
# ARQUITECTURA WORMHOLE PER A UNA NOC 2D-MESH

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Eduard Fernandez Alonso
i dirigit per
Jordi Carrabina Bordoll i Jaume
Joven Murillo Bellaterra 17 de
Setembre de 2007

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Jordi Carrabina i Bordoll, i Jaume Joven i Murillo, Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

**CERTIFICA**:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Eduard Fernandez Alonso

I per tal que consti firma la present.

Signat: Jordi Carrabina Bordoll  Jaume Joven Murillo

Bellaterra, 17 de Setembre de 2007

# DEDICATÒRIA

No vull donar per acabada la memòria sense abans haver agraït a totes les persones que m'han ajudat en aquest llarg camí fins al final de la carrera. Gràcies a tots ells. Alguns ja van acabar, altres encara no ho han fet i n'hi ha que es van quedar pel camí, però sense ells no seria qui sóc ni estaria on estic.

Vull agrair especialment tot el suport incondicional que he rebut de la meva família, que sempre ha confiat en mi, i encara més de la Laia, que m'ha ajudat a revisar aquesta memòria i m'ha donat ànims en els moments més durs.

Finalment, vull donar les gràcies per tot el suport i la guia rebuts durant el projecte per part d'en Jaume, i el bon rollo del meu company i amic Sergi. Ha estat divertit, oi?

# INDEX

Pàgina


**1. Introducció ..... 1**
1.1. Introducció i anàlisi del projecte ..... 2
1.2. Motivacions i objectius del projecte ..... 4

**2. Evolució i conceptes bàsics sobre NoCs ..... 5**
2.1. Evolució microelectrònica: des dels ASICs als NoCs i SoCs ..... 6
2.2. Conceptes bàsics ..... 12
2.2.1. Packet switching ..... 14
2.2.1.1. Store-and-forward ..... 14
2.2.1.2. Wormhole ..... 15
2.2.1.3. Virtual cut-through ..... 17
2.2.2. Circuit switching ..... 18

**3. Disseny i metodologia de treball ..... 19**
3.1. Flux de disseny i metodologia de treball ..... 20
3.2. Llenguatges descriptius de hardware (HDL) ..... 25
3.3. Verificació ..... 27
3.3.1. Simulació i ModelSim ..... 27
3.3.2. Síntesi i Synplify ..... 28

**4. Implementació ..... 29**
4.1. Disseny del router ..... 34
4.1.1. Protocol de control Simple 4-phase handshake full duplex ..... 36
4.1.2. Format del paquet ..... 37
4.1.3. Arquitectura del router per packet switching NoC ..... 41

**5. Resultats ..... 44**
5.1. Costos Hardware ..... 50
5.1.1. Àrea usada per la NOC ..... 50
5.2. Simulacions ..... 54
5.3. Comparació amb altres Nocs ..... 55

**6. Conclusions i línies de futur ..... 57**
6.1. Conclusions ..... 58
6.2. Línies de futur ..... 60

**Bibliografia ..... 63**

**Resum ..... 65**

# CAPÍTOL 1

## Introducció

## 1.1 Introducció i anàlisi del projecte

En la microelectrònica de l'actualitat comencen a ser una realitat els xips que contenen sistemes complets en el seu interior (*System-on-Chip, SoC)*. Gràcies a la llei de Moore, la tecnologia fa possible poder integrar dintre de la mateixa pastilla de silici més quantitats de recursos, com ara memòries, processadors i DSP i altres components.

Aquest avenç tecnològic dota els dissenyadors i enginyers d'una quantitat cada vegada més gran de recursos disponibles per als seus "integrats", i tot en el mateix espai, o de vegades fins i tot menor.

El gran problema existent des dels inicis de la microelectrònica i de la informàtica, el poder de càlcul, està evolucionant actualment en un nou problema centrat en les comunicacions.

Les solucions de connectivitat tipus punt a punt, o fins i tot les de tipus *On-Bus*, van deixant pas a les primeres xarxes de comunicacions *On Chip.*

Així doncs, de la mateixa manera que els ordinadors fan servir protocols i elements de xarxa per aconseguir la comunicació, els sistemes integrats en un procés anàleg han anat creant els seus protocols i els seus elements de xarxa per poder millorar la transferència de dades entre els diferents components que formen el sistema.

L'element de xarxa més important és el *router*, i el següent pas dintre de l'evolució en la microelectrònica és passar dels *System-on-Chip (SoC)* als *Network- on–Chip (NoC),* on la connectivitat adquireix una rellevància especial.

La realització d'aquest projecte implicarà l'estudi dels sistemes integrats en un xip, així com les diferents tècniques de commutació de circuits i paquets (circuit switching o packet switching) existents per a aquests components microelectrònics.

També caldrà revisar els coneixements adquirits durant la carrera dels llenguatges descriptius de *hardware* disponibles, així com d'altres conceptes que s'hauran de posar en pràctica, com poden ser les màquines d'estats, els diagrames de flux, sistemes combinacionals , LUT's i altres conceptes de disseny de circuits o sistemes integrats.

El projecte l'he dividit en quatre grans etapes:

1. Estudi
2. Implementació
3. Verificació
4. Experimentació i extracció de resultats

Les quatre etapes, atès el seu interès evident, estan explicades en els capítols que vénen a continuació, i a totes els dedico un mínim d'un punt de la memòria o un capítol sencer en alguns casos.
D'aquesta manera, doncs, la memòria ha quedat dividida en 5 punts principals:

- Un capítol, anomenat evolució i conceptes bàsics sobre NoCs, dedicat a l'evolució de la microelectrònica que ens emmarcarà dintre de l'entorn tecnològic del projecte.
- Un capítol centrat en el disseny i la metodologia seguida en la realització del projecte, que, juntament amb el capítol anterior, engloben la primera etapa, l'estudi del projecte.
- En el capítol Implementació es dissenyarà i analitzarà l'arquitectura del *router* i dels components necessaris.
- La verificació i els resultats obtinguts s'analitzaran en el penúltim capítol, Resultats.
- Finalment, es dedicarà un capítol sencer a l'explicació de les conclusions del projecte i de les possibles línies de futur.

Als punts anteriors cal afegir aquest capítol introductori, amb el qual conformen la totalitat de la memòria, tal com es pot veure més detalladament a l'índex.

## 1.2 Motivacions i objectius del projecte

L'objectiu principal d'aquest projecte és dissenyar una arquitectura de router per a una NoC de tipus 2D-Mesh, basant-se en l'arquitectura de *router* de tipus *packet switching* on la tècnica de commutació que es vol emprar és una wormhole.

Entendre l'arquitectura de les NoCs, endinsar-me en la microelectrònica, estudiar les diferents estratègies d'encaminament, *circuit switching* o *packet switching*, i sintetitzar els components seran els objectius personals del projecte.

Aconseguir el correcte funcionament dels components creats i arribar al nivell descriptiu del component, RTL, en seran els resultats tangibles.

Per tot el que he comentat fins ara, em resulta fàcil justificar la meva elecció personal en la tria d'aquest repte proposat pel Dr. Jordi Carrabina i Bordoll, i dirigit per l'enginyer Jaume Joven i Murillo, com a projecte final de carrera.

L' univers dels sistemes integrats i concretament dels SoCs/NoCs és un camp encara jove i en expansió, amb moltes novetats i amb molt camí per explorar i recórrer.

Durant la carrera he pogut fer un petit tast sobre aquest món en creació, i m'ha semblat un colofó perfecte per a la carrera finalitzar-la fent el projecte del departament de microelectrònica de la Universitat Autònoma de Barcelona, del qual he pogut gaudir de totes les assignatures optatives ofertades i dels seus professors.

# CAPÍTOL 2

## Evolució i conceptes bàsics sobre NoCs

## 2.1 Microelectrònica: des dels ASICs als NoCs i SoCs

En l'actualitat l'evolució de la tecnologia microelectrònica permet la integració d'uns pocs milions de transistors dintre d'un mateix xip. Això permet el disseny, el desenvolupament i la fabricació de sistemes complets dins d'un mateix dau.

Durant els anys 90 aquests sistemes complets integrats en un mateix xip de silici es van etiquetar com a ***System-On-Chip*** **(SoC)**. [Jantsch03]
Aquest fet va portar com a conseqüència inevitable l'aparició d'un nou paradigma de disseny. L'Era SoC agafava el relleu de la seva predecessora, l'Era ASIC, que va tenir un paper destacat en el món de la microelectrònica cap als anys 80.
Durant l'Era ASIC el nivell d'integració de portes va evolucionar dels aproximadament 20K portes al principi dels anys 80 fins als 500k aproximadament que es van poder aconseguir cap al final de la seva època. Els SoCs poden integrar més de 2,5 milions de portes en el mateix silici.

Malgrat tot, aquesta tendència evolutiva de capacitat d'integració no s'atura i, per tant, els SoCs poden anar incorporant més components al sistema. Això comporta un gran número de problemes de comunicació.
En els SoCs l'element principal és el bus central de comunicació al qual estan connectats tota la resta d'elements components del sistema. Aquesta topologia de comunicació és difícilment escalable. Els busos poden escalar eficientment de 3 a 10 components, però es fa molt complicat incrementar aquest número. [Jantsch03].

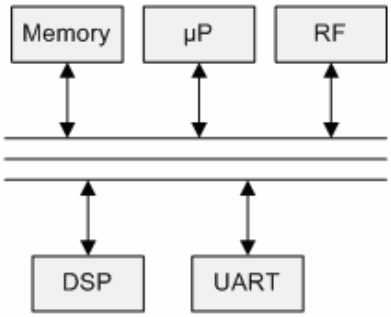



Figura 2.1: Exemple de SoC amb un bus principal.

Com a conseqüència d'aquest fet, cap al 1999 es va començar a pensar a millorar la comunicació dintre dels SoCs i van aparèixer així les ***Network-On-Chip*** **(NoC)**, on els sistemes de comunicació estan distribuïts i no centralitzats en un sol bus. [Jantsch03]

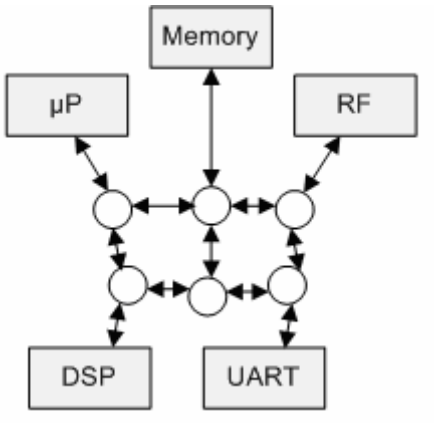



Figura 2.2: Exemple de NoC amb una xarxa.

En tot aquest camí els dissenyadors han intentat sempre aprofitar al màxim les possibilitats d'integració que hi havia disponibles, de manera que les eines de disseny i metodologies de treball van guanyar una extrema importància.

Durant aquest procés es va observar que tant les metodologies com les eines de disseny no eren adequades per a les noves situacions emergents, no tenien suficient potència per a aprofitar tots els nous recursos disponibles, ja que la productivitat que es podia aconseguir evolucionava de forma lineal i la capacitat de la tecnologia ho feia seguint la **Llei de Moore**, que prediu que cada 18 mesos es duplica l'increment de portes.

Tots aquests fets van comportar l'aparició de l'anomenat ***Productivity Gap***, conegut també com a **crisi de la productivitat.**

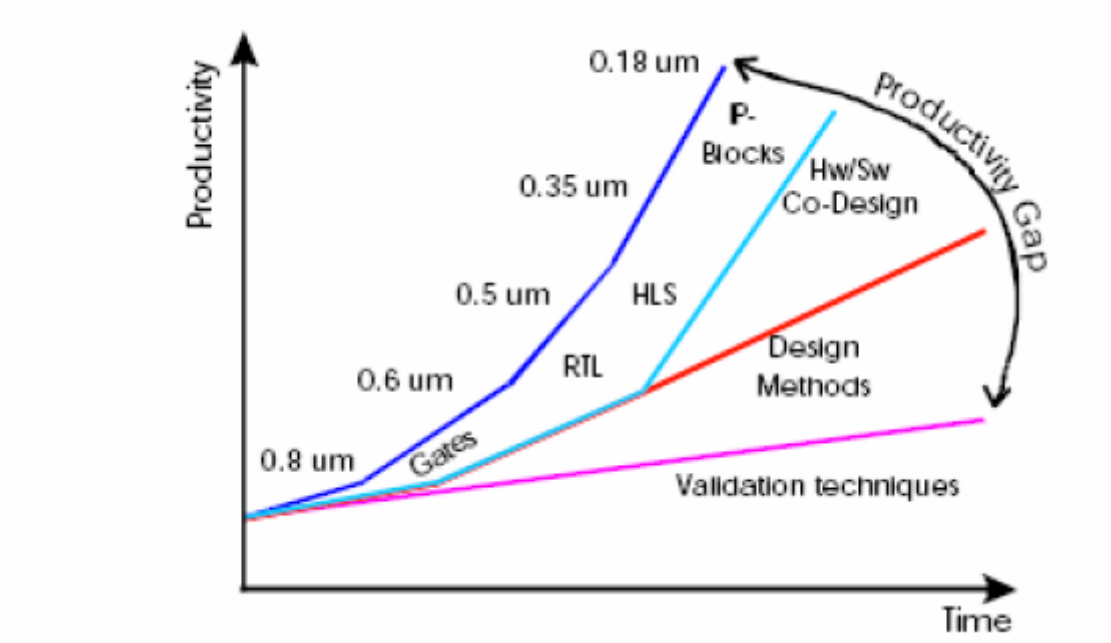



Figura 2.3: Evolució de la crisi de la productivitat. [LluísTeres06]

Per tal de reduir aquest forat, han aparegut noves i millors eines de disseny, com les EDA (E*lectronic Desing Automation*), i noves metodologies diferents del *bottom up* dels ASIC del principi dels 80 o del *top bottom* dels primers SoCs, centrades en el testeig i la verificació dels components, amb dissenys reutilitzables i la incorporació dels ***IP cores (Intellectual Property)*** ja testejats i verificats.

Aquests ***IP Cores*** són blocs lògics, dissenyats i testejats i verificats per algú, el qual té els drets de la propietat intel·lectual, que es poden fer servir en qualsevol sistema, ja sigui *Application-Specific Integrated Circuit (ASIC)* o un *Field Programmable Gate Array (FPGA).*

Aquests blocs arriben a ser components tan complexos com processadors, o controladors de diferents interfícies, poden ser memòries o *Asynchronous Receiver/Transmitter (UARTs),* per posar-ne alguns exemples.

Els ***IP Cores*** poden ser de diferents tipus:

- *Soft Cores*: *netlist* o codi HDL sintetitzable cap a diferents tecnologies. Són els més portables i, per tant, els més flexibles a l'hora d'incorporar-los al disseny del SoC/NoC.

- *Hard Cores*: *layout* específic cap a una tecnologia concreta (el *place & route* ja està fixat), normalment ja estan optimitzats per rendiment, àrea i/o consum. Són els menys portables i flexibles, però són els millors per a aplicacions de *plug-and-play*.
- *Firm Cores*: *netlist* genèrica, relativament independent de la tecnologia, ja que tenim unes restriccions de *place & route*. Són molt similars als *hard cores*, però una mica més portables cap a altres tecnologies.

Finalment, aquests elements s'integren dintre d'un mateix sistema per crear el SoC/NoC.

Aquestes millores en la tècnica de disseny i les noves eines de desenvolupament més eficients es recolzen en unes *best practices* que redueixen el forat de la producció enfront de la tecnologia (que mai no ha quedat eliminat del tot). Aquestes *best practices* fan incidència en la reutilització dels components (fer servir *IP Cores)*, el codisseny HW/SW i fer servir sempre eines de verificació i comprovació millors.

El fet de fer servir *IP Cores* també va comportar que s'haguessin d'estandarditzar les interfícies, tant entre els *IP Cores* amb els sistemes que els volguessin incorporar com entre els mateixos *IP Cores.*

Els SoCs/NoCs han hagut d'enfrontar-se a diferents problemes com poden ser el gran increment en la dificultat del *placement* de les línies que han d'arribar a tots els components del sistema (alimentació, rellotge, reset, entre d'altres), el soroll que les pròpies línies podien produir entre si, l' anomenat *crosstalk*, o problemes amb les freqüències dels rellotges. Però tot i això, els SoCs/NoCs han aconseguit ser una realitat, i poden fer servir milions de transistors, augmenten les prestacions dels sistemes, redueixen costos, ja que es fa servir menys silici per al mateix sistema, i tenen una verificació més senzilla i ràpida a nivell d'IP.

Aquests objectius s'han assolit també gràcies als canvis que ha provocat l'evolució del flux de disseny, des del sistema en cascada o *waterfall* emprat en els ASICs, en el qual s'avançava a cada pas sense tornar enrere i el treball de desenvolupament de software s'iniciava en finalitzar el hardware, fins als models en espiral emprats pels SoCs/NoCs.

El mètode *waterfall* va ser factible mentre la quantitat de portes que es podien integrar no era gaire elevada, sobre uns 100K amb una tecnologia per sobre de les 0,5 µ.

