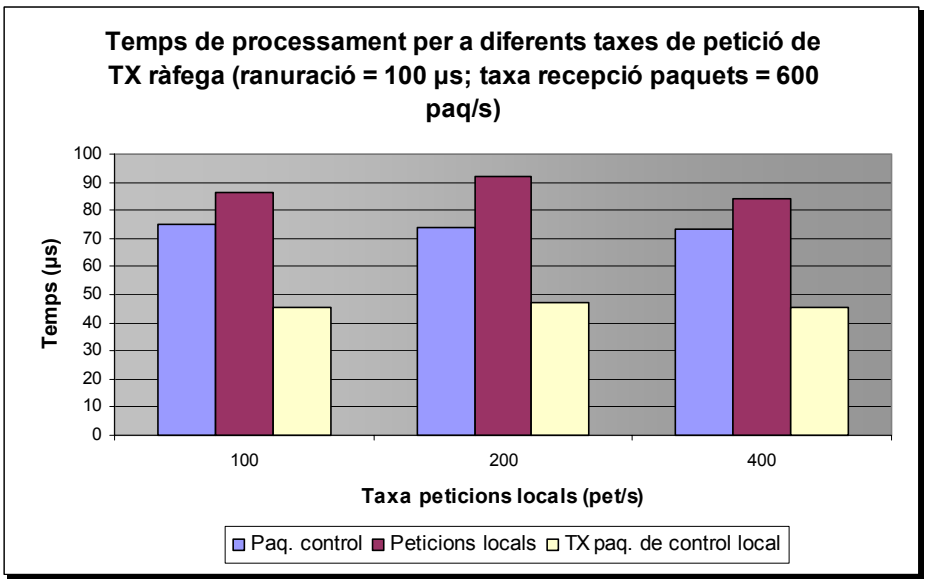
De forma anàloga, en les següents quatre gràfiques (veure Graf H.20, Graf H.21, Graf H.22 i Graf H.23) es mostra l'evolució entre gràfiques al incrementar la taxa de recepció de paquets, des d'un valor inicial de 200 paq/s, fins a 1000 paq/s. El comportament en aquest cas difereix dels de les gràfiques anteriors. Ara, per a una taxa de paquets determinat, al incrementar la taxa de peticions locals, el temps de procés disminueix lleugerament.