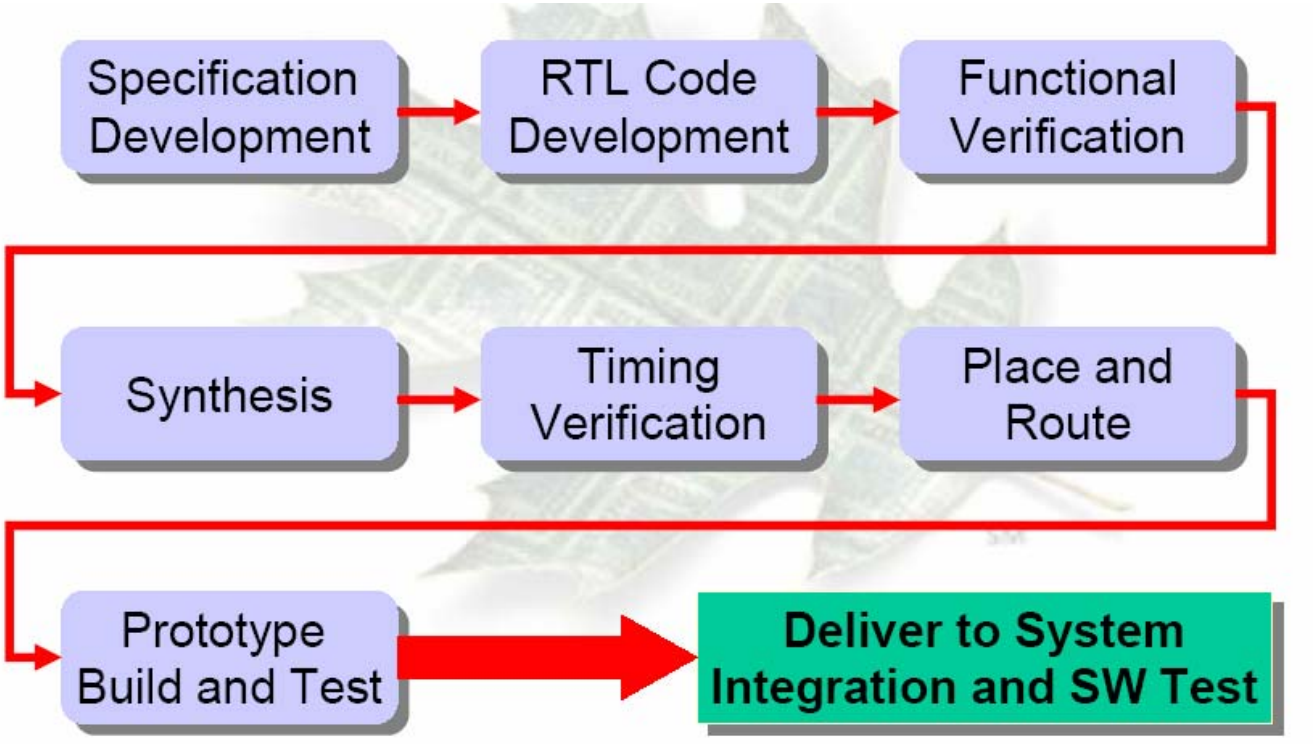



Figura 2.4: Model *WaterFall* de flux de disseny.[SPLevitan01]

En canvi, el sistema en espiral promou el codisseny hardware/software, el paral·lelisme entre la verificació i la síntesi i un cicle de disseny iteratiu/evolutiu incremental.

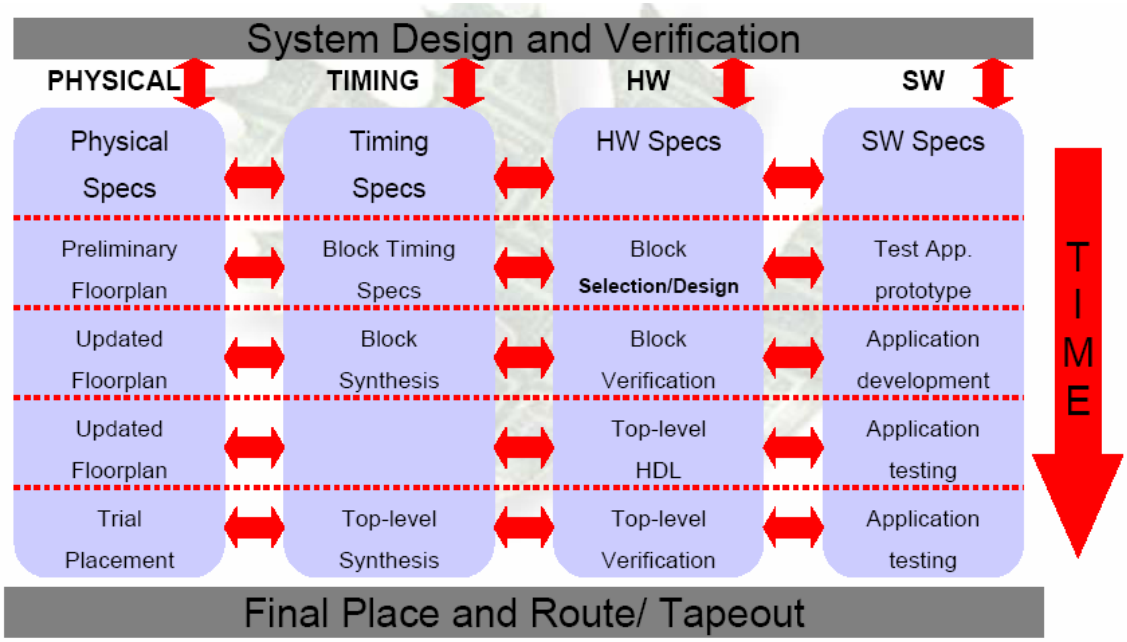



Figura 2.5: Sistema de flux de disseny en espiral. [SPLevitan01]

En la figura 2.6 podem veure un resum de l'evolució dels ASICs fins als NoCs.

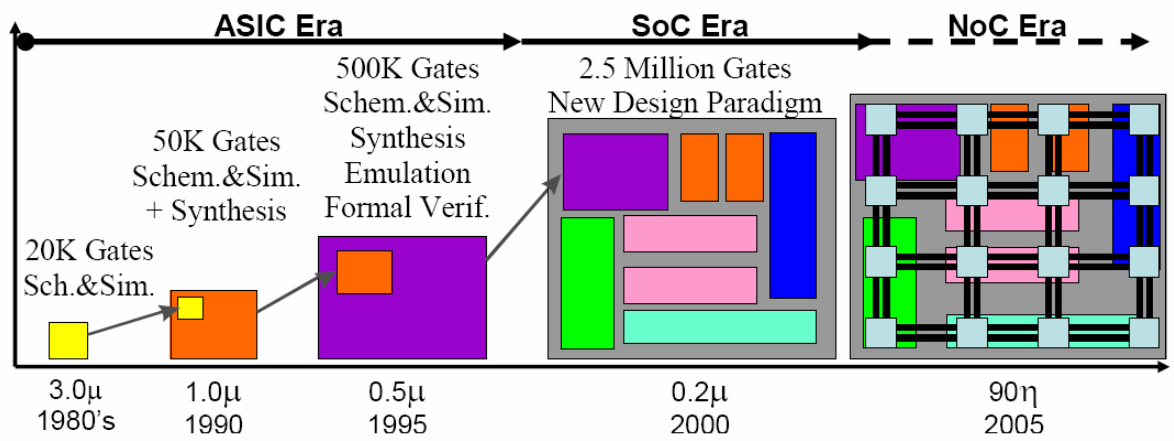



Figura 2.6: Evolució de la tecnologia.[LluísTeres06]

## 2.2 Conceptes bàsics

El concepte de **Network-On-Chip** va aparèixer com una necessitat per poder escalar els components que es poden encabir dintre d'un mateix sistema, a causa que el bus dels SoCs presenta grans problemes d'escalabilitat per a un nombre elevat de nodes.

Així doncs, en les NoCs trobem dos tipus diferents d'elements: un són els ***resources*** (recursos) del sistema que es vol integrar, i l'altre són els ***switch/routers.***

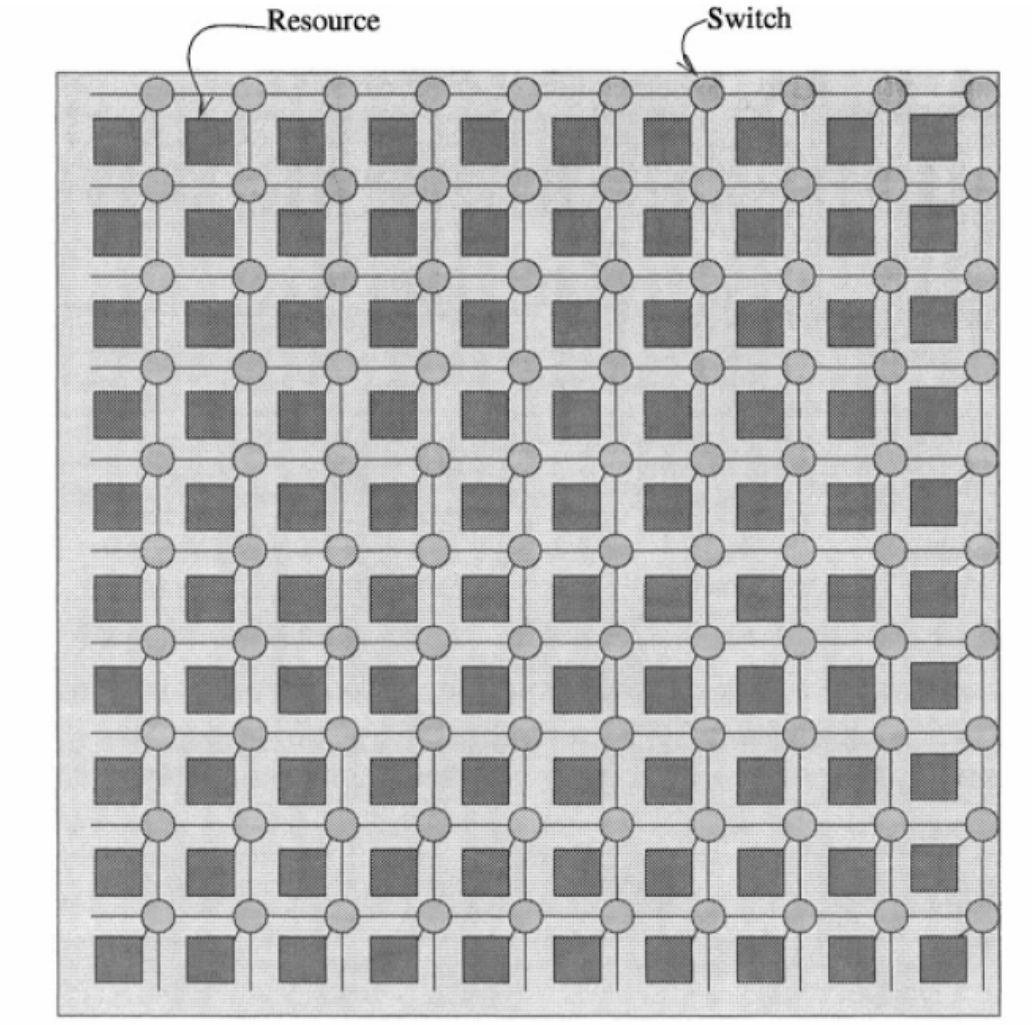



Figura 2.7: Nodes d'una NoC, on cada node té un recurs i un *switch/router*. [Jantsch03]

En les NoCs les topologies de connexió tenen una gran importància, entre les quals la més típica és la topologia de **malla** o ***Mesh***.
Aquesta topologia defineix que cada *switch/router* que forma part de la NoC està connectat a 4 *switch/router* més: nord, sud, est i oest. I cada recurs, per la seva banda, està connectat a un únic *switch*.

En la figura 2.8 trobem un exemple de NoC amb topologia *Mesh* i diferents recursos.

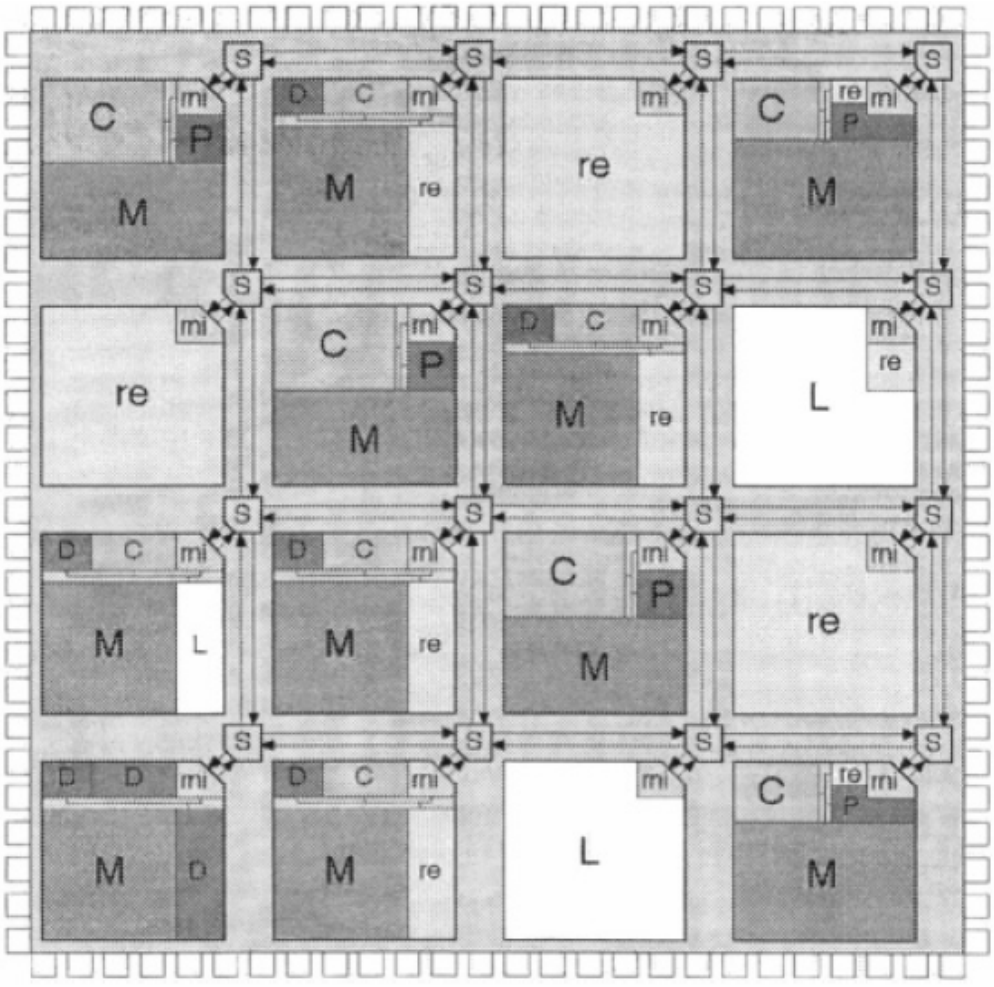



Figura 2.8: Exemple de NoC. S = Switch/Router. Rni = Resource network interface. P = Processador central. C = Cache. M = Memòria. D = DSP. Re = Lògica reconfigurable.L = Hardware dedicat específic. [Jantsch03]

La comunicació dintre d'una NoC es fa a través de paquets, i existeixen diferents mètodes de ***switching/routing***. Les tècniques més comunes de commutació són el ***packet switching***, on trobem les estratègies de ***store-and-forward***, de ***virtual cut-through*** i de ***wormhole*** [Rijpkema03], i el ***circuit switching.*** Per a més detalls, vegeu la figura 2.9.

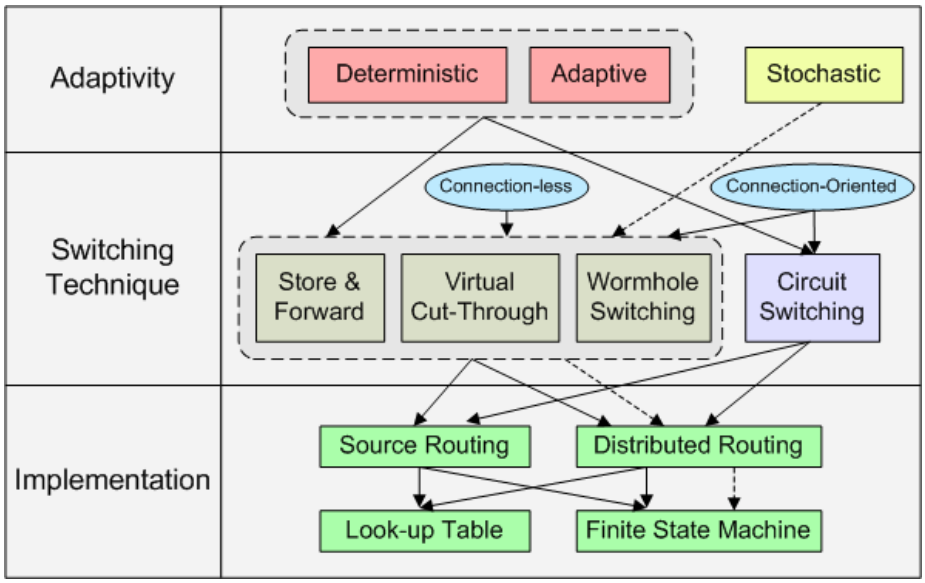



Figura 2.9. Tècniques d'encaminament i de commutació amb les seves possibles implementacions. [Ogras05]

## 2.2.1 Packet switching

La característica principal de totes les estratègies de commutació de paquets (***forwarding***) tipus *packet switching* és que tenen com a element principal de control de flux l'anomenat **flit**. El flit és la unitat mínima d'informació que viatja per la NoC, i per tant un paquet pot contenir un o més flits depenent de les dades que es volen transmetre/rebre.

### 2.2.1.1 Store-and-forward

La tècnica de ***store-and-forward*** [Kumar02] és un protocol de ***packet switching*** en la qual cada node guarda el paquet complet i el torna a enviar basant-se

en la informació que conté l'encapçalament. En aquest mètode el paquet es pot perdre o saturar en la cua depenent de si el *router* del node no té prou espai de *buffer* per rebre'l.

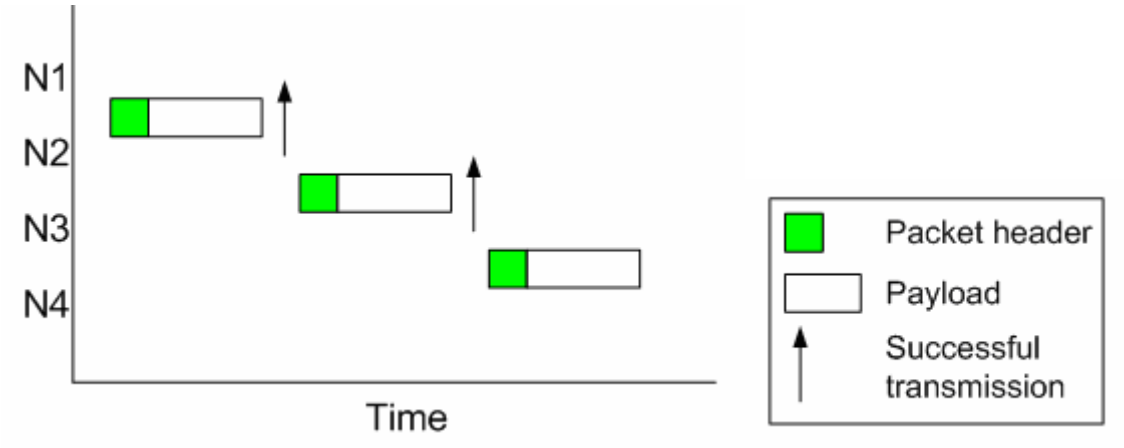



Figura 2.10: Esquema de transmissió de la tècnica de *store-and-forward*. [Castells06]

### 2.2.1.2 Wormhole

La tècnica d'encaminament *Wormhole* combina el *packet switching* amb la qualitat de l'*streaming data* del *circuit switching*, per tal d'obtenir un mínim *latency/delay*. [Benini02]

En el *wormhole switching* cada paquet es fragmenta en flits. Aquests tenen diferents significats, i poden ser de diferents tipus:

- **Header flit**: reserva el canal d'encaminament de cada *router*. Pot contenir, per exemple, l'adreça d'origen i la de destinació del paquet.
- **Body flit**: segueix el *header flit* i pot contenir, per exemple, dades.
- **Tail flit**: allibera la reserva del camí que havia fet el *header flit* en iniciar l'enviament del paquet.

Una propietat de l'encaminament *wormhole* és que no necessita emmagatzemar el paquet complet per encaminar el següent paquet. Aquesta tècnica no només redueix el retard que es produeix en la tècnica *store-and-forward,* sinó que també necessita molt menys espai de *buffer*.

Per contra, el fet que un mateix paquet pugui estar en més d'un node encaminador, l'efecte de la pèrdua o saturació del paquet afectarà més d'una línia, ja que es propagarà pel "cuc".

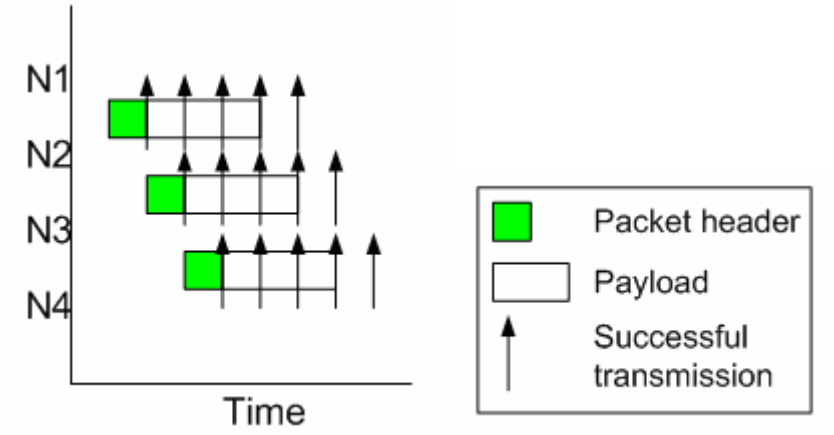



Figura 2.11: Tècnica *Wormhole*. [Castells06]

Així, un qüestió important alhora dissenyar un *router* és intentar reduir al màxim possible la quantitat d'àrea de *buffer* sense perdre les prestacions que es requereixen [Coenen06]. Hi ha dos aspectes principals a considerar sobre els *buffers*:

1. La seva mida: bàsicament amplada i profunditat que defineixen el nombre de paraules que es poden encabir.
2. La seva posició dins del *router*: en la discussió sobre *input* versus *output buffers*, per a iguals execucions la longitud de la cua en un sistema amb *output port buffering* sempre és més curta que la longitud de cua d'un sistema equivalent amb *input port buffering* [Rijpkema03]. Això és així, ja que en node encaminador amb *input buffering* un paquet queda bloquejat si esta en cua darrera d'un paquet el port *output* del qual està ocupat (head-of-the-lineblocking).

Un altre aspecte important que cal remarcar és que incrementar la mida del *buffer* no és una solució per evitar la congestió [Kumar02].
En el millor dels casos, endarrereix l'inici de la congestió, ja que el *throughput* no ha augmentat. S'experimenta una millora marginal amb relació al *overhead* del consum i l'àrea que es requereixen. D'altra banda, les cues són útils per absorbir les ràfegues de trànsit, ja que anivella els pics. En [Hu04b] es proposa que la mida del *buffer* sigui especifica per a un disseny concret, tot i que una aproximació general optimitzada és molt més flexible, i es pot ajustar a més aplicacions. A més, combinat amb [Hu04b], en [Saastamoinen03] es descriu com són d'importants les característiques de trànsit generades per l'aplicació per tal d'obtenir una mida de *buffer* millor. A [Saastamoinen03] s'arriba a la conclusió que depenent dels recursos físics de la plataforma, és important escollir registres o memòria dedicada per implementar el *buffer* de les cues. D'altra banda, en [Rijpkema03] *input queuing* i *output queuing* es combinen per aconseguir un *output queuing* virtual. La decisió de seleccionar un *input queuing* tradicional o un *output queuing* virtual depèn d'aspectes de nivell de sistema com la topologia, la utilització de la xarxa i el cost de cablejat global. A més, un altre aspecte significatiu és triar un espai de *buffer* centralitzat compartit entre ports d'*input* i *output* múltiples o FIFOs dedicades distribuïdes. Generalment, es considera que les implementacions de *buffer* centralitzades són cares en àrea a causa de l'*overhead*en la implementació del control, i es converteixen en colls d'ampolla durant els períodes de congestió.

### 2.2.1.3 Virtual cut-through

El mètode d'encaminament *virtual cut-through* té un funcionament semblant a la tècnica del *wormhole.* La diferència rau en el fet que abans de l'enviament del primer flit del paquet, el node s'espera per garantir que el següent node del camí pugui acceptar el paquet sencer. Així, si el paquet es perd, el problema es focalitza en el node en concret i no es produeix cap bloqueig de cap línia.

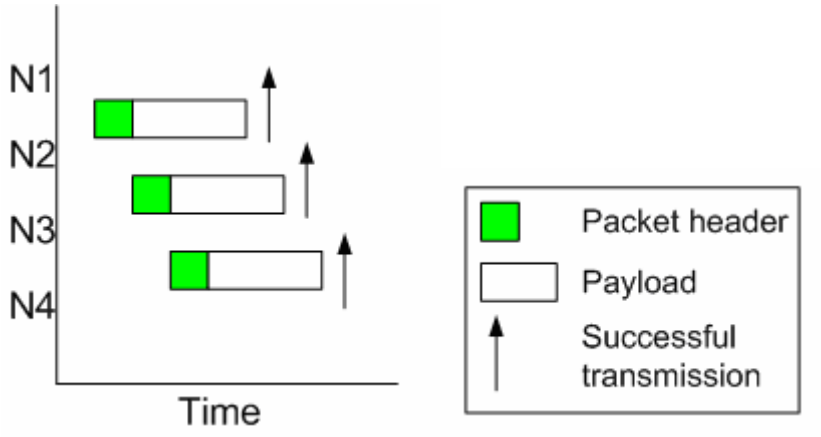



Figura 2.12: Tècnica *virtual cut-through.* [Castells06]

## 2.2.2 Circuit switching

En el cas de l'altra tècnica de *switching*, el *circuit switching*, el camí queda completament reservat d'origen a destinació, fins que l'enviament de la dada finalitza totalment.

En canvi, el *packet switching*, com hem vist, l'encaminament el fa entre dos nodes veïns sense reservar tot el camí d'origen a destinació, fent servir els flits. [Guerrier00]

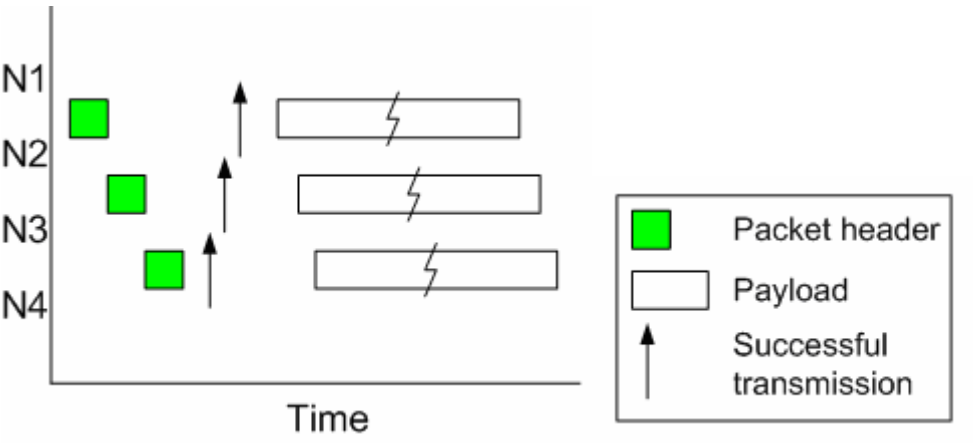



Figura 2.13: Tècnica de *circuit switching*. [Castells06]

# CAPÍTOL 3

## Disseny i metodologia de treball

## 3.1 Flux de disseny i metodologia de treball

Aquest apartat se centrarà a veure les eines de disseny que s'han fet servir per crear les descripcions hardware dels mòduls del *router* amb arquitectura *wormhole*, així com la metodologia i el flux de disseny que s'han seguit durant el projecte.

El projecte de creació d'una arquitectura *router* amb *wormhole* per una topologia 2D-Mesh partia d'un projecte anterior creat per l'enginyer Jaume Joven Murillo, el qual consistia en un *router* amb *circuit switching* amb un algorisme d'encaminament determinista XY, i s'ha adaptat per convertir-ho en un *router* amb *packet switching wormhole*.

Per tal de dissenyar els mòduls necessaris del router wormhole, s'ha seguit una metodologia d'implementació incremental i evolutiva. Inicialment s'ha dissenyat i testejat tots els submòduls que formaran el router de forma independent, i acte seguit s'han interconnectat per implementar, testejar i extreure els resultats del disseny complert del router.

Durant la primera etapa, anomenada **aprenentatge i disseny**, es van definir dues subfites.
La primera subfita a assolir, que podríem anomenar **subfita d'aprenentatge**, consistia a:

- Entendre l'arquitectura d'una NoC amb topologia *Mesh circuit switching*.
- Entendre el llenguatge de descripció hardware que finalment s'esculli per definir els mòduls del *router*.
- Estudiar les estratègies de *switching* i entendre el funcionament de la tècnica *wormhole*.

Un cop completat aquest primer grup d'objectius, comença el treball d'estudi i disseny dels mòduls del que anomenarem **subfita de disseny.**
Aquesta part de la primera etapa se centra a:

- Definir els mòduls que formaran el *router*.
- Estudiar les necessitats de les diferents parts a implementar.
- Crear les màquines d'estats finits que controlen els flits del *router wormhole*.

Com a resultats d'aquesta primera etapa del projecte, tenim els esquemes de capses del *router* i els seus components, així com els diagrames d'estat de les màquines de control.

Abans d'haver finalitzat la subfita de disseny de l'etapa d'aprenentatge i disseny, es va iniciar la segona etapa del projecte, anomenada etapa d'**implementació**.

En aquesta segona fase es van crear els fitxers descriptius del hardware del *router*. Les tasques bàsiques van ser:

- Afegir les cues/FIFO, necessàries per a la tècnica d'encaminament *wormhole,* al mòdul de sortida i la seva unitat de control. Així s'obté un *output queuing*. Amb les unitats de control se sap quan la cua/FIFO és buida o plena, de manera que quan la cua sigui plena no s'hi enviaran més dades ni se'n voldran extreure quan sigui buida. A més, les unitats de control han de generar els *acks* quan s'hagi guardat a la cua el paquet, o el *request* quan es vol enviar un paquet al següent *router*.
- Afegir la detecció dels bits de codificació (flitType=(00,01,10)) i la màquina d'estats associada.

El resultat de la implementació són els fitxers HDL del *router*.

La tercera etapa del projecte, **simulació i verificació**, pràcticament conviu alhora amb l'etapa anterior d'implementació, ja que aquesta fase treballa amb els fitxers HDL obtinguts, i els resultats parcials poden afectar l'etapa precedent.

Així doncs, les tasques consistien a:

- Crear els arxius de verificació, o *testbeanch*, dels mòduls HDL.
- Comprovar que funcionen correctament i que tenen el funcionament desitjat.
- Crear amb l'eina de síntesi escollida els fitxers RTL.

Finalment, els resultats de l'etapa de simulació i verificació es fan servir a l'última fase, l'etapa de **resultats.**

Aquí s'han analitzat els resultats obtinguts fins ara per extreure'n conclusions sobre l'àrea utilitzada pel *router* i possibles millores futures.

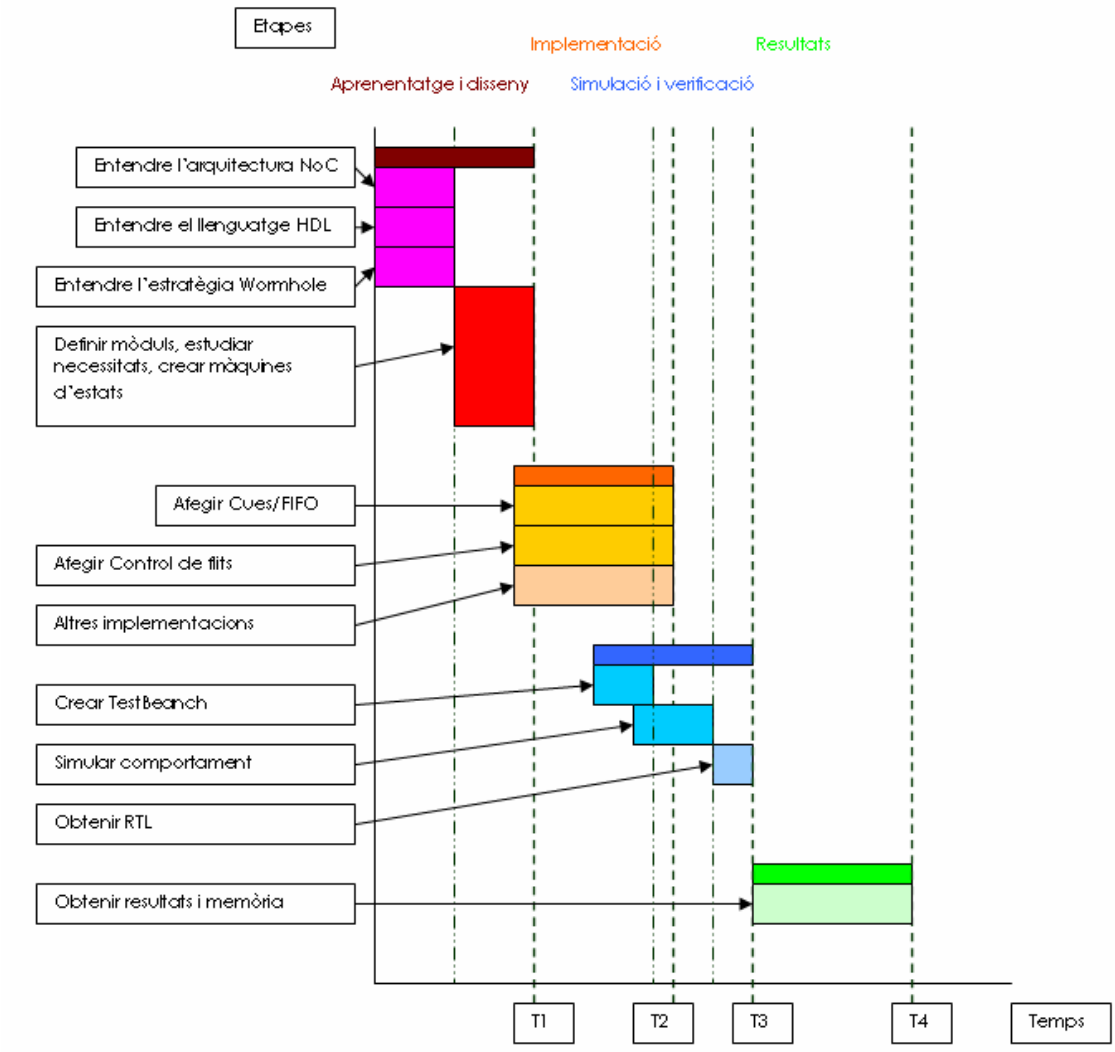



Figura 3.1: Esquema de la metodologia de disseny.

Un cop vista la metodologia seguida en el projecte, el flux de disseny queda de la següent forma:

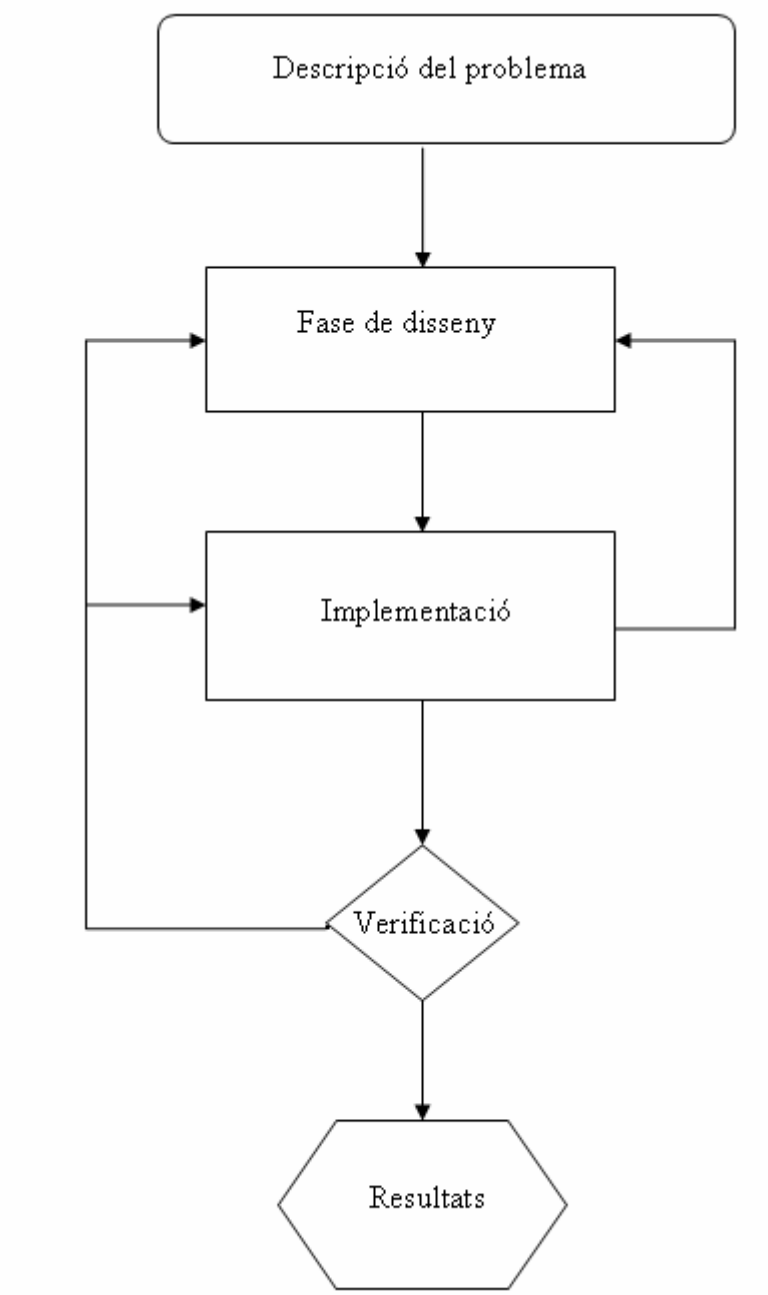



Figura 3.2: Flux de disseny del *router wormhole*.