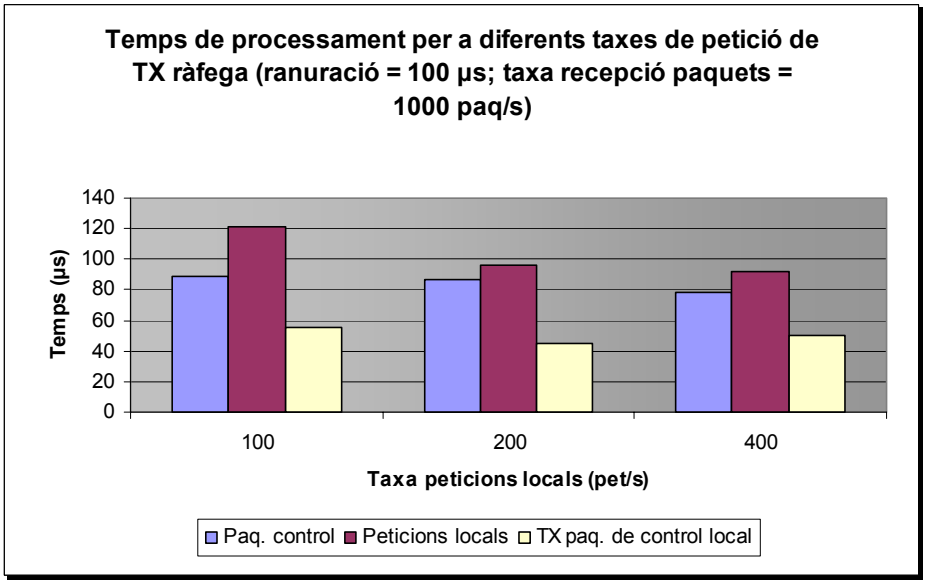

**Graf H.20.** Temps de processament per a diferents taxes TX de ràfega local (taxa recepció paquets = 200 paq/s).




**Graf H.21.** Temps de processament per a diferents taxes TX de ràfega local (taxa recepció paquets = 400 paq/s).



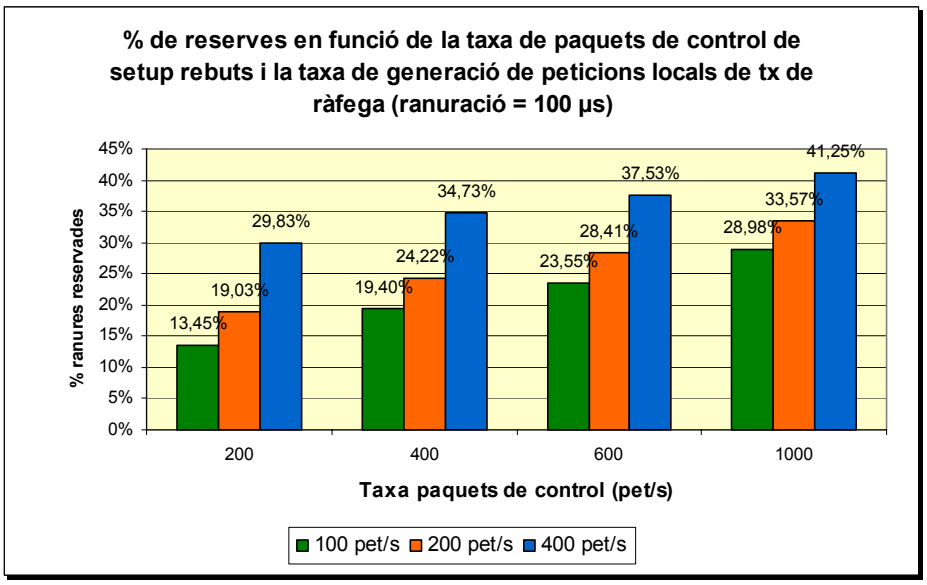

**Graf H.22.** Temps de processament per a diferents taxes TX de ràfega local (taxa recepció paquets = 600 paq/s).



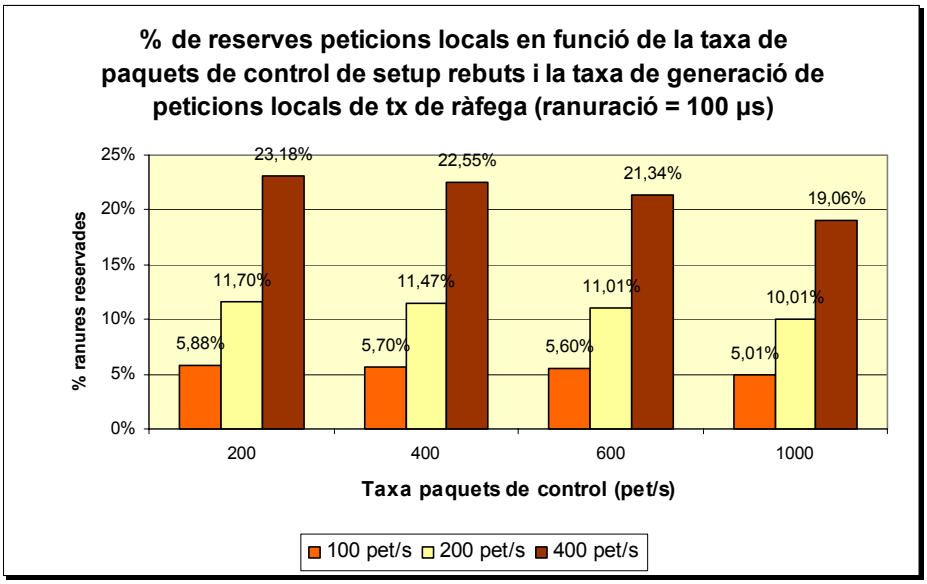

**Graf H.23.** Temps de processament per a diferents taxes TX de ràfega local (taxa recepció paquets = 1000 paq/s).

### H.3.3. Gràfiques de reserva de recursos

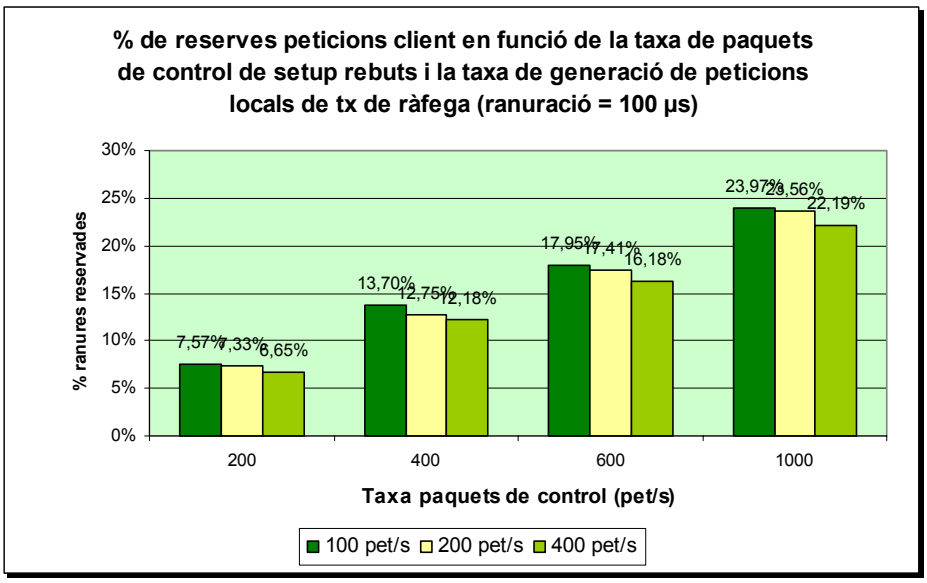
Les següents quatre gràfiques (Graf H.24, Graf H.25, Graf H.26 I Graf H.27) estan comentades en el cos de la memoria, apartats 5.2.4 i 5.3.