Pel que fa als llenguatges descriptius de hardware que s'han utilitzat durant el projecte, es va partir inicialment de dues opcions:

- Verilog
- VHDL

Tots dos llenguatges estan acceptats (i, de fet, ho són) com a estàndards per descriure hardware, tot i que el VHDL és un llenguatge més orientat al desenvolupament i la investigació que no pas el Verilog, que va més enfocat a la síntesi.

El VHDL conviu en àmbits més aviat universitaris i el Verilog està molt present en el món laboral.
En la figura 3.3 podem observar una comparativa entre els dos estàndards i les petites diferències existents.

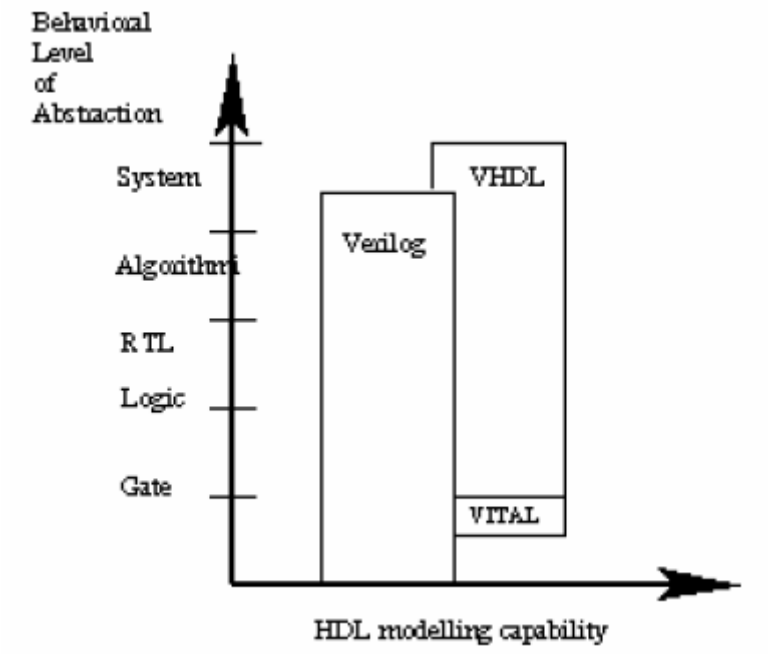



Figura 3.3: Capacitats de modelatge (Verilog, VHDL).

Tots dos llenguatges tenen prou potència per poder satisfer les necessitats que requeria el projecte, i el llenguatge escollit va ser Verilog.
En el punt 3.2 del capítol veurem una explicació més detallada dels llenguatges descriptius de hardware.

Entre les eines disponibles per a simulació i síntesi, es podia triar entre ModelSim i Quartus per a la definició i simulació dels mòduls, i Synplify per a la síntesi.
Pel que fa a la simulació, l'eina escollida va ser ModelSim perquè disposa d'una interfície amb l'usuari més acurada que no pas Quartus, i el fet que ja conegués aquesta eina per haver-hi treballat anteriorment, va acabar de desequilibrar la balança.

## 3.2 Llenguatges descriptius de hardware (HDL)

En aquest apartat del capítol es farà una introducció i una valoració tècnica dels llenguatges descriptius de hardware, i més concretament de Verilog.

Per tal de poder aprofitar les disponibilitats de recursos tecnològics cada vegada més potents, van aparèixer els llenguatges descriptius de hardware. Amb aquests llenguatges es poden modelar els comportaments dels sistemes digitals, i tenen uns objectius molt concrets:

- Simulació del circuit
- Síntesi automàtica del circuit
- Verificació formal
- Especificació i documentació

Tenen moltes característiques heretades dels llenguatges de creació de software, però amb certes propietats pròpies. [LluísTeres98]
Entre les similituds que existeixen entre els llenguatges descriptius HW i els SW tenim, per exemple:

- Capacitat per descriure funcionalitat (tipus de dades i sentències)
- Modularitat: estructuració del codi.
- Ús de llibreries: reutilització del codi.

En canvi, la principal diferència consisteix en:

- El software s'executa en un processador, és seqüencial.
- El hardware es simula i es sintetitza, és concurrent.

Els dos llenguatges de descripció de hardware (HDL) que es fan servir majoritàriament en l'actualitat són el Verilog i el VHDL.
El VHDL va convertir-se en l'estàndard IEEE 1076 l'any 1987. El 1994 es va fer una segona revisió important del llenguatge, i fins ara se n'han anat fent petites revisions. Per la seva banda el Verilog va néixer el 1985 com a

llenguatge propietari de la companyia Gateway, però el 1990, en formar-se **OVI** (*Open Verilog International*), va fer-se de domini públic, de manera que es va permetre a altres empreses que poguessin fer servir Verilog com a llenguatge, amb l'objectiu d'augmentar-ne la difusió. Finalment, l'any 1995 va convertir-se en l'estàndard IEEE 1364.

Tot i ser dos estàndards que tenen el mateix objectiu i que compleixen amb els mínims requerits a un llenguatge de descripció (abstracció del comportament i estructura a nivell de hardware), tots dos presenten unes certes característiques pròpies que els diferencien.

Com s'ha comentat en el punt 3.1, hi ha una petita diferència en la capacitat de modelatge, ja que Verilog dóna més facilitat per descriure nivells d'abstracció baixos propers al nivell de porta, mentre que el VHDL proporciona més facilitat per descriure comportaments d'algorismes i el nivell de sistema. Vegeu la figura 3.3.

Pel que fa a la reutilització, tots dos llenguatges tenen les seves pròpies opcions, encara que el VHDL proporciona els *package* semblants als *package* software i, en canvi, Verilog necessita crear *include* per tal de poder aconseguir el mateix objectiu. Aquestes eines, juntament a unes *best practice* de desenvolupament, com pot ser la parametrització de les variables, faciliten la reutilització del codi.

Pel que fa a la sintaxi del llenguatge, Verilog incorpora operadors unaris de gran utilitat que VHDL no conté. Fora d'aquí la seva sintaxi és molt semblant amb els mateixos operadors.

A nivell d'aprenentatge, normalment es diu que Verilog és més fàcil d'assimilar, ja que és un llenguatge més compacte, ocupa menys línies per descriure el mateix. També cal dir que existeixen altres motius que el fan una mica més senzill d'entendre, com per exemple el fet que VHDL és un llenguatge fortament tipat, que el fa molt robust i potent per a usuaris avançats, però més carregós per a usuaris principiants.

Com a conseqüència d'aquestes propietats, i tenint en compte que com a objectius i resultats del projecte es volien obtenir esquemes RTL i que el *router* del qual es partia estava descrit amb Verilog, el llenguatge triat, doncs, va ser Verilog.

## 3.3 Verificació

Per tal de verificar el bon disseny i la correcta funcionalitat de l'arquitectura cercada es crearan arxius de verificació o testbenchs de cada component. Aquests arxius són de vital importància en el projecte, i de les seves execucions que simulen el comportament lògic dels components s'haurà d'anar extraient informació cabdal per a la depuració del disseny i de la seva implementació.

Així doncs, la verificació autoalimentarà tant el disseny com la implementació del *router* i, alhora, garantirà la bondat del treball realitzat i, finalment, proveirà la principal informació que es farà servir per als resultats i per extreure'n possibles conclusions.

La verificació quedarà dividida en dues parts d'igual importància:

- Simulacions: verificaran el comportament dels mòduls creats.
- Síntesis: segona part de la verificació, encarregada d'obtenir els esquemes RTL.

Totes dues parts faran servir eines específiques que es comentaran a continuació.

### 3.3.1 Simulació i ModelSim

Durant el projecte s'ha fet servir l'entorn de desenvolupament i depuració ModelSim, per tal de descriure i verificar el comportament dels diferents components que formen el sistema.

Aquest simulador de hardware és una eina multiplataforma desenvolupada per Menthor Graphics que permet treballar sota sistemes Unix, Linux i Windows. També combina unes grans prestacions amb una GUI molt potent i intuïtiva.

Així doncs, amb ModelSim s'han creat els fitxers Verilog descriptius del components del *router*, i de la mateixa manera també s'han dissenyat els arxius de verificació. Amb ModelSim podíem accedir a totes les línies de cada mòdul que s'ha integrat en el sistema.

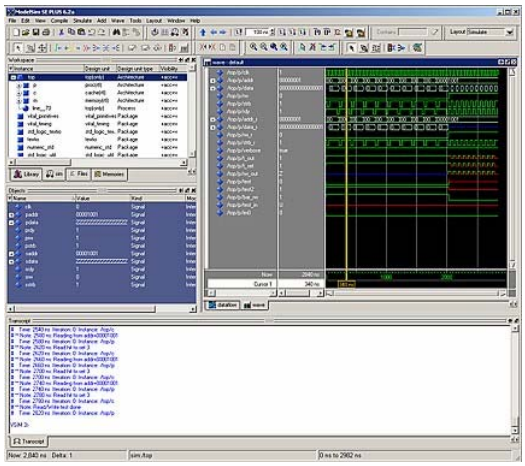


Figura 3.4: Captura de pantalla de l'eina ModelSim en la seva component de simulació en forma d'ones.

## 3.3.2 Síntesi i Synplify

Per finalitzar l'apartat de verificació es parlarà de Synplify, eina que s'ha utilitzat per compilar els fitxers Verilog dels components i fer-ne la síntesi per generar esquemes RTL de l'arquitectura especificada.

D'aquesta manera s'acaba de verificar el projecte i s'obtenen els diagrames de bloc dels components. Al capítol 5, Resultats, es podran veure exemples de captures d'aquesta eina.

# CAPÍTOL 4

## Implementació

En aquest capítol passarem a veure el treball realitzat sobre el *router*, com s'ha implementat, així com també destacarem els punts més crítics i interessants del treball.

Per tal de tenir una base sobre la qual explicar els components del *wormhole*, primer agafarem un coneixement bàsic sobre el funcionament d'un *router*.

El *router* és l'element comunicatiu principal dintre de la NoC, i en una topologia de tipus *Mesh* en 2D en què cada node està comunicat amb els seus veïns, implica que cada *router* tingui un màxim cinc connexions, una per a cada direcció possible dintre de la *Mesh* (nord, sud, est, oest) més una per a la direcció amb el propi node (local).

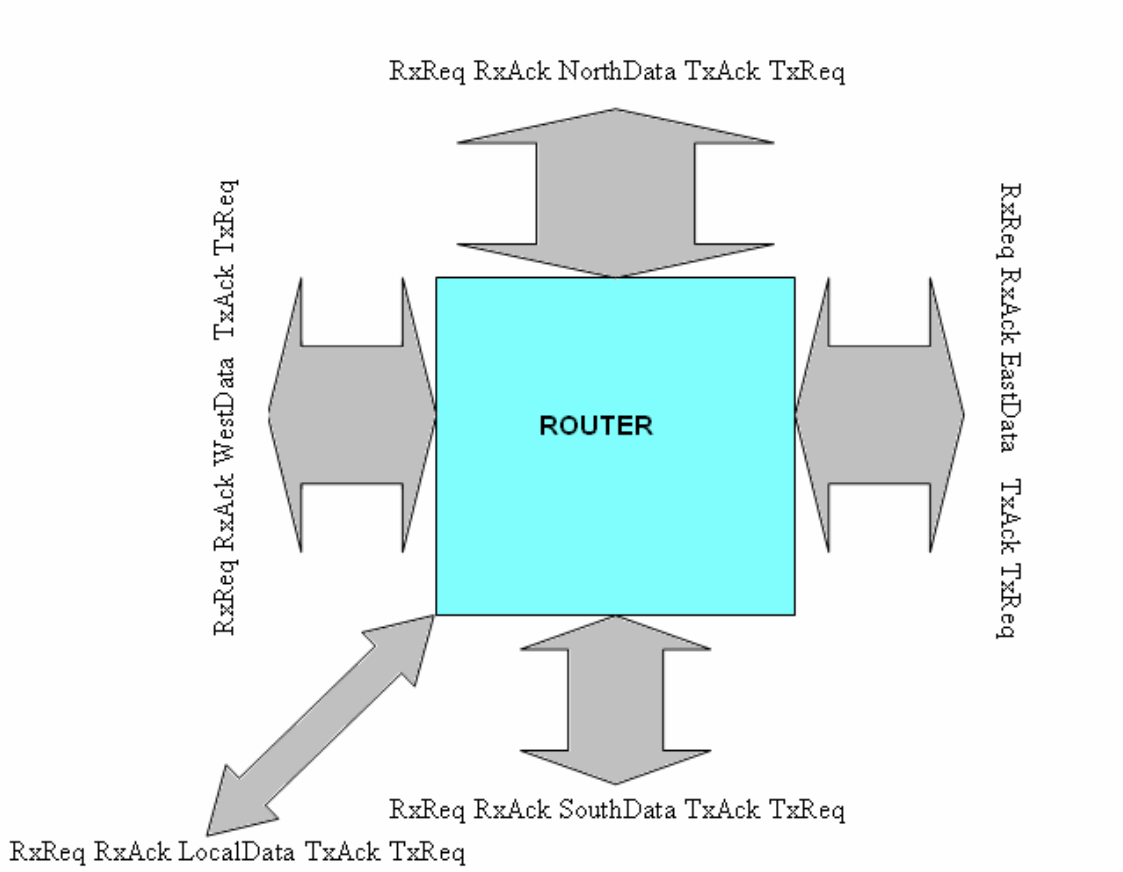



Figura 4.1: Esquema d'un *router* amb les seves connexions màximes possibles.

Existeixen excepcions en el nombre de connexions, ja que, depenent de la situació del node dintre de la malla de la NoC, pot no tenir alguna de les direccions. Si, per exemple, el node està situat a l'extrem superior esquerre (que d'ara en endavant anomenarem amb les coordenades 0,0), no

disposarà dels *paths* que van cap al nord ni cap a l'oest. En la figura 4.2 tenim un exemple visual sobre les excepcions en el nombre de connexions degudes a la posició del node.

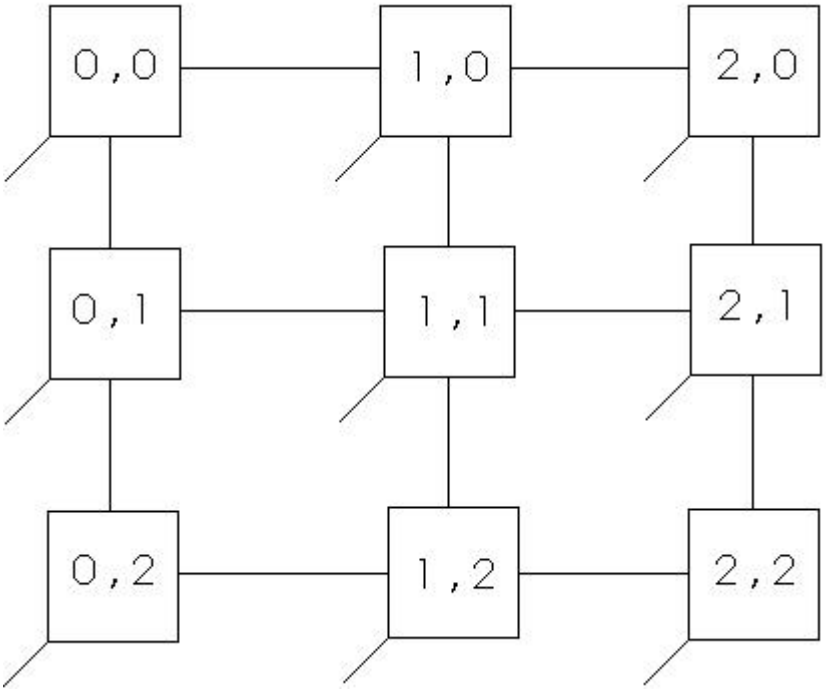



Figura 4.2: Exemple de *Mesh*.

Així doncs, veiem com els nodes (0,0), (2,0), (0,2) i (2,2) tenen dues direccions possibles amb els veïns, més la necessària per a la connexió amb el NiC del seu node. En canvi, els nodes (1,0), (0,1), (1,2), (2,1) disposen de 3 + 1 connexions, i només el node central (1,1) arriba al màxim de les connexions, ja que té els quatre veïns disponibles més la pròpia del NiC.

Altrament, també existeix un altre tipus d'excepció que pot fer modificar el nombre màxim de connexions dintre d'una *Mesh*, que és precisament la mida d'aquesta, ja que pot variar en el nombre de nodes dels quals disposa.

És precisament l'escalabilitat un dels factors que fa que l'estratègia de *packet switching* d'encaminament *wormhole* tingui avantatges considerables respecte a una tècnica de *circuit switching*. Aquest fet el veurem amb més detall en el punt 4.1.2 quan parlem sobre el format del paquet i la importància dels flits.

Pel que fa a les parts que formen el *router*, els seus principals components són els esmentats *paths* o camins, que, com ja hem dit anteriorment, cada *router* tindrà un camí per a cada connexió que necessiti.

De la mateixa manera, cada *path* està format per dos elements molt importants, dos mòduls de comunicació.

El primer mòdul és l'encarregat de fer el *routing* o encaminament, i el segon mòdul s'encarrega de rebre i transmetre les dades.

Pel que fa al mòdul d'**encaminament** cal destacar que es composa d'un sol component encarregat d'implementar l'algorisme corresponent de *routing*. Aquests algorismes poden ser de dos tipus diferents:

- Deterministes
- Adaptatius

Pel que fa als algorismes deterministes destaquem l'algorisme XY. Aquest mètode tracta d'encaminar sempre fent primer coincidir l'eix X de l'origen amb l'eix X de la destinació, per després moure l'eix Y per igualar les dues direccions.
La figura 4.3 ens mostra gràficament uns quants exemples de la tècnica XY.