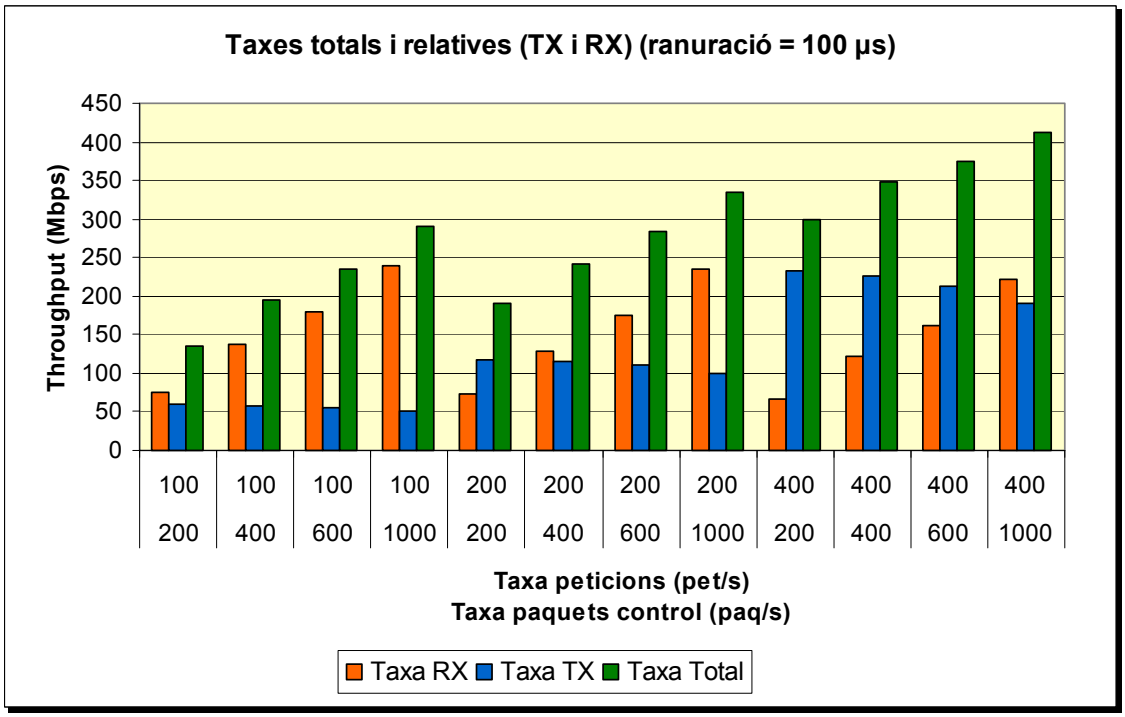

**Graf H.24.** % reserves funció de la taxa de paquets rebuts i taxa de peticions locals.




**Graf H.25.** % reserves locals en funció de la taxa de paquets rebuts i la taxa de peticions locals.




**Graf H.26.** % reserves de paquets rebuts en funció de la taxa de paquets rebuts i la taxa de peticions locals.




**Graf H.27.** Taxes totals relatives.

# ANNEX I. ARTICLE PER A L'EUNICE 2006

# An Optical Burst Switching Control Plane Architecture and its Implementation

J. Triay, J. Rubio and C. Cervelló-Pastor
Dep. Enginyeria Telemàtica, Universitat Politècnica de Catalunya
E-mail: joan.triay@upc.edu, jesus.rubio@entel.upc.edu , cristina@entel.upc.edu

***Abstract*—This paper proposes a new design and implementation of a control plane for Optical Burst Switched networks. The design is based on the principles of generality, transparency, portability and efficiency. In this way, the control plane is designed to be easily reused in any type of network node, low-level Data Plane or high-level Wavelength Reservation Scheme. Making efforts to address these issues, we implement a general-purpose, flexible and feasible OBS network testbed using field programmable gate arrays (FPGA).**

## I. Introduction

THE rapid growth of Internet traffic requires high transmission rates between network nodes, specially in core networks. These rates fall beyond conventional electronic router's capabilities. The exploitation of huge bandwidth in optical fiber cost effectively is essential for the development of the next-generation optical Internet.

Current WDM networks operate over point-to-point links, being required an optical-to-electrical-to-optical (O/E/O) data conversion at every step. The future designs are focused on optical networks where the user data travel entirely in the optical domain without any O/E and E/O conversion. Three main optical network solutions have been proposed as Gauger *et al.* presents in [1]: Wavelength-Routed Networks (WRNs), Optical Burst Switched Networks (OBSNs), and Optical Packet Switched Networks (OPSNs). Due to the lack of a practical pure-optical packet switching solution for an optical Internet, OBS has been proposed as the best feasible alternative.

OBSNs have some advantages in comparison to the other two technologies [2] [3]. On one side, it efficiently uses statistical multiplexing in the optical layer to overcome the wavelength switching inefficiency of WRNs. On the other side, it is more feasible to be implemented than OPSs since requirements of the last one demand for very high switching rates, all optical processing, and faster optical memory technologies that, in fact, are not commercially available yet. Optical approaches are under development to address the problem of switching capacity in the data network but commercial solutions are not expected within a few next years.

Another advantage of OBS, and maybe the most important one, is that the switching time requirements are much more relaxed, since the length of a burst generally is much longer than the length of a packet. Currently, optical switches of higher performance provide a switching time between 25 ns to 1 ms and a switching rate of about 10000 operations per second depending on their switching technology. OBS's switching requirements can easily fall between those values.

In optical burst switching (OBS), data is transported in various-size units, which are called bursts. This is the basic data block that is transferred, and it can be defined as a collection of data packets that are assembled into a bigger unit. These packets can share any of these features: they can have the same network egress address, or have common attributes, like QoS requirements.

Nodes in an OBS network can either be edge nodes or core nodes. Edge nodes are responsible for assembling input packet within burst. It is a task of these nodes to determine the way of assembling these bursts.

Core nodes are responsible for receiving, processing and forwarding the control packets and reserving the optical resources, specifically, those designated to carry out the switching on the optical cross-connect matrix (OCX) at a concrete time between input and output data ports.

OBS differentiates two planes, as it is shown in Fig. 1: the data plane and the control plane. The all-optical (OOO) data plane is responsible for the transportation of the bursts as optical signals; meanwhile the opto-electronic (OEO) control plane is responsible for the signaling, routing, network management and other network control functions. The data plane is entirely optics; this means that bursts are just switched in the optical domain. On the other way, control plane requires the processing of the control packets at nodes, and so, an O/E/O conversion is required. A control packet is transmitted ahead of the burst in order to configure the switches along the burst's path. An OBS node does not wait for confirmation that an end-to-end connection has been set up, instead it starts transmitting a data burst after an offset time, following the transmission of the control packet.

This strong separation between data and control planes offers a better network manageability and flexibility. For this, an innovative way of implementing this technology could be on providing the best performance at each plane.

The main objective of the data plane is to execute the assembling/disassembling of bursts based on the packet's

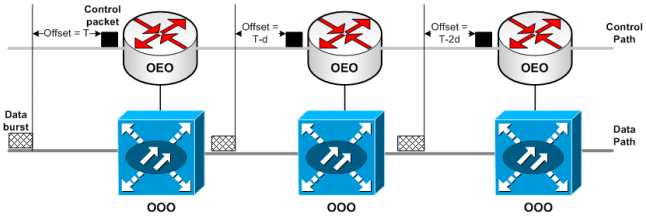



Fig. 1. Separated data and control planes on an OBS network.

attributes as fast as possible.

On the other hand, the objective of the control plane is to carry out the reservation of the optical resources demanded by the control packet.

Due to the one-pass reservation strategy and statistical multiplexing, burst loss can occur. In the case of contention, efficient resolution strategies in OBS core nodes are essential. These schemes achieve a low burst blocking probability as required in transport networks.

This paper focuses on the definition and design of the control plane's features and architecture providing a way for implementing them on real hardware. Currently, there is no standard definition of how this implementation should be done. What is more, there is not any standard protocol of the control plane's operation, for example, the control packet's structure and the control message types remain unstandardized. However, several wavelength reservation schemes have been proposed and research on this field continues.

The paper is organized as follows. In Section II control plane design and architecture model are presented, along with the description of the high level operation of the control plane. We also propose a format of the control packets. In Section III, we present some highlights of the technologies that are currently used at the implementation phase. Finally, Sections IV and V are focused on future research and efforts, and the conclusions related to the present work.

## II. CONTROL PLANE DESIGN AND ARCHITECTURE

This section describes the operation protocol of the control plane, and the proposed design and architecture to implement it. As the design will be implemented into a real testbed, some considerations about it should be met. These can be summarized on:

- *Generality*. The control plane must provide a common framework where different reserve protocols could be implemented on the top.
- *Transparency*. Control plane must be independent of the data plane implementation. The same control plane should be easily adapted whatever the data plane is.
- *Portability*. The control plane implementation must be designed for being easily portable to any kind of network node. For example, the same control plane implementation could be used on a node that is acting only as an edge or core router, or a node that has both capabilities.
- *Efficiency*. The last principle focuses on providing the more efficient way of processing control packets, and as a result, to allow the reservation and switching of data bursts as fast as possible.

An example of network topology and possible proof-of-concept implementation is shown on Fig. 2. Three OBS nodes compose this network: two nodes act as edge, and the last operates both as edge and core. The three nodes share the same OXC and each one can be origin or destination of user data.

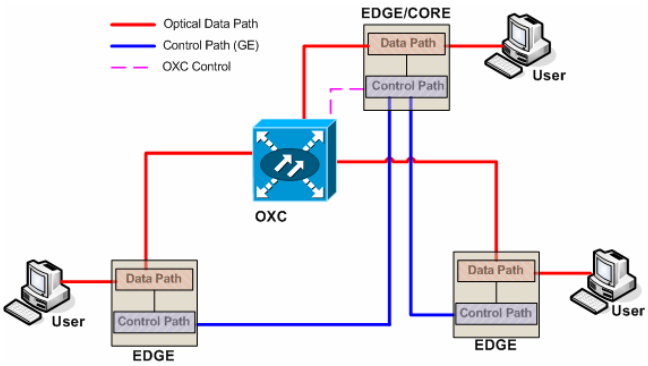



Fig. 2. Data and control plane interactions

### A. High Level Protocol Operation

As previously pointed, generality is a constraint in the design. A common framework must be provided in which different and concrete reserve schemes can be used.

To support this principle a common high level scheme must be implemented. This scheme operates as follows: Considering a flow of bursts between two different endpoints, A and B, for each of these flow a SETUP message will be sent on the control path to the first node, demanding channel resources. The transmission of the burst will be delayed a period of time known as *offset*. During this time, the CPU of the current node must process the message and calculate the ingress and egress ports on which the burst will be received and transmitted, and the exact time and duration of the optical cross-connected matrix switching. Moreover, the control process will regenerate the SETUP message according to the burst attributes. This message is then delivered to the next node through the correct control channel.

It is worth standing out the importance of correctly assigning the switching times of allocating and releasing of resources. Two types of resource release can be noted: implicit, if the node does not need any explicit message or notification (see Fig. 3); or explicit, if a RELEASE message is received from the previous OBS node according to the protocol definition (see Fig. 4). The burst will be deleted if the OBS node has not enough resources to switch it.

### B. Wavelength Reservation Schemes

This subsection describes different wavelength reservation schemes for OBS. Based on the setup and release mechanisms, Baldine *et al.* [4] describe four types of wavelength reservation schemes.

*i)* Explicit setup and explicit release. In this scheme, the control packet contains the offset of the burst, but not the duration of it. The reservation of resources starts immediately after the switch receives the setup message, and ends when the release message is received. *Just-In-Time* (JIT) proposed by Wei and McFarland [5] is an example of this scheme.

*ii)* Explicit setup and an estimated release. In this case, the setup message contains both the offset and the duration of the burst. Each wavelength has an associated deadline indicating the resource will become free. The reservation starts after the switch receives the setup message and ends when the burst is switched and transmitted, a time that is

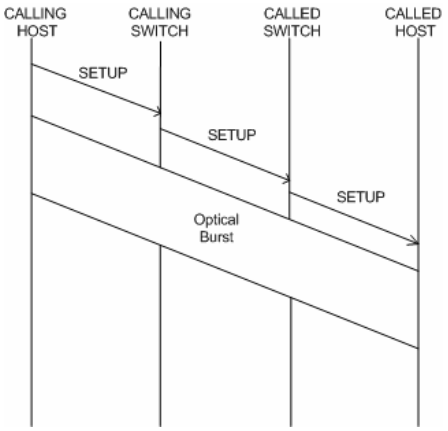



Fig. 3. Example of explicit setup signaling.