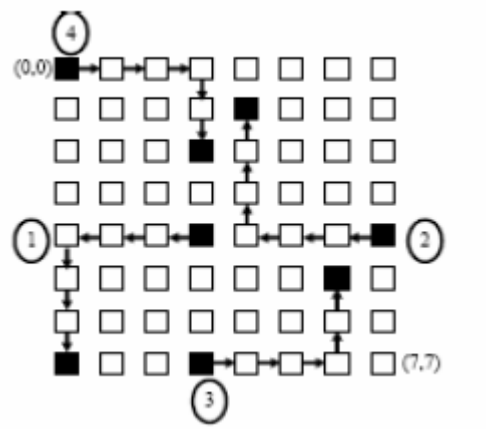



Figura 4.3: Algorisme XY [MELLOt].

Respecte als algorismes adaptatius, només comentarem el nom d'uns quants exemples, ja que l'estudi d'aquests ja podria ser un projecte diferent.
Algorismes adaptatius:

- West First
- North Last
- Negative First
- Odd even

L'altre mòdul del *path* és el mòdul **receptor/transmissor** de dades. Aquest mòdul funciona a mode de *switch*, rebent dades dels *paths* als quals està connectat i enviant-les després cap al camí que correspon.
Per tal de fer la tria del camí de sortida, aquest mòdul disposa, en un nivell lògic inferior, d'un component codificador de prioritats. En el nostre cas, l'ordre d'aquestes prioritats és de sentit horari amb la connexió local com a última preferència.
Prioritats:

I. Nord
II. Sud
III. Est
IV. Oest
V. Local

Així es prioritza la transmissió i circulació de les dades existents en la *Mesh*, per sobre de la possibilitat d'injectar nova informació a la malla. Aquesta tècnica afavoreix la circulació per intentar millorar l'amplada de banda disponible.

Per acabar, comentar que la comunicació es realitza amb un protocol de *handshake* que va d'origen a destinació final. El primer paquet s'encamina cap al *target* i, en arribar al *router* destinació, aquest envia el corresponent *ack*.
Quan el *router* origen rep aquest *ack*, llavors pot tornar a enviar la dada següent.

Aquesta manera de comunicar-se, d'origen a destinació final, pot provocar molts problemes, ja que si durant l'encaminament el cap del cuc no troba un camí disponible per continuar, s'espera, i provoca que la resta del cuc també quedi esperant i bloquegi així tots els nodes pels quals ha passat. Aquest és un altre aspecte que es millora en un *router wormhole*, com veurem en el punt següent.

## 4.1 Disseny del *router*

Ara que ja sabem com funciona bàsicament un *router*, explicarem el disseny concret implementat del *router* amb estratègia *packet switching wormhole*.

Per poder adaptar el *router* descrit al que busquem, haurem d'afegir una sèrie de mòduls addicionals que comportaran més control sobre les comunicacions i la forma de fer l'encaminament.

El primer canvi destacat que es presenta és el format del paquet amb la incorporació de flits, que explicarem en el punt 4.1.2.

Ara, però, parlarem sobre la incorporació d'una cua/FIFO a la sortida del mòdul receptor/emissor de dades.
Com hem vist quan es parlava sobre les tècniques de *packet switching*, els nodes necessiten guardar els paquets que van rebent. En el nostre cas, s'ha incorporat un mòdul anomenat CompleteFIFO a la sortida del mòdul emissor, que incorpora una cua FIFO amb una petita unitat de control, per als casos de Full i Empty.

Aquesta FIFO té associada una memòria RAM totalment parametritzable en termes de profunditat i amplada de paraula a través de l'ús de paràmetres en Verilog. En el nostre cas l'amplada de la paraula s'ha fixat a 34 bits d'acord amb l'amplada del flit, mentre que la profunditat s'ha anat variant per així extreure diferents resultats en termes d'area, tal i com veurem en el capítol de resultats.

En la taula següent, figura 4.4, podem veure una comparació entre diferents cues.

| | Cua A | Cua B | Cua C | Cua D |
|---|---|---|---|---|
| **Profunditat x amplada** | 8 x 34 | 16 x 34 | 32 x 34 | 256 x 34 |
| **Mida Kb** | 1.3 | 2.6 | 5.3 | 42.5 |

Figura 4.4: Taula de diferents cues.

Aquesta RAM associada és de doble port, ja que ha de permetre la lectura i l'escriptura al mateix temps.

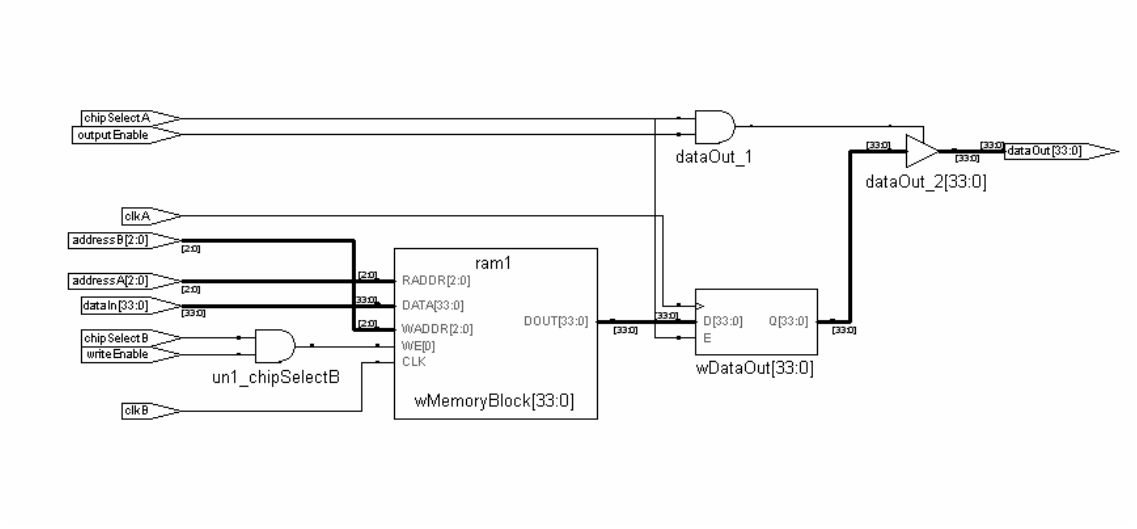



Figura 4.5: RAM simple de doble port.

Aquesta cua es troba a la sortida final del *router*, i tots els camins comparteixen la mateixa cua. Quan aquesta cua queda plena, no podem escriure més paquets i el *router* queda bloquejat.

Això ens porta cap al protocol de control de les comunicacions entre el mòdul emissor d'un *router* i el receptor del *router* veí. Aquesta part la veurem en el punt següent.

Finalment, aclarir que la implementació de la cua i la RAM han estat obtingudes de www.OpenCores.org, però ha estat adaptada i testejada per a aquest projecte.

### 4.1.1 Protocol de Control de flux: full duplex 4-phase handshake

El *router wormhole* fa servir un protocol de control de comunicacions entre *routers* de *handshake* simple de 4 fases *full duplex* no ambigu.

Aquest protocol consisteix en un diàleg entre el receptor i l'emissor el qual utilitza dues senyals per crear la demanda i acceptació del paquet, són les senyals de Request i Acknowledge.

En la figura 4.6 podem veure el funcionament del protocol.

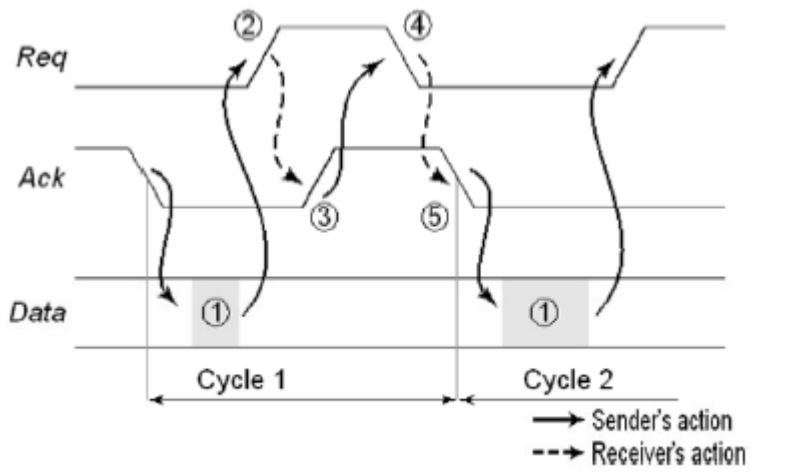



Figura 4.6: Protocol de 4 fases *Handshake full duplex*.

El mòdul emissor envia una petició per sol·licitar l'enviament de dades. Aquest fet es mostra quan la línia Req fa la transició de 0 a 1 en el moment en què les dades estan estables al bus de dades (pas 1). Aleshores, el receptor, quan està disponible, accepta les dades i envia un missatge de resposta positiva a l'emissor (pas 2, flanc de pujada de 0 a 1 d'Ack).

En el moment en què l'emissor veu el canvi en l'ack enviat pel receptor, canvia l'estat de la línia Req de 1 a 0 i retira les dades del bus (pas 3). Aquest fet provoca que el receptor, quan observa el flanc de baixada de Req, faci el mateix amb la seva línia Ack.

Arribats a aquest punt, el receptor ja està lliure per poder tornar a començar el diàleg i tornar a enviar dades quan ho necessiti.

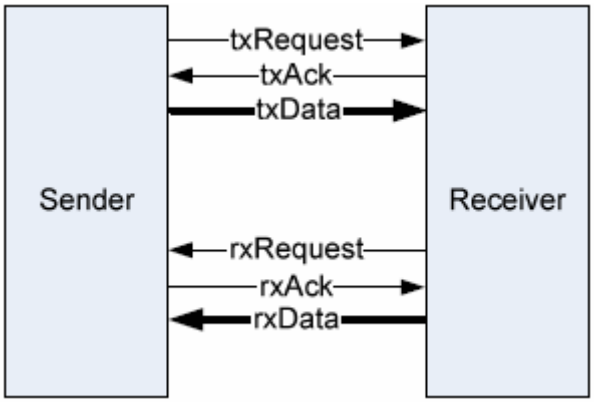



Figura 4.7: Connexions entre un emissor i un receptor amb *handshake full duplex*.

Com hem dit anteriorment, si la cua/FIFO del receptor és plena, aquest no enviarà l'ack de confirmació de recepció. Aquest fet farà que l'emissor no estigui disponible per enviar més dades i provocarà en cascada que tots els mòduls anteriors quedin bloquejats.

### 4.1.2 Format del paquet

En *packet switching* apareix el concepte de flit. Aquest governa el flux de funcionament de tot el *router*.

En el nostre cas, el format del paquet *wormhole* estarà format per una capçalera on anirà les adreces d'origen i destí i un cos on s'empaquetaran les dades del paquet.
El flit serà de 34 bits, on els dos primers bits estaran reservats per l'identificador dels flits i els 32 restants seran les dades pròpiament dites.

El fet de tenir dos bits per flits ens fa possible distingir fins a quatre casos possibles de tipus diferents de flits. Depenent de la codificació d'aquests bits tindrem 3 tipus de flits:

- Head Flit, serà el primer, amb codificació 00, i estarà format per 16

bits adreça origen i 16 bits adreça destinació.

- Body Flit, contindrà 32 bits amb les dades, amb codificació 01.

- Tail Flit, contindrà la resta de les dades del paquet i serà de codificació 10.

El cas de codificació 11 queda sense fer servir.

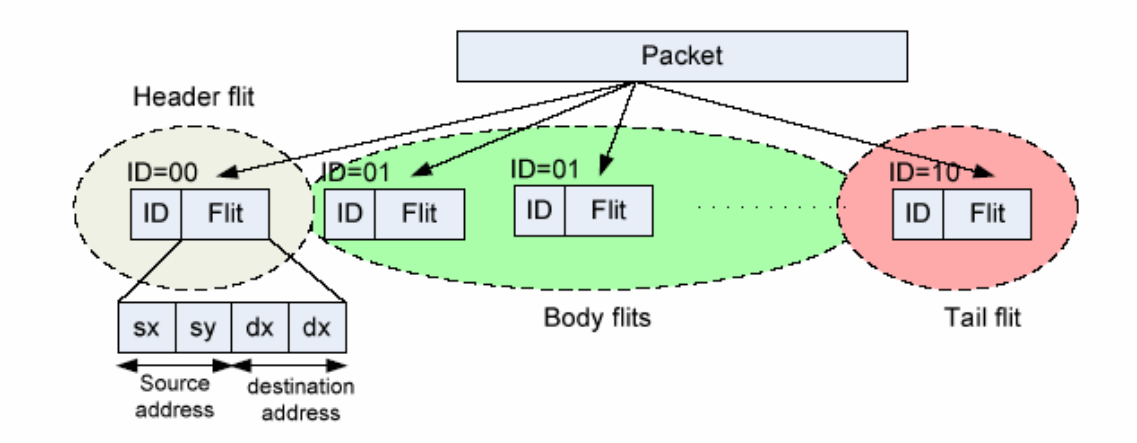



Figura 4.8: Tipus de flits.

Ara explicarem com s'ha fet la detecció dels flits i la unitat de control associada.

Com ja s'ha comentat anteriorment, l'element principal de control en el *packet switching* són els flits; depenent del tipus de flit que trobem les accions a realitzar seran diferents. El Header flit s'encarrega d'encaminar el cuc; a aquest flit el segueix un o diversos Body flits, que contenen les dades del paquet, i, finalment, arriba el Tail flit, que conté la informació per tancar i alliberar el camí.

En la figura 4.9 veiem la màquina d'estats que governa el *router* depenent dels flits d'entrada. És una màquina d'estats amb quatre estats i sis transicions possibles entre nodes. A l'estat HEAD, estat de sortida, la capçalera del flit d'entrada haurà de ser 00 per provocar un canvi d'estat. Quan això passi, l'estat *head* dividirà el paquet per obtenir les adreces origen i destinació, enviar-les al mòdul encarregat de l'algorisme d'encaminament i estimular els senyals necessaris perquè aquest mòdul actuï. Addicionalment, aquest estat

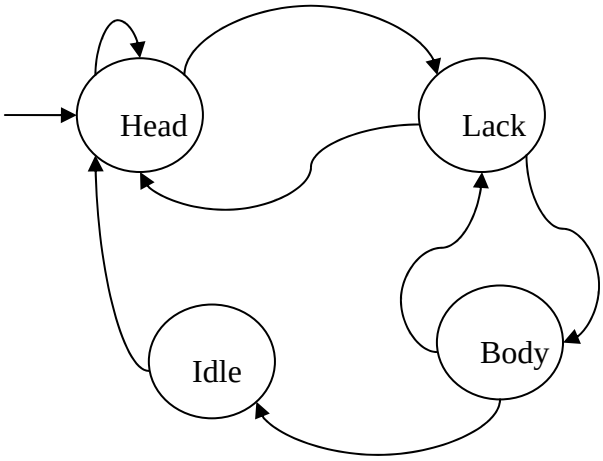
reserva el *router* per on passa, de manera que fa una reserva del camí des de l'origen fins a la destinació.

Després d'aquest pas, la màquina d'estats saltarà a un estat de comprovació que, depenent de l'estat del qual arriba, tornarà a saltar a l'estat següent correcte. Aquest estat de comprovació es diu LACK.

L'estat BODY incorporà tant el cas del flit Body com el del Tail. Si es tracta de la codificació 01, BodyFlit, la màquina de control s'encarrega d'enviar les dades al mòdul receptor/emissor per tal que segueixi el camí obert pel *head*.
Si, en canvi, la codificació del flit és 10, es tracta del cas TailFlit, i la màquina d'estats envia l'última part de les dades existent en el paquet i allibera el camí obert pel *head*. Finalment, el següent estat serà un altre cop l'estat LACK de comprovació.

Si durant l'estat Body arriba una petició de *request* amb codificació de flit 11 o 00, la màquina d'estats reconeix un error i salta a l'estat IDLE.

Aquest estat fa un reset del *router*, l'allibera i torna a saltar a l'estat inicial HEAD.



| Estat Inicial | Estat Final | Condicions | Sortida |
|---|---|---|---|
| HEAD | LACK | Flit = 00<br>Req = 1; | AddressX;<br>AddressY;<br>Data;<br>Enable = 1;<br>Ctrlack = 0; |
| HEAD | HEAD | Req = 0; | |
| BODY | LACK | Req = 1;<br>Flit = 01; | Data;<br>Crtlack = 0; |
| BODY | LACK | Req = 1;<br>Flit = 10; | Data;<br>Ctrlack = 1; |
| BODY | IDLE | Req = 1;<br>Flit = (00 \| 11) | Enable = 0; |
| LACK | HEAD | Crtlack = 1; | |
| LACK | BODY | Crtlack = 0; | |
| IDLE | HEAD | | Ctrlack = 0; |

Figura 4.9: Graf de la màquina d'estats i taula de transicions.

Aquest tipus de format representa una gran millora respecte al format d'un *circuit switching*, en el qual als bits de dades del paquet s'hi han d'afegir els bits d'adreces necessaris per representar l'origen i la destinació.

Amb el nostre Head flit de 34 bits disposem d'adreces de 16 bits per a l'origen i de 16 bits per a la destinació, 8 bits per a l'eix de les X i 8 bits per a l'eix de les Y. Així doncs, podem direccionar fins a $2^8$ x $2^8$ routers, és a dir, podem crear una *Mesh* de 256 x 256 nodes.

Per tant, aquest tipus de format proporciona una gran escalabilitat de la NoC amb molt poc increment de la mida del paquet.

Tot i això, es presenten també certs inconvenients amb aquest format quan la informació que volem enviar no supera la mida d'un flit de dades. En aquest cas, com que el primer flit que enviem conté les adreces i és el segon flit el que conté les dades, amb aquest format estaríem el mateix temps encaminant que emetent dades efectives.

### 4.1.3 Arquitectura del router per packet switching NoC

Finalment, veurem com està format el *router* de manera visual.

Com ja hem dit anteriorment, un *router* està format per cinc mòduls *path* o camins, un per a cada possible direcció que pugui tenir.