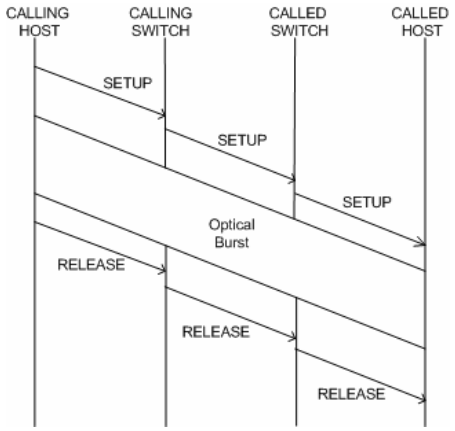



Fig. 4. Example of explicit setup and explicit release signaling.

calculated using the duration field. An example of this scheme type is the *Horizon* scheme proposed by Turner [6].

*iii)* Estimated setup and an explicit release. In this scheme, the setup control message contains only the offset of the burst. The reservation starts at the beginning of the burst. This time is calculated using the previous offset information. The release of resources is executed once the release message arrives.

*iv)* Estimated setup and an estimated release. The setup message of this scheme has the offset and the duration of the burst. Reservation and releasing of resources are calculated using previous data. Qiao and Yoo proposed an example of this scheme, the *Just-Enough-Time* (JET) scheme [2].

The JIT protocol is significantly simpler than either JET or Horizon, since it does not involve complex scheduling or void filling algorithms. Therefore, JIT is amenable to hardware implementations. The difference between JET and JIT resides on the idea of how JET intends to estimate when the data burst will be received, and therefore it won't reserve resources until that instant. This can increase the efficiency and performance of the network in terms of channel utilization, but it can arouse burst looses if the instant is not well calculated. On the other hand, JIT reserves the resources on the instant at which the setup control packet is receive. This method decreases the performance of the network, but it improves the burst losses rate and its implementation is more feasible.

Over the last years, research in OBS networks has progressed to prototypes and proof-of-concepts demonstrations. For example, the JITPAC hardware [7], which was developed by MCNC-RDI, implements the JIT signaling protocol and in [8] the authors present the design and implementation of the JET protocol.

In this work, we propose to implement OBS nodes operating under diverse wavelength reservation schemes in order to obtain low latency and high bandwidth utilization, combined with contention resolution protocols.

### C. OBS Control Packet Format

To support a common framework for the wavelength reservation schemes and their high-level operation, it is necessary to define the format of the control packets. This common format will allow defining easier methods to generate and to process each and every kind of possible control message.

Fig. 5 shows the proposal for control message format specifying its fields and the length. Following, the meaning of each field is described:

- NDA (2 bytes): It is the Node Destination Address. For instance, the egress OBS node's address. It is a 2-byte length field, resulting 65.536 possible addresses. This number is sufficient for deploying an operator's optical transport network. Moreover, the proposed addressing is not hierarchical (there are not different classes of addresses), but reserve 0x0000 and 0xFFFF for operational purposes.
- NSA (2 bytes): The Node Source Address specifies the address of the ingress OBS node. It has the same length of the previous NDA field.
- IDBURST (2 bytes): It is the ID of the burst. Its length, 2 bytes, provides 65.536 possible IDs. This ID is generated by the ingress OBS node. The set of the previous three fields (NDA + NSA + IDBURST) identifies uniquely a burst in the network. Once the OBS node reaches the end of IDs, it should restart its ID counter to 0, and so on.
- TYPE (1 byte): This field specifies the type of control packet. A 1-byte length field gives us up to 256 types of messages. As previously introduced in the figure, some messages can be defined, for instance: SETUP, ACK, NACK and RELEASE messages.
- QOS (1 byte): The QoS field provides information about any QoS requirement that should be met when

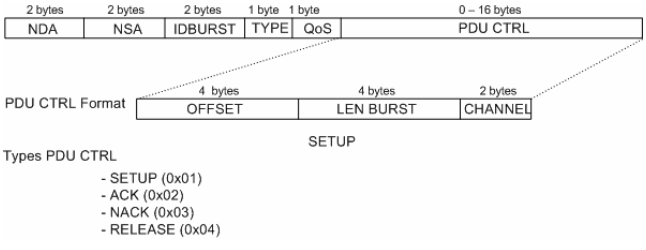



Fig. 5. Fields and length of the control packet.

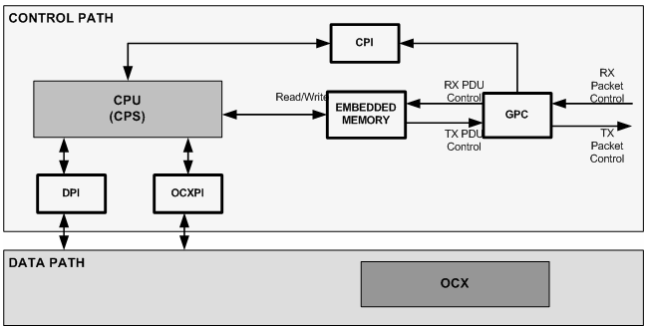



Fig. 6. High level architecture of the control plane.

the burst is being processed. For example, this field could be filled with any QoS profile for adapting the content resolution of bursts at the core routers or when disassembling the burst into packets, if for instance, the packets need a special treatment or de-queuing method.