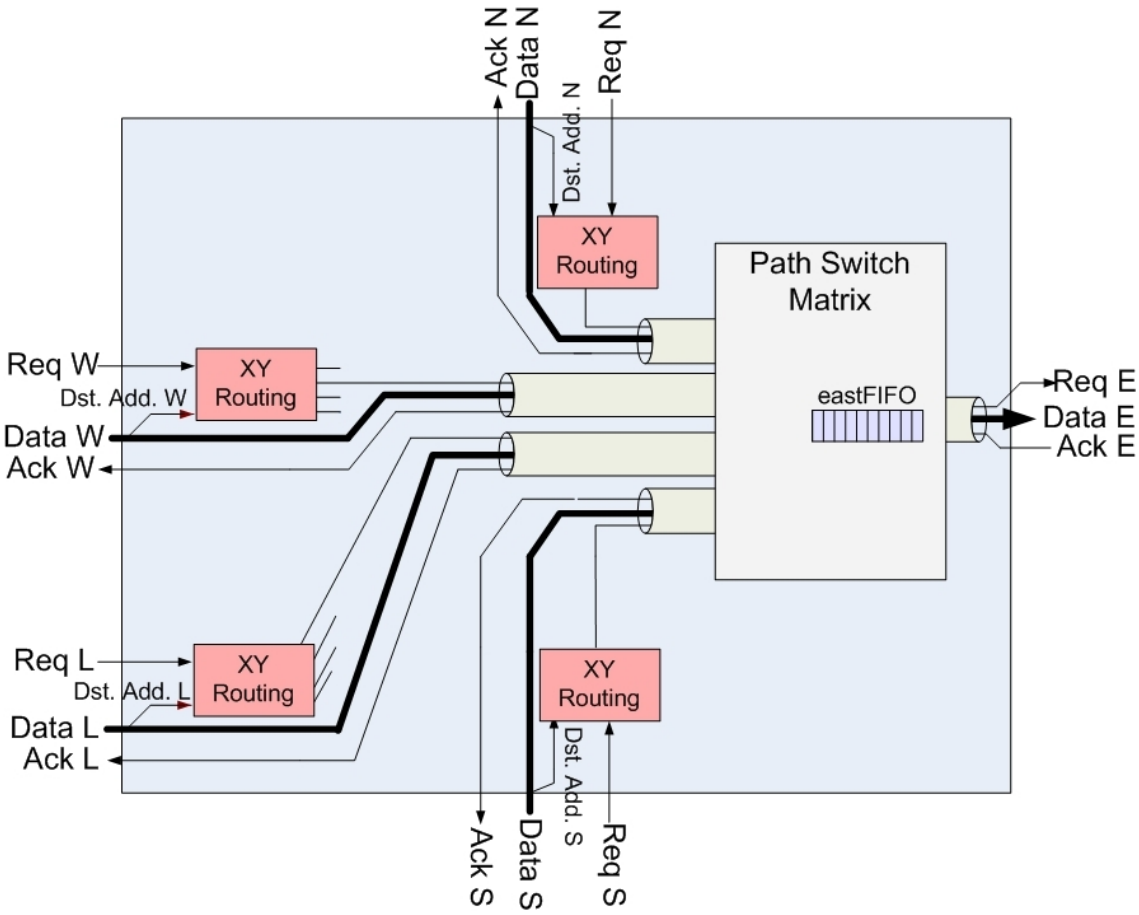



Figura 4.10: Esquema d'alt nivell d'un *router*.

Alhora, cada *path* està format per dos components, el mòdul d'encaminament, anomenat *MeshXYRoutingWormHole*, que incorpora els mòduls encarregats de realitzar l'algorisme XY d'encaminament, *XYRoutingWormHole,* i de detecció i control de flits, *FlitControlFSM*, i el mòdul receptor/emissor de dades, conegut com a *MeshSwitchMatrix*. Aquest mòdul incorpora els mòduls *CompleteFIFO* i *PathSwitchMatrixWormHole*, el qual s'encarrega de la tria de prioritats i la resposta dels missatges.

En les figures 4.11 i 4.12 veiem els mòduls *MeshXYRoutingWormHole* i *MeshSwitchMatrixWormHole*.

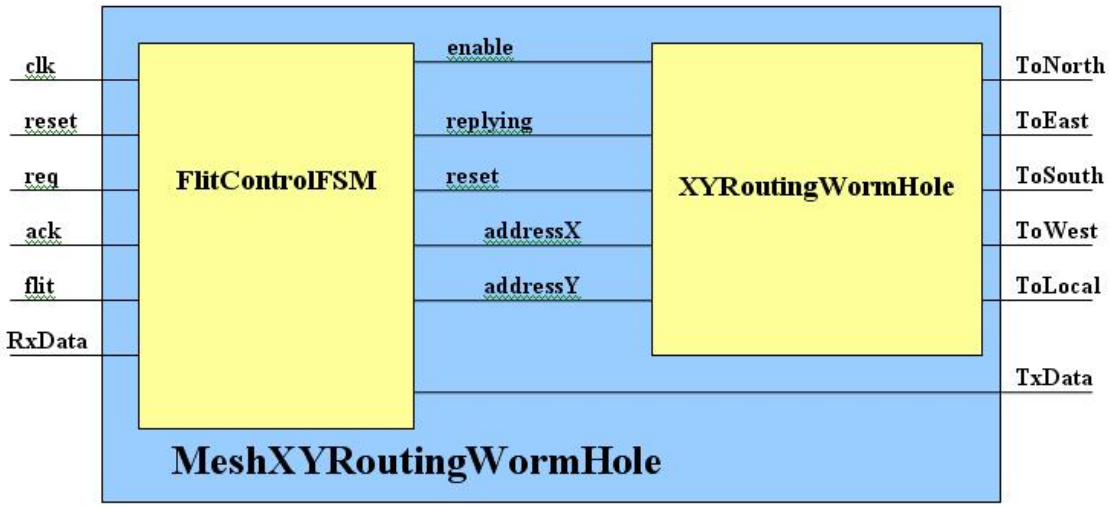



Figura 4.11: *MeshXYRoutingWormHole*.

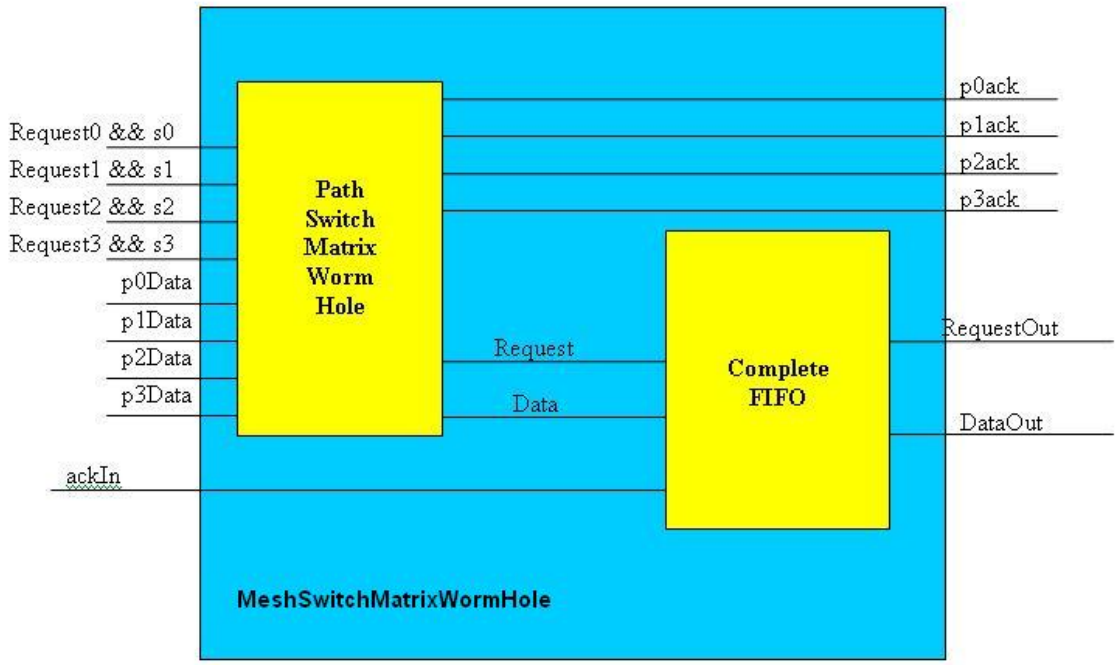



Figura 4.12: *MeshSwitchMatrixWormHole*.

Per últim, en la figura 4.13 veiem el mòdul *PathSwitchMatrixWormHole* format amb els mòduls *PriorityEncoder, DeMux* i *Mux*.

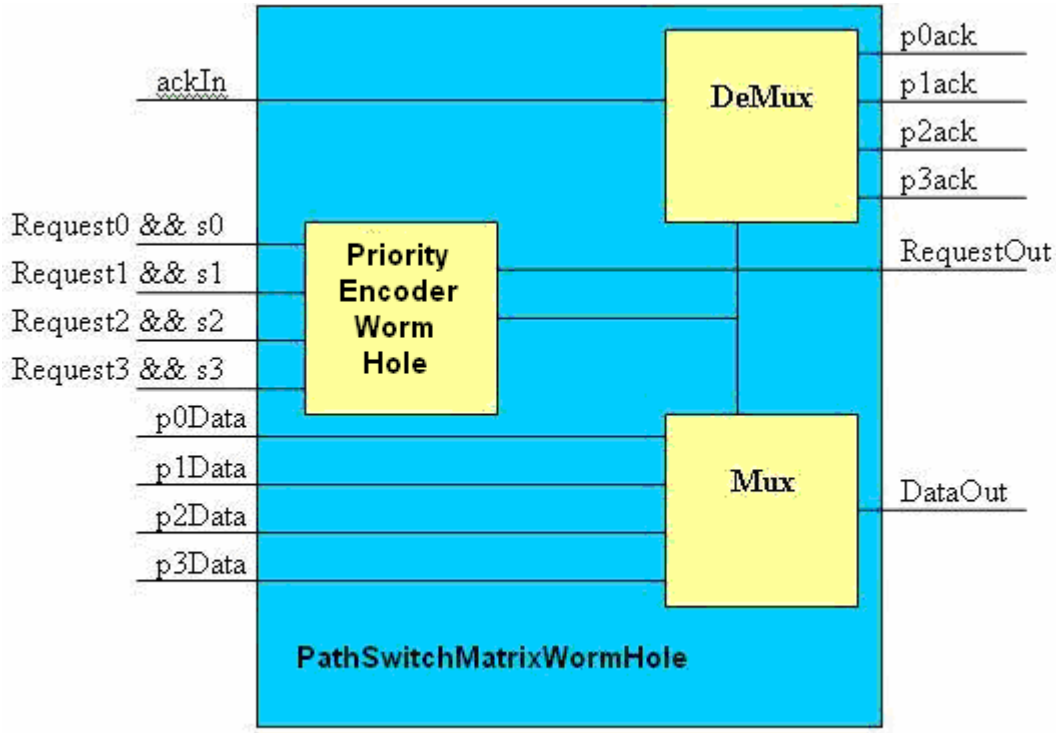



Figura 4.13: *PathSwitchMatrixWormHole*.

Aquesta és l'arquitectura del *router wormhole* i els components creats en aquest projecte.

# CAPÍTOL 5

## Resultats

L'anàlisi dels resultats de la creació de l'arquitectura d'un *router* amb estratègia de commutació *wormhole* requereix veure la síntesi i simulació dels mòduls descrits en Verilog.
Gràcies a les eines de treball esmentades en el capítol 3, podem arribar a validar el disseny resultant.

En el punt 5.2 d'aquest mateix capítol ens centrarem en les simulacions dels mòduls.

Ara doncs, comentarem els esquemàtics RTL que hem obtingut mitjançant l'eina de síntesi Synplify. Primer veurem l'esquema del mòdul receptor de dades MeshXYRoutingWormHole, explicat al capítol 4.

En la figura podem veure com s'han creat els mòduls FlitControlFSM encarregat de generar els senyals que estimulen el mòdul d'encaminament, XYRoutingWormHole.

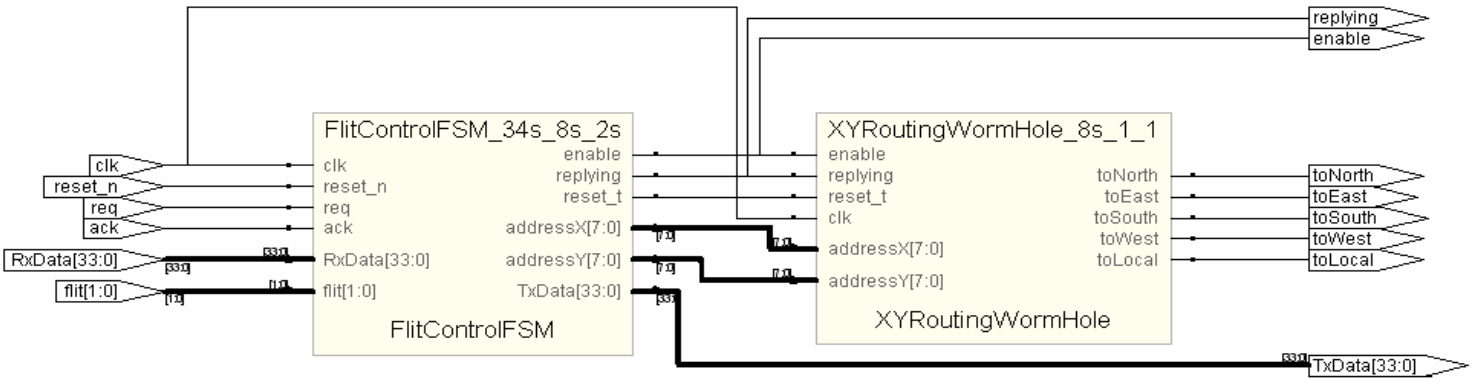



Figura 5.1: Esquema del mòdul MeshXYRoutingWormHole.

També cal comentar que els senyals *enable* i *replying* que surten del mòdul global són línies utilitzades com a línies de control durant la simulació i que verifiquen el funcionament del mòdul.

En les figures 5.2 i 5.3 veurem els següents esquemes, corresponents als mòduls MeshSwitchMatrixWormHole i PathSwitchMatrix, el qual engloba el primer.

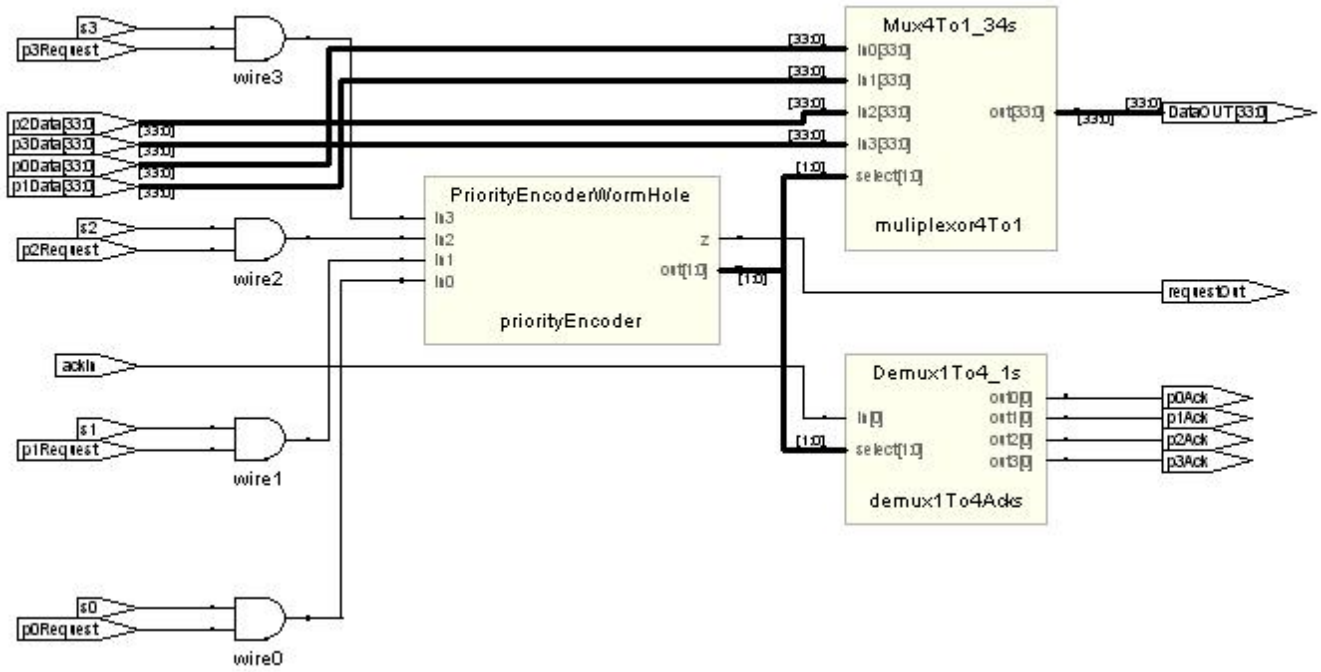



Figura 5.2: MeshSwitchMatrixWormHole

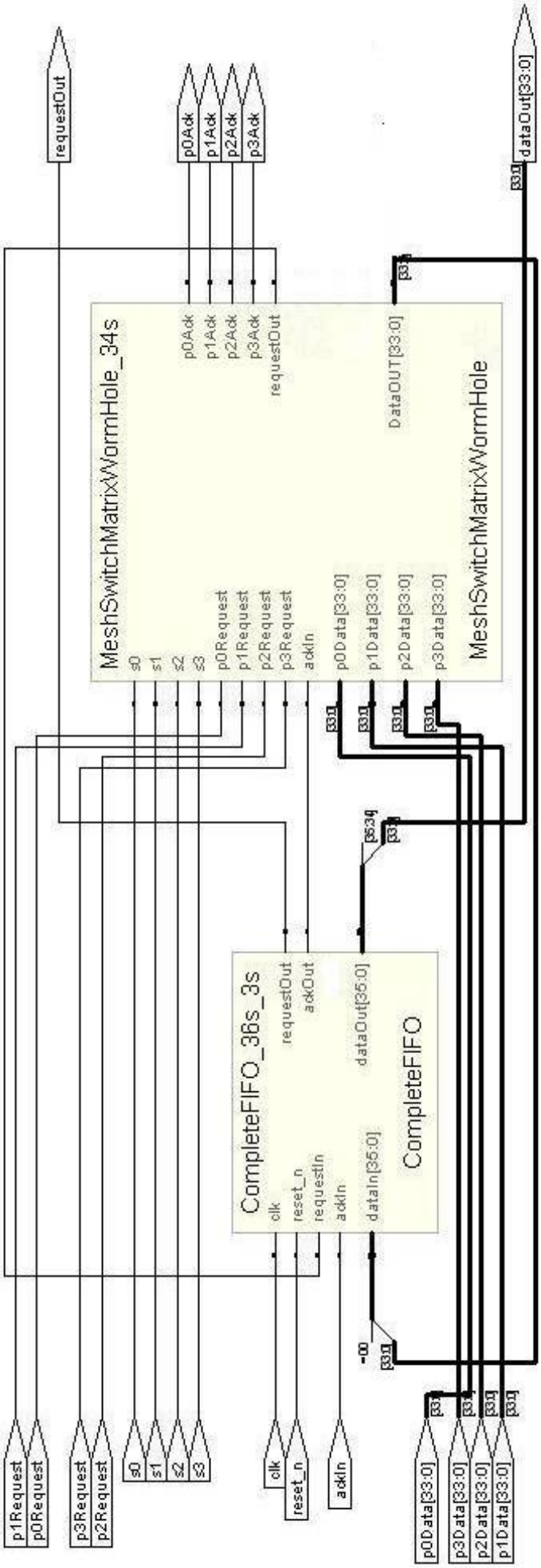



Figura 5.3: PathSwitchMatrix

En la figura 5.2 podem veure com s'ha integrat en el disseny el mòdul PriorityEncoder, punt clau en l'elecció de les prioritats quan hi ha col·lisions en el cuc, i els mòduls Demux1to4 i Mux4to1. Igualment, veiem com en el mòdul PathSwitchMatrix s'ha integrat una CompleteFIFO amb la seva RAM associada, inferida darrere del mòdul, tal com hem explicat, de manera que s'aconsegueix el *queue output*.