- PDUCTRL (0-16 bytes): This field varies its contents depending on the type of control packet specified by the TYPE field. For example, the format of a SETUP PDU could be as follows: 4 bytes to specify the OFFSET of the burst, 4 more bytes to inform of the burst length in bytes, and finally 2 bytes to specify on which wavelength (CHANNEL) the burst will be received.

Some of the previous fields can be modified at each node hop. For example, depending on the wavelength reservation scheme that is being used, the offset can be recalculated or not. The input and output wavelengths at core routers could change, and so, the value of CHANNEL field should have a value according to the output wavelength to be used.

The proposed addressing format follows a routing perspective. This means, that at each network hop, the NDA is checked, and the next hop on the path is determined from the routing tables. Moreover, this gives to the nodes the possibility of acknowledging to the ingress OBS node (it just has to check the NSA of the control packet) if any of the bursts are lost because there were not enough resources for switching it.

### *D. OBS Control Path Design*

According to the principles of design specified in the section II, the control plane and its associated control path should be transparent to lower data planes, and so, it must remain as much independent as possible to the concrete data path implementation.

To succeed in performing a fast and efficient treatment of the bursts, most of the elements involved in the design must be implemented directly into hardware, and so, software should only be executed on very specific tasks. Taking this into mind, it is necessary specifying which operations are suitable to be processed by the Control Path Software (CPS). Depending on the kind of router, edge or core, the node must execute different operations. On one hand, an edge node has the following capabilities:

- Assembling and disassembling of bursts depending on the egress destination address or other attributes.
- Transmission and reception of bursts at a certain time over a specified wavelength.
- Creation of control packets depending on the type of traffic to be transmitted.
- Reception and processing of control packets transmitted by other nodes.

On the other hand, a core switch should be capable to perform:

- Reception and processing of control packets.
- Modification of the control packet, if required, and transmission to the next OBS node on the path to the egress node.
- Reservation and release of resources in the OXC according to the information contained in the received control packets.

From above operation, the Control Path Software must be only responsible of creating and processing the contents of the control packet depending on the upper reservation schemes that could be used, or other capabilities, as specific QoS processing. It won't carry out any operation needed to support storing or releasing of packets into/from memory. Fig. 6 shows our control plane architecture, in which the Control Path clearly differentiates to the Data Path and the OCX, but it is not totally isolated. The CPU which runs the Control Path Software, can interact with both through their respective interfaces, the Data Path Interface (DPI) and the OCX Path Interface (OCXPI).

The DPI is responsible for querying the CPS for signaling a new Data Path when one or some bursts are ready to be transmitted as a set of data packet are assembled into a burst/bursts. This task identifies an edge node capability, and so, the core nodes do not interact to the DPI. Analogously, the OCXPI is responsible of translating the CPU queries of switching to the OCX. In contrast to the previous interface, the OCXPI must always interact with the CPU whatever type of node is being implemented.

As a part of the control plane, we also need to receive, process and transmit the control packets. To alleviate the CPS, some pre-process of the control packets is entirely run from hardware. This comprises the Gigabit Packet Controller (GPC) and the embedded memory. The GPC is responsible for assembling control messages into low-level frames in order to be transmitted over any kind of link layer protocol, for instance, a Gigabit Ethernet interface; or de-assembling these link layer frames, extracting the control message and storing it in the embedded memory, from which the CPU and its CPS can process it.

### *E. Control Path Software and Use Case Examples*

Control Path Software (CPS) is responsible for calculating the offset and duration of the bursts. Once this information is ready and recorded, it must be able of constructing the control PDU if a new control packet is going to be transmitted to the next OBS node. Fig. 7 and Fig. 8 show some use cases that the CPS has to carry out.

Fig. 7 exposes the case in which a new burst is ready to be transmitted. The CPS receives the query of signaling a path for transmitting the last burst. Before its transmission, it is necessary to perform a setup and wavelength reservation along the light path. In this way, CPS and its

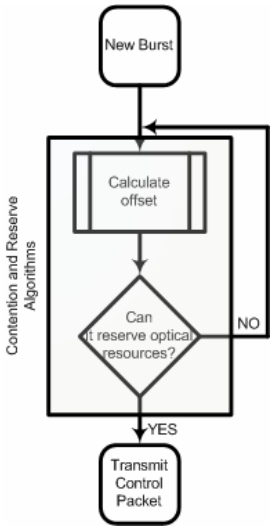



Fig. 7. Control Packet creation algorithm.

higher contention and reserve algorithms calculate the required amount of delay time (offset) depending on the available resources. If transmission is possible, the Control packet is finally assembled and transmitted to the next OBS node. Depending on higher level protocols or QoS profiling, if there are not enough resources, the burst can remain in memory and the CPS can reinitialize the operation of reservation later.

Fig. 8 presents an example of control packet reception/forwarding. In this case, the offset must be checked in order not to reserve resources if the burst is going to be received wrongly. If there is enough offset in which carrying out the control packet processing and reservation of resources, the reception/forwarding of a control packet is possible. If not, this control packet should be discarded, and depending on the reservation protocol, some kind of control message should be transmitted back to the previous OBS node notifying about this error.

In the next step, CPS filters if the control packet's destination address is the current node. In this case, the node only needs to reserve optical resources for reception, and so, forwarding of control packet will not occur. When forwarding is required, the program must check the availability of optical resources to switch the burst. If there are not available resources an explicit notification method should take the responsibility of sending back an error message. In the case of availability of resources, the offset will be recalculated subtracting the processing time. Finally, the control packet is transmitted to the next OBS node on the egress path.