Finalment, en la figura 5.4 veiem el *top* d'un *path* o camí, en el qual apareixen els mòduls MeshXYRoutingWormHole i PathSwitchMatrix.

Un cop vist un camí, per tal de crear el *router* sencer, només s'ha de replicar cinc vegades el camí (un per a cada direcció).

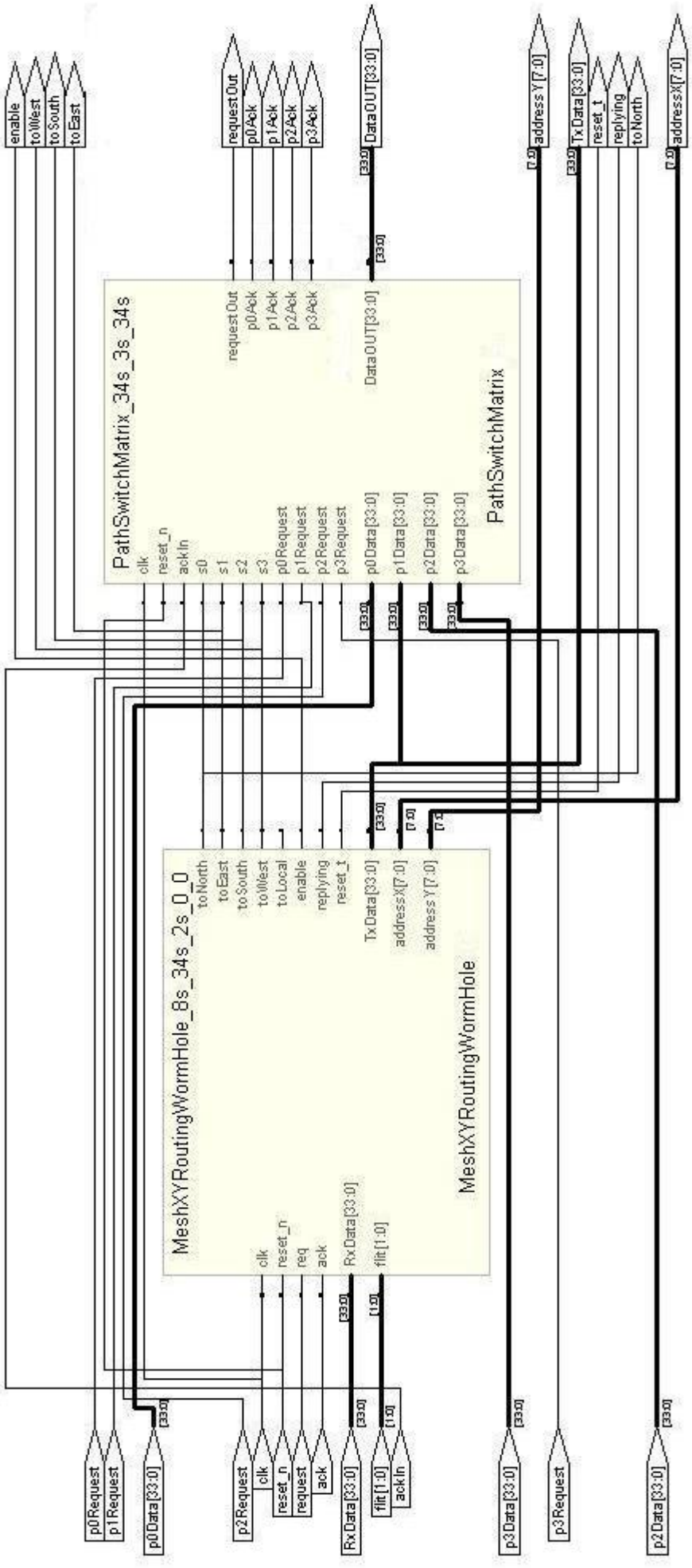



Figura 5.4: Esquema d'un camí o *path*

## 5.1 Costos Hardware

### 5.1.1 Àrea utilitzada per la NoC

Un cop vistos els esquemes RTL dels diferents components del *router*, un altre punt a tenir en compte en els resultats del projecte és el cost en hardware que té el *router wormhole*.

Per obtenir aquest cost també fem servir l'eina de síntesi, el Synplify, que ens dóna una sèrie d'arxius amb informació acurada del projecte. Un d'aquests arxius consisteix en els esquemàtics vistos anteriorment, però addicionalment també ens proporciona més informació complementària, com són, per exemple, el nombre de LUT's-LEs que fa servir la nostra solució, la freqüència mínima requerida o l'estimada, el temps de *setup* o el nombre de nivells de lògica que incorpora el disseny.

Però abans de poder veure els resultats cal saber sobre què ha estat sintetitzat el *router* per poder entendre'ls.

Synplify permet sintetitzar per a diverses plataformes basades en hardware reconfigurable - FPGAs els dissenys dels diferents mòduls descrits usant Verilog. En concret, per al nostre cas hem escollit una FPGA d'Altera, ja que durant la carrera hem treballat molt satisfactòriament amb diferents FPGA d'aquesta companyia i Synplify ens ofereix aquesta possibilitat.

La *Field Programable Gate Array* (FPGA) escollida és una de la família STRATIX, en concret el model EP1S25 que disposa d'un total de 25.660 elements lògics programables, que a priori representen una potència molt superior a la que nosaltres necessitem per a la creació de l'arquitectura *router* amb *wormhole*.

La STATRIX EP1S25 que hem fet servir en la síntesi disposa de diferents elements de memòria que es poden fer servir per sintetitzar la cua.

Aquests elements són:

- 2 blocs de  512K
- 138 blocs de 4kB.
- 224 blocs de 512 B

Els blocs M512 són memòries Simple dual-port RAM amb 512 bits més la paritat (576 bits), Els blocs M4K són memòries True dual-port RAM amb 4K bits més la bits de paritat (4.608 bits), i Per finalitzar, els blocs M-RAM són memòries True dual-port amb 512K bits més la paritat (589.824 bits).

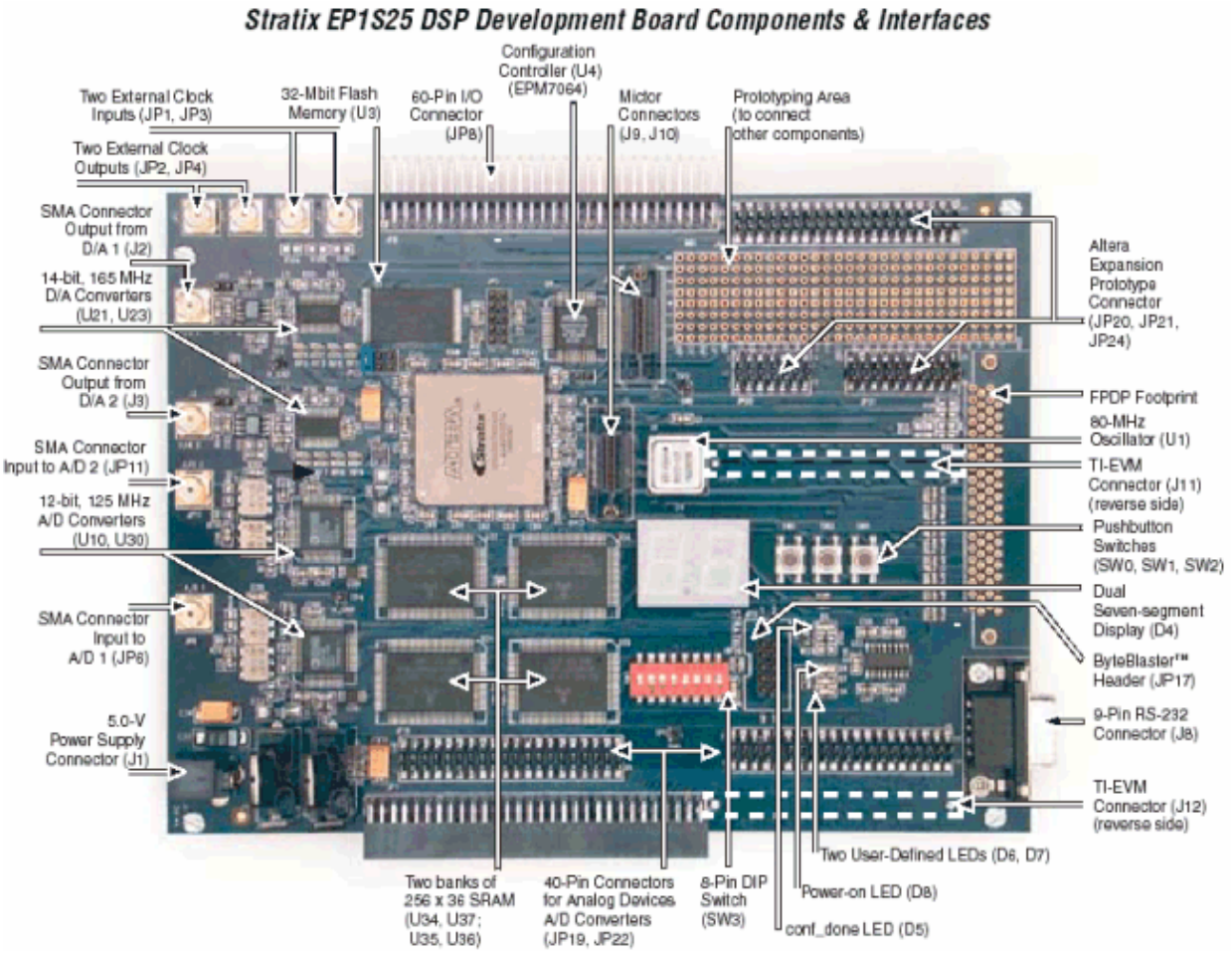



Figura 5.5: Imatge FPGA Altera STRATIX

Les característiques d'aquesta FPGA són:

Altera® FPGA Stratix EP1S25F780C5
- Package type 780-pin FineLine BGA
- C5 speed grade (-5)
- Voltage1.5V internal, 3.3V I/O
- 597 maximum user I/O pins
- Contains 25.660 LEs
- Includes 1.944.576 bits of on-chip memory
  - 224 M512 RAM blocks (32x18 bits)
  - 138 M4K RAM blocks (128x36 bits)
  - 2 M-RAM blocks
- 10 DSP blocks & 80 embedded multipliers (based on 9x9)
- 6 PLLs

Un cop vist tot això, podem passar a veure quin és el cost hardware que se sintetitza inferit de la nostra solució.

El tret més característic de la tècnica *wormhole* és la inclusió de les cues que fan millorar el rendiment de la commutació, però implica un cert cost.
Tenim dues possibilitats per sintetitzar la memòria que fa servir la cua. La primera es fent servir registres interns que, a priori, ens proporcionen més prestacions, i la segona és fent servir els blocs de memòria interna que té la pròpia FPGA, que tot i que tenen pitjors prestacions que els registres en qüestió de velocitat, generen menys costos. Arribats a aquest punt ens trobem davant d'un problema d'equilibri, que és més important, més capacitat amb pitjor rendiment, o menys capacitat amb millor rendiment?
La resposta a aquesta pregunta no és fàcil, però tot depèn de diversos factors, el preu, els requeriments que es necessiten, l'espai disponible dintre del xip, entre d'altres.

En la figura 5.6 podrem veure una comparativa entre les cues esmentades al capítol 2 , sintetitzades amb registres i memòria interna:

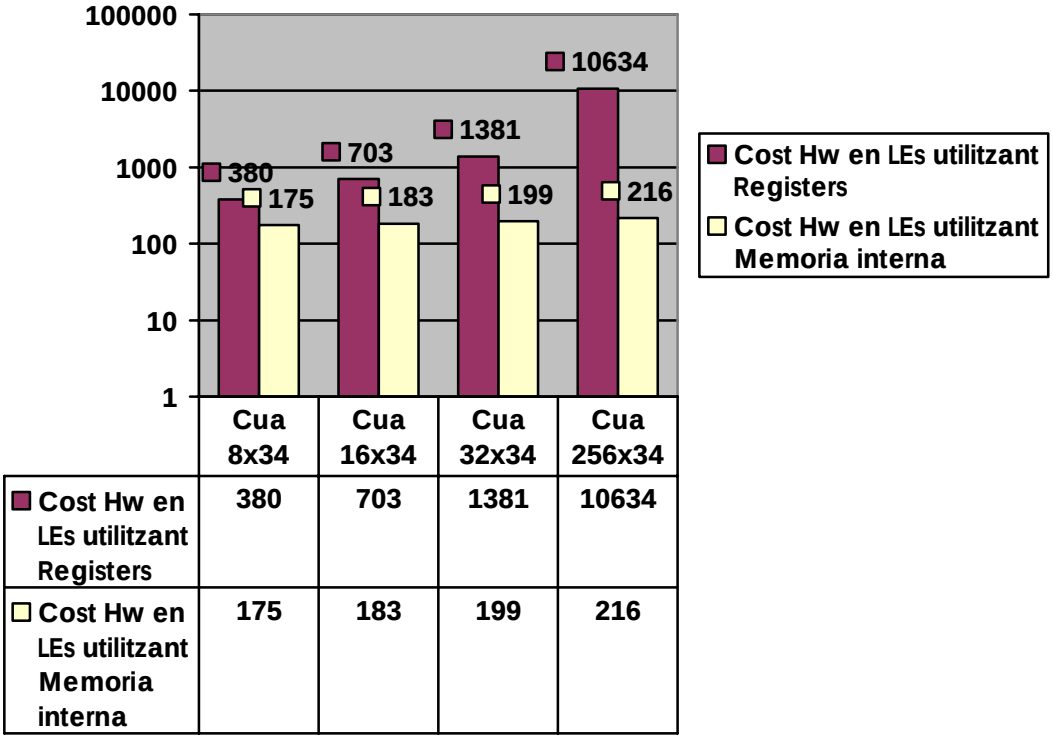



Figura 5.6: Cost de diferents cues.

Com es pot veure en la figura 5.6 (d'escala logarítmica) hi ha molta diferencia en la quantitat d'elements lògics utilitzats per sintetitzar les dues cues.

Per obtenir els resultats que venen a continuació, s'ha utilitzat la cua 8x34 amb memòria interna, ja que és la que genera menys cost hardware.

Així doncs, segons els resultats obtinguts per Synplify, un *path* o camí del *router* fa servir 289 elements lògics incloent-hi la memòria inferida per la cua/FIFO.

En canvi, el *router* sencer necessita uns 867 elements lògics. Aquest resultat és degut al fet que en inferir-se l'eina de síntesi ha eliminat tots els elements lògics que no podria fer servir, ja que el *router* escollit és el que ocuparia la posició (0,0) dintre de la matriu formada per la *Mesh* de *routers*. Per tant, no té els camins per anar a l'oest ni al nord. Això és degut al fet que s'ha experimentat amb un router de 3 ports (2 + local) ja que la Mesh és de 2x2.
Aquests resultats s'obtenen inferint la memòria RAM com a memòria interna, per tant no ocupa cap element lògic.

Amb aquests resultats veiem que el *router* necessita sobre el 4,5% de la FPGA d'Altera escollida.

Pel que fa a la freqüència de funcionament l'eina proporciona informació sobre el rellotge. Pel cas concret del *router wormhole* dissenyat, synplify calcula un rellotge de freqüència estimada de 231.0 Mhz amb un període de 4.329 ns. Això es degut a que en analitzar els camins interiors del router, synplify ha trobat que el pitjor camí triga 4,3 ns.
El temps total de retard (temps de propagació més temps de setup) dels 4.3 ns es divideix en:

- 3.9 ns logics (91,9%)
- 0.4 ns route (8,1%).

## 5.2 Simulacions

Per tal de comprovar que el funcionament del *router* i els seus mòduls és el satisfactori, necessitem crear bancs de proves.
Seguint la metodologia explicada al punt 3, cada component del *router* té un conjunt específic de proves que en garanteix el funcionament.

Per fer la simulació, hem fet servir un altre cop el ModelSim. En la figura 5.7 podem veure una simulació simple del mòdul de control dels flits i com es genera l'encaminament adequat depenent de la informació del paquet.

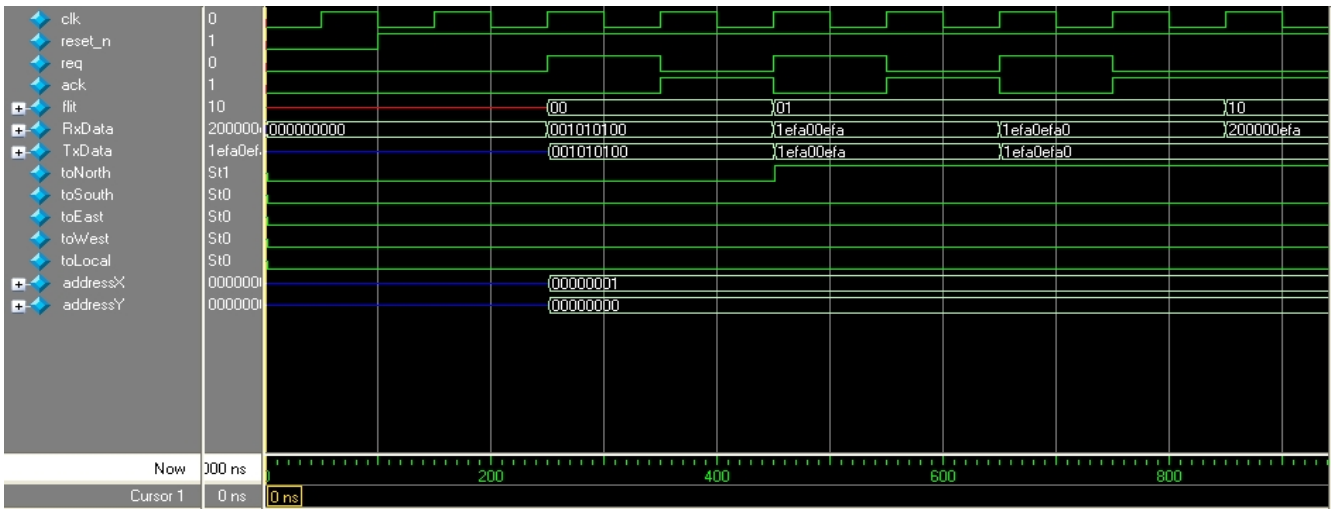



Figura 5.7: Simulació de formes d'ona obtinguda amb ModelSim.

En la imatge veiem com després del reset global que afectaria tot el *router*, entra el primer grup de dades, injectat al router pel NiC que té associat. Gràcies al flit 00, el mòdul de control de flits divideix el paquet i obté les adreces destinació i origen que estimularan els mòduls comparadors. Aquests generaran els senyals que necessita el mòdul encarregat de l'algorisme d'encaminament, que activa la sortida requerida, en aquest cas toNorth.

Seguint el *handshake full duplex*, dos cicles més tard entra el següent flit de dades amb el control 01, cosa que indica que només conté informació que ha de seguir l'encaminament del *head* del cuc.

Finalment, amb el flit 10 de l'últim paquet de dades s'acaba l'enviament d'informació.

Pel que fa al *router* sencer, cal afegir a aquesta simulació que abans de sortir del *router* tots els paquets són guardats dintre de la FIFO de sortida.
Com ja hem dit anteriorment, si la FIFO és plena, tot el procés d'encaminament queda aturat.

## 5.3 Comparació amb altres NoCs

Per acabar amb el capítol de resultats, crec convenient fer una petita comparació entre l'arquitectura *router wormhole* creada al projecte i una altra de ja existent.

El *router* escollit té una tècnica d'encaminament XY determinista amb *circuit switching*. Aquest *router* per comparar ha estat proporcionat per l'enginyer Jaume Joven i Murillo.

Un cop sintetitzat el segon *router* amb la mateixa eina i contra la mateixa FPGA, el nombre d'elements lògics requerits ha estat de 370 LEs, que representa un 1,5% del total de la FPGA, enfront del 4,5% obtingut en sintetitzar el *router packet switching wormhole*.

Tot i que el cost hardware és el triple en el cas del *wormhole*, per tots els motius exposats en els capítols anteriors la solució *packet switching* presenta molts avantatges respecte al *circuit switching*, ja que al disposar de cues es millora el tràfic d'informació dintre de la NoC i, a més a més, la comunicació es produeix entre nodes veïns i no d'origen a destinació final.

# CAPÍTOL 6

## Conclusions i línies de futur

## 6.1 Conclusions

Les conclusions a què podem arribar havent realitzat aquest projecte són que en el món de la microtecnologia (cada vegada més a prop de la nanotecnologia) les millores i la evolució en les metodologies i en els dissenys són constants, i una d'aquestes millores és la presentada en aquest projecte.

Com hem pogut veure quan hem estudiat les diferents tècniques de commutació de paquets, el *wormhole* presenta grans avantatges respecte a les tècniques de *circuit switching*, i el seu cost només representa un increment del 3% de l'àrea disponible.

Amb aquest 3% la comunicació dintre de la *Mesh* és òptima i l'amplada de banda que es pot aconseguir és màxima i constant (ja que, com que la distància entre l'emissor i el receptor és constant, no en depèn).

La inclusió dels mòduls encarregats de controlar els flits que conformen els paquets dins de la NoC, i les cues són els principals culpables de l'increment de l'àrea a fer servir. Concretament la memòria necessària per implementar les cues representa el gran "*trade off*" al qual s'ha d'arribar per obtenir l'equilibri que es busca entre rendiment i cost.

Una major capacitat de memòria dins de la cua ens pot oferir grans beneficis, ja que trigarà més a omplir-se facilitant la comunicació dintre de la Mesh.

Cal dir, però, que la tècnica *wormhole* amb la cua unificada no és la solució final, perquè, encara que disposi d'una cua de sortida, aquesta pot quedar plena i el *router* i el seu node quedarien aturats, de la mateixa manera que passa en el *circuit switching*.

En el punt 5.2, línies de futur, donarem una visió de les possibles millores i evolucions d'aquest projecte.

L'elecció de la forma de sintetitzar amb Verilog la memòria de les cues també es presenta com una tria important. Depenent de si s'utilitza registres o blocs de memòria el cost a suportar pot ser inaccessible, així per exemple si es vol utilitzar una cua amb profunditat de 256 posicions amb amplada de paraula de 34 bits (amplada utilitzada en tot el projecte) i 42,5 kb de capacitat, sintetitzada amb registres interns, necessitarem més de 10.000 elements lògics, que multiplicats pel nombre de paths del router (tres en el cas analitzat en el capítol de resultats), donaria que es necessiten més de 30.000 elements lògics, cosa inviable ja que la FPGA que hem fet servir al projecte només en té 25.000 elements lògics, per exemple.

A més, l'increment de memòria comporta un increment en el consum, i alhora un increment de consum comporta haver de dissipar més calor en el xip, i com hem pogut veure en el capítol 2, quan es parlava de la tècnica *wormhole*, un increment de la memòria no comporta necessariament una millora equivalent en el rendiment del conjunt.

Així doncs, tant la profunditat de les cues com la manera de sintetitzar la memòria són altres importants compromisos a resoldre que dependran de les necessitats del projecte.

Tot i això, gràcies a la parametrització que s'ha fet servir en l'especificació del projecte i a les característiques esmentades al capítol 3 sobre la reutilització de codi, es poden satisfer diferents necessitats amb el mateix codi sintetitzable sense haver de canviar grans blocs de codi.

Així doncs, per exemple, canviant dos paràmetres s'ha pogut sintetitzar amb el mateix codi diferents versions de cues per tal de poder fer les comparacions vistes al capítol de resultats.

Durant el desenvolupament del projecte s'han fet servir gran part dels coneixements adquirits durant la carrera en diferents àmbits, no sols en les assignatures de hardware, sinó també en les de software. I per aconseguir especificar i validar el codi generat, primer ha calgut passar per una fase

d'estudi del problema en qüestió i del marc tecnològic en el qual està enquadrat aquest projecte.

Fent servir les metodologies i la disciplina apresa en la carrera, s'ha anat avançant en les diferents etapes del projecte, no lliures de problemes, com era d'esperar, i s'han anat complint els objectius proposats al principi.

Altres conclusions que puc extreure del projecte són de caire personal. El treball realitzat ha estat molt satisfactori, ja que he tingut la possibilitat de completar i posar a prova els meus coneixements sobre la matèria en el camp de la microelectrònica, que, d'altra banda, és la branca de la carrera que personalment més m'ha satisfet.

Gràcies a aquest projecte, he pogut aprofundir en la utilització d'eines de descripció i simulació, com és el ModelSim, i hi he adquirit així més experiència i desimboltura. També he pogut conèixer noves eines de síntesi molt potents i intuïtives de fer servir, el Synplify.

També he pogut ampliar els meus coneixements amb el llenguatge HDL Verilog, estudiat durant la carrera però poc utilitzat. Aquest llenguatge és el més comú en el món laboral, i gràcies al projecte em sento més competitiu de cara a la meva incorporació en el mercat del sector tecnològic hardware de treball.

Finalment, dir que estic satisfet i orgullós d'haver pogut realitzar el projecte per al departament de microelectrònica de la UAB i, en concret, d'haver treballat en un projecte per al Dr. Jordi Carrabina i Bordoll sota la direcció d'en Jaume Joven i Murillo.

## 6.2 Línies de futur

Com ja s'ha anat comentant durant la memòria, aquest projecte no està del tot tancat.

La possibilitat de millorar altres aspectes de l'arquitectura donen un camp molt ampli per a l'evolució del *router*.

Una primera millora que proposo seria aprofundir en la millora de l'algorisme d'encaminament. En aquest projecte s'ha donat una visió molt genèrica de les possibilitats existents en aquest aspecte, i tot i reconèixer que podrien existir altres algorismes més eficients, s'ha optat per escollir un mètode XY simple, ja que entre els objectius no hi havia la millora de l'encaminament.

Però existeixen altres possibilitats: els algorismes adaptatius representarien una millora clara respecte al determinista que hem fet servir. En un altre projecte es podrien estudiar les diferents qualitats que ofereixen les alternatives dels algorismes adaptatius, com podrien ser, com hem comentat breument en el capítol 4, Implementació:

- West First
- North Last
- Negative First
- Odd even

Una altra possible millora seria el fet de presentar una cua pròpia de sortida per a cada camí existent en el *router*.
En l'arquitectura presentada, existeix una única cua de sortida per a tots els camins, i el que aquí es proposa seria fer un estudi comparatiu entre el rendiment que podria donar si s'afegissin més cues de sortida i l'increment dels recursos i de l'àrea que es necessitaria per a les cues afegides.

Com a conseqüència, En afegir més d'una cua de sortida, necessitaríem trobar una política de cues per governar-les.

Un primer intent de realització podria implementar un Round Robin simple a la sortida, i a partir d'aquí continuar l'estudi amb altres polítiques tot comparant els recursos/àrea requerida i la millora que representaria.

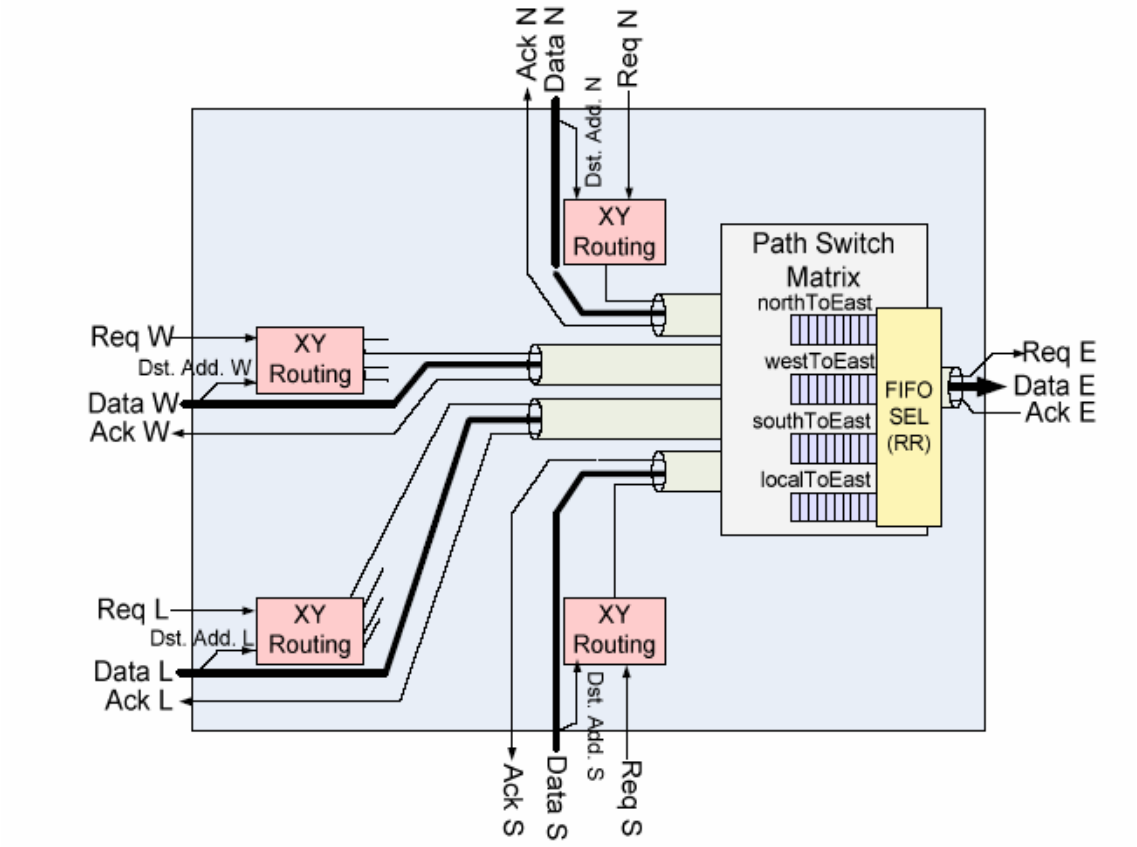



Figura 6.1: Router Wormhole amb diverses cues de sortida i política RoundRobin.

Amb aquestes millores s'aconseguiria un gran rendiment del *router* i de la comunicació dintre de la NoC, combinant la possibilitat d'encaminar entre nodes veïns, tal com permet el *packet switching* implementat, i alhora poder adaptar l'encaminament depenent de l'estat dels veïns.

# Bibliografia

**Referències de llibres.**


-[Benini02]
Benini, L. De Micheli, G. "Networks on chip: a new paradigm for systems on chip design", This paper appears in: Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings

-[Castells06]
David Castells-Rufas, Jaume Joven, Jordi Carrabina, "A validation and performance evaluation tool for ProtoNoC", International Symposium on System-on-Chip'06, 2006

-[Coenen06]
Martijn Coenen, Srinivasan Murali, Andrei Radulescu1, Kees Goossens, Giovanni De Micheli, "A buffer-sizing Algorithm for Networks on Chip using TDMA and credit-based end-to-end Flow Control", CODESS+ISSS, 2006

-[Guerrier00]
P. Guerrier and A. Greiner, "A generic architecture for on-chip packetswitched
interconnections." Proc. DATE 2000, March 2000,

-[Hu04b]
Jingcao Hu, Marculescu, R., "Application-specific buffer space allocation for networks-on-chip router design", Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on, vol., no.pp. 354-
361, 7-11 Nov. 2004

-[Jantsch03]
Axel Jantsch i Hannu Tenhunen. "*Networks on chip"*, Kluwer Academic Publishers (2003)

-[Kumar02]
Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., Tiensyrja, K., Hemani, A., "A network on chip architecture and design methodology", VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on , vol., no.pp.105-112, 2002

-[LluísTeres98]
Ó E. Lecha, M. Moré, F. Rincón, L. Terés y J. Vidal, "el lenguaje vhdl", cnm-uab, Seminario de VHDL, 1998-00

-[LluísTeres06]
L.Terés. “Visión global preliminar de la microelectronica”.cnm-imb (CSIC) 2006

-[MELLOt]
Aline Mello, Luciano C. Ost, Fernando G. Moraes, Ney L. Calazans. “Evaluation of Routing Algorithms on Mesh Based NoCs”, Internal Technical Report temporally available in http://www.inf.pucrs.br/tr/tr040.pdf, 2004.

-[Ogras05] U. Y. Ogras, J. Hu, R. Marculescu, "Key Research Problems in NoC Design: A Holistic Perspective", in Proc. of the International Conf. on HWSW Codesign and System Synthesis, Sep. 2005

-[Rijpkema03]
E. Rijpkema, K.G.W. Goossens, A. Rˇadulescu,
J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander
“Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort
Services for Networks on Chip”, Philips Research Laboratories, Eindhoven, The Netherlands

-[Saastamoinen03]
Saastamoinen, I., Alho, M., Nurmi, J., "Buffer
implementation for Proteo network-on-chip", Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on, vol.2, no.pp.
II-113- II-116 vol.2, 25-28 May 2003

-[SPLevitan01]
Steven P. Levitan “Introduction to System on a Chip (**SoC**) Design” Computer Engineering and Electrical Engineering, University of Pittsburgh 2001

**Referències electròniques.**

http://www.altera.com/

http://www.model.com/

http://www.synplicity.com/

- Especificacions dels seus productes.

http://www.OpenCore.org/

- Components Verilog.

## RESUM

Aquest projecte presenta la implementació d'un disseny, i la seva posterior síntesi en una FPGA, d'una arquitectura de tipus *wormhole packet switching* per a una infraestructura de *NetWork-On-Chip* amb una topologia 2D-Mesh. Agafant un *router circuit switching* com a punt de partida, s'han especificat els mòduls en Verilog per tal d'obtenir l'arquitectura *wormhole* desitjada. Dissenyar la màquina de control per governar els flits que conformen els paquets dins la NoC, i afegir les cues a la sortida del *router* (*outuput queuing*) són els punts principals d'aquest treball. A més, com a punt final s'han comparat ambdues arquitectures de *router* en termes de costos en àrea i en memòria i se n'han obtingut diverses conclusions i resultats experimentals.

## RESUMEN

Este proyecto presenta la implementación de un diseño, y su posterior síntesis en una FPGA, de una arquitectura de tipo *wormhole packet switching* para una infraestructura de *Network-On-Chip* con una topología 2D-Mesh. Tomando un *router circuit switching* como punto de partida, se han especificado los módulos en Verilog para obtener la arquitectura deseada. Diseñar la máquina de control que gobierna los flits que conforman los paquetes dentro de la NoC, y añadir las colas de salida del *router* (*output queuing*) son los puntos principales de este trabajo. Además, como punto final se han comparado ambas arquitecturas del *router* en términos de costos en área y en memoria obteniendo varias conclusiones y resultados experimentales.

## SUMMARY

This project presents the implementation and synthesis in a FPGA of a wormhole packet switching architecture design for a NetWork-On-Chip structure with 2D-Mesh topology. Taking a circuit switching router as a starting point, the modules have been specified in Verilog in order to achieve the wormhole architecture. The main points of this work are to design the control machine that controls the flits conforming the packets within the NoC, and to include the output queue of the router. Finally, both router architectures have been compared in area and memory costs and some conclusions and experimental outcomes have been drawn.