## III. IMPLEMENTATION TECHNOLOGIES

### *A. FPGA: A suitable hardware solution*

FPGA is a technology that allows programming and reusing of hardware. These devices are a compromise between general purpose processors used in software implementations and Application Specific Integrated Circuits (ASIC). FPGA can be reprogrammed as control plane protocols evolve. Moreover, new FPGA provide high speed serial interfaces, from 1 Gb/s to 10 Gb/s. These

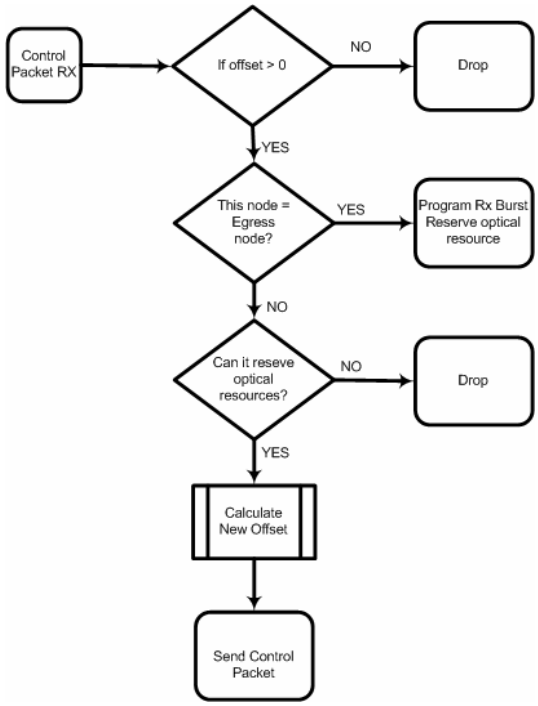



Fig. 8. Control Packet reception/forwarding algorithm.

interfaces allow the implementation of proprietary protocols and variations of standards. Therefore, this technology becomes the most appropriate to develop the control plane, in which a process of design, implementation and testing is required.

Another interesting capability of the current generation of FPGA is that they can assemble one or more RISC CPUs. This kind of CPUs are characterized by their capacity and processing power for running any type of software, which for instance could manage any wavelength reservation scheme for an OBS network.

Currently we are working with a Xilinx's Virtex II Pro FPGA [9] [10]. This FPGA model contains all the characteristics specified previously. Specifically, this chipset has RocketIO Multi-Gigabit Transceiver [11] interfaces that allow serial transmissions with rates among 622 Mb/s up to 3.125 Gb/s. It also contains 2 IBM's PowerPC 405 [12] CPUs of 32 bits that have a maximum clock frequency of 400 MHz and give a maximum capacity of 600+ DMIPS. This is enough calculation power for the purposes on the implementation of the CPS.

The development kit is Avnet's badge PCI board (Xilinx Virtex-II Pro Development Kit, see Fig. 9) that has the following components:

- Xilinx Virtex-II Pro XC2VP30 that contains

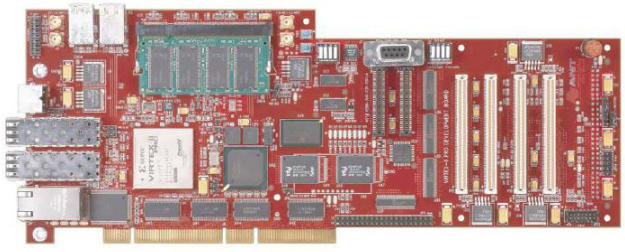


Fig. 9. Avnet's Virtex-II Pro Development Kit.

30.000 logical cells to implement the Control Path hardware.
- 2 SFP Modules for GigabitEthernet.
- 10/100/1000 GigabitEthernet physical interfaces.
- 32 MBytes of SDRAM memory.
- 128 MBytes of DDR SDRAM memory.
- Four 140-pin general purpose I/O expansion connectors (AvBus).

This kit will provide in future research sufficient capabilities to connect a Data Path by means of their extension connectors.

### *B. Link and Physical Layers*

One of the decisions to adopt is which physical level protocol should be used to communicate the control plane between nodes. This protocol should provide simplicity in its implementation, as well as efficiency in transmission. A minimum speed of transmission of 1 Gb/s is also required.

The protocol that endows with these requirements is GigabitEthernet, because it has a great efficiency and it is easy to implement. Therefore the control messages defined in section II.C will be encapsulated in a GigabitEthernet frame.

### *C. Control Plane's Implementation*

Next, the control plane proposal implementation explained in previous sections will be explained, specifically, how a GPC, Control Packet Interface, DPI and OXCPI modules work.

The processing of control packets is formed by two blocks: a Gigabit Packet Controller (GPC) and a Control Packet Interface (CPI). The GPC module is directly connected to GigabitEthernet MAC which offers Gigabit encapsulation to control messages. This block takes charge on de-framing the information of the OBS control protocol and writing these data in the embedded memory of the PPC405 CPU. When this process ends, the GPC generates an interruption to the processor. Then the microprocessor would request the GPC for where the control message information is stored. The GPC answers the CPU with a pointer which indicates the memory address of the control messages and the size and type of the message that has been recently received.

Analogously, in the case the CPS needs to deliver a control packet, it would advise the GPC the pointer where control data has been stored, size of this message and the OBS node's destination address. With this information GPC will create a new GigabitEthernet frame filling its data field with the control packet. GPC communicates with the PPC405 by means of the CPI. At the same time, the CPI is a module that is connected to the processor by means of the DCR bus.

This way of implementation is a recommendation of Xilinx in an Application Note [13] where they indicate that this is the best method to obtain a greater performance on packet processing. Xilinx compared this processing to other architectures on a later document [14].

This IBM's DCR bus (CoreConnect Architecture [15]) is composed by a bus of data, which is responsible for writings, and another bus that processes the readings of 32-bit width words. It also uses three signals of control (READ, WRITE, ACK) [16]. The speed of data transferring of this bus is of surroundings 1 Gb/s. The PPC405 gains access to DCR bus data by means of a reading instruction and a writing instruction. In these instructions it has to indicate the address to which data are written or from they are read. The word's width is 10 bits and it is completely independent from the memory system organization. Therefore an interval of addresses should be reserved to enable this operation between CPS and GPC.

The DPI and OXCI modules carry out the same functions of the CPI, that is to say, some addresses of the BUS DCR of the PPC405 are reserved and its mission is decoding these addresses in order to enable writings and readings to/from the DataPath's and OCX's registries.

## IV. FUTURE RESEARCH

Currently we are at the implementation phase of the Control Plane and one important goal for the immediate future is to analyze the results. An interesting point is that we not only work on simulations but we are implementing a real testbed, and so, results are expected to be more accurate to any future commercial OBS device.

In this testbed we implement our proposed control plane algorithms. These proposed algorithms have been previously simulated.

Despite we have focused on providing a transparent control plane for OBSNs, in order to correctly evaluate this plane we will also evaluate how it can be integrated with a data plane. This is necessary not only for checking the correct control plane behavior, but also for providing to the data plane accurate, fast and efficient optical paths.

## V. CONCLUSION

In this paper we have given an overview of the OBS networks and presented architecture of the control plane that is currently being implemented. Due to the lack of protocol standards related to OBS, main efforts are designated to design, implement and test a control protocol for optical burst switching. Some problems to tackle are the synchronization between nodes in the control plane or the optimization of processing time in the CPS. This architecture focuses on providing a common framework on which to implement any kind of wavelength reservation scheme, and so, giving an interesting platform to researchers where future developments of OBS could be carried on.

## ACKNOWLEDGMENT

This work is founded by the Spanish Ministerio de Ciencia y Tecnología (MCYT) and FEDER within the framework of the TIC2003-09042-C03-02 project, the i2CAT Foundation and the EURO-NGI project.

## REFERENCES

[1] C. Gauger *et al.*, "Service Differentiation in Optical Burst Switching Networks", *ITG Fachtagung Photonische Netze,* 2001, pp. 124-32.

[2] C. Qiao and M. Yoo, "Optical Burst Switching (OBS) – A New Paradigm for an Optical Internet", *J. High Speed Nets*, vol. 8, no. 1, Jan. 1999, pp. 69-84.

[3] Fei Xue, S.J.B. Yoo, H. Yokoyama, and Y. Horiuchi, "Performance comparison of optical burst and circuit switched networks", *Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC*, Volume 3, March 2005, pp 3.

[4] I. Baldine, G. N. Rouskas, H. G. Perros, and D. Stevenson. "JumpStart: A just-in-time signaling architecture for WDM burst-switched networks", *IEEE Communications*, 40(2), pp. 82-89, Februrary 2002.

[5] J. Y. Wei and R. I. McFarland, "Just-in-time signaling for WDM optical burst switching networks", *Journal of Lightwave Technology*, 18(12), pp. 2019-2037, December 2000.

[6] J. S. Turner, "Terabit burst switching", *Journal of High Speed Networks*, 8(1): pp. 3-16, January 1999.

[7] I. Baldine, M. Cassada, A. Bragg, G. Karmous-Edwards and D. Stevenson, "Just-in-time optical burst switching implementation in the ATDnet all-optical networking testbed" in Proc. GLOBECOM'03, vol. 5, pp. 2777-2781, San Francisco, CA, Dec. 2003.

[8] Y. Sun, T. Hashiguchi, V. Q. Minh, X. Wang, H. Morikawa and T. Aoyama. "Design and Implementation of an Optical Burst-Switched Network Testbed'', IEEE Optical Communications, vol. 3, No. 4, pp. S48-S55, Nov. 2005.

[9] Xilinx Inc., *Virtex-II Pro Product Brochure*. [Online document], [visited April 2006], Available HTTP: http://www.xilinx.com/products/virtex2p/v2p_brochure.pdf.

[10] Xilinx Inc., *Table Virtex II Pro Family Device/Package Combinations and available RocketIO and RocketIO X*. [Online document], [visited April 2006], Available HTTP: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/product_table.htm.

[11] Xilinx Inc., *RocketIO Multi-Gigabit Transceiver*, [Online document], [visited April 2006], Available HTTP: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/capabilities/rocket_io_mgt.htm.

[12] Xilinx Inc., *PowerPC 405 Processor,* [Online Document], [Visited April 2006], Available HTTP: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fgpas/capabilities/powerpc.htm.

[13] Xilinx Inc., *XAPP669 – PPC405 PPE Reference System Using Virtex-II Pro RocketIO Transceiver,* [Online document], [Visited April 2006], Available HTTP: http://direct.xilinx.com/bvdocs/appnotes/xapp669.pdf.

[14] Xilinx Inc., *XAPP664 – PLB vs. OCM Comparison Using the Packet Processor Software,* [Online document], [Visited April 2006], Available HTTP: http://direct.xilinx.com/bvdocs/appnotes/xapp664.pdf.

[15] Xilinx Inc., *Core Connect Architecture – Device Control Register Bus (DCR)*, [Online Document], [Visited April 2006], Available HTTP: http://www.xilinx.com/ipcenter/processor_central/coreconnect/coreconnect_dcr.htm.

[16] Xilinx Inc., *PowerPC 405 Processor Block Reference Guide*, Chapter2. Section "External DCR Bus Interface", pp. 102-110, [Online Document], [Visited April 2006], Available HTTP: http://www.xilinx.com/ise/embedded/ppp405block_ref_guide.pdf.