



**Escola Politècnica Superior
d'Enginyeria de Vilanova i la Geltrú**

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE MÀSTER

TÍTOL: Anàlisi de la interconnexió de dispositius lògics programables mitjançant Ethernet

AUTOR: Peshevski, Marko

DATA DE PRESENTACIÓ: Febrer, 2017

COGNOMS: Peshevski

NOM: Marko

TITULACIÓ: Màster Universitari en Enginyeria de Sistemes Automàtics i Electrònica Industrial (MUESAEI)

PLA:

DIRECTOR: Mariano López García

DEPARTAMENT: EEL - Departament d'Enginyeria Electrònica

QUALIFICACIÓ DEL TFM

TRIBUNAL

PRESIDENT

SECRETARI

VOCAL

DATA DE LECTURA:

Aquest Projecte té en compte aspectes mediambientals: ☐ Sí ☐ No

RESUM

Paraules clau (màxim 10):

ABSTRACT

Keywords (10 maximum):

SUMARI

1. INTRODUCCIÓ.....	8
1.1. Objectius	8
2. FPGA.....	9
2.1. Estructura.....	9
2.2. Nuclis de propietat intel·lectual	11
2.3. MicroBlaze.....	12
Generació de Hardware.....	12
Programació de Software	14
3. ETHERNET	15
3.1. Pila de protocol per capes	15
3.2. Funcionament d'Ethernet a la placa utilitzada	17
3.3. Protocols utilitzats	18
4. IMPLEMENTACIÓ PRÀCTICA.....	23
4.1. LwIP	28
4.2. Pila programada per l'autor	34
5. RESULTATS EXPERIMENTALS	41
5.1. Mètode d'estudi	41
5.2. Resultats de les proves.....	43
5.3. Comparació entre les piles	0
6. CONCLUSIONS I TREBALL FUTUR	0
6.1. Conclusió	0
6.2. Treball futur.....	0
7. BIBLIOGRAFIA I ANNEXES	0

SUMARI DE FIGURES

FIGURA 1. ESTRUCTURA DE LA FPGA, ALTAMENT SIMPLIFICADA	9
FIGURA 2. CEL·LA LÒGICA INDIVIDUAL SIMPLIFICADA D'UNA FPGA	9
FIGURA 3. REPRESENTACIÓ SIMPLIFICADA D'UN DELS INTERRUPTORS D'INTERCONNEXIÓ D'UNA FPGA	10
FIGURA 4. REPRESENTACIÓ SIMPLIFICADA D'UN DELS BLOCS D'ENTRADA/SORTIDA DE LA FPGA	10
FIGURA 5. FOTO DE LA PLACA UTILITZADA EN AQUEST TREBALL	11
FIGURA 6. EXTRACTE D'UNA LLISTA DE IP CORES DISPONIBLES PER L'USUARI	12
FIGURA 7. VISTA DE CONFIGURACIÓ DELS MÒDULS IP CORE TRIATS PEL PROJECTE EN CURS	13
FIGURA 8. ASSIGNACIÓ D'ADRECES DE L'ESPAI DE MEMÒRIA DE MICROBLAZE	13
FIGURA 9. CAPTURA DE PANTALLA DEL PROGRAMARI SOFTWARE DEVELOPMENT KIT	14
FIGURA 10. REPRESENTACIÓ DEL MODEL OSI	15
FIGURA 11. DIFERENTS CATEGORIES DE CABLEJAT PER ETHERNET QUE ES PODEN TROBAR	16
FIGURA 12. REPRESENTACIÓ DE VÀRIES TOPOLOGIES DE XARXES POSSIBLES	16
FIGURA 13. DIAGRAMA QUE REPRESENTA EL FUNCIONAMENT D'ETHERNET A LA PLACA UTILITZADA	17
FIGURA 14. DIAGRAMA QUE REPRESENTA LA ESTRUCTURA D'UNA TRAMA ETHERNET	19
FIGURA 15. DIAGRAMA QUE REPRESENTA LA ESTRUCTURA D'UN PAQUET IPV4	20
FIGURA 16. DIAGRAMA QUE REPRESENTA LA ESTRUCTURA D'UN PAQUET ARP	21
FIGURA 17. DIAGRAMA QUE REPRESENTA LA ESTRUCTURA D'UN PAQUET ICMP	22
FIGURA 18. CREACIÓ D'UN PROJECTE NOU AL PROGRAMARI XPS	23
FIGURA 19. SELECCIÓ DE LA FPGA UTILITZADA I EL SISTEMA QUE S'HI VOL IMPLEMENTAR	24
FIGURA 20. SELECCIÓ DELS MÒDULS PREDEFINITS PER LA PLACA EN QÜESTIÓ QUE S'ESTÀ UTILITZANT	24
FIGURA 21. VISTA DESPRÉS D'EXPORTAR EL DISSENY DEL XPS	25
FIGURA 22. CREACIÓ D'UN NOU BOARD SUPPORT PACKAGE	25
FIGURA 23. SELECCIÓ DE LA CAPA DE SOFTWARE D'UN BSP	26
FIGURA 24. CONFIGURACIÓ DELS ELEMENTS QUE INTEGREN UN BSP	26
FIGURA 25. CREACIÓ D'UN NOU PROJECTE D'APLICACIÓ	27
FIGURA 26. SELECCIÓ DE BSP QUE ES VOL FER SERVIR PEL PROJECTE D'APLICACIÓ	27
FIGURA 27. PROVES D'ECO AMB LES DIFERENTS IMPLEMENTACIONS SOBRE UN CABLE CAT3	43
FIGURA 28. PROVES D'ECO AMB LES DIFERENTS IMPLEMENTACIONS SOBRE UN CABLE CAT6	44
FIGURA 29. PROVES D'ECO AMB LES DIFERENTS SUMES DE VERIFICACIÓ	46

1. INTRODUCCIÓ

En aquest treball s'estudien els dispositius coneguts com FPGA, de l'anglès Field Programmable Gate Array. Aquests dispositius han crescut en popularitat al llarg de les últimes dècades degut a què cada cop s'han fet més accessibles i han incorporat major nombre d'elements lògics. A grans trets, són dispositius que permeten reprogramar les connexions entre els *blocs lògics* a l'interior dels mateixos, per aconseguir des de funcions lògiques senzilles, fins a aplicacions relativament complexes que necessitin d'una elevada densitat a nivell d'electrònica. El principal avantatge que ofereixen aquests dispositius respecte a d'altres dispositius similars i d'altres mètodes per implementar funcions lògiques és la seva gran integració (solen ser circuits integrats molt densos, amb un nombre d'elements lògics des de desenes de milers fins a milions), i la seva reprogramabilitat, a diferència dels circuits integrats i sistemes sobre xip dedicats.

Per altra banda, en aquest mateix treball s'analitza i estudia la connectivitat Ethernet. Aquesta connectivitat ja té una llarga vida, existeix des de la dècada dels 1970. És àmpliament coneguda i àmpliament utilitzada, tant a nivells industrials com a nivells d'electrònica de consum. És, per tant, de gran interès conèixer com funciona, i quines són les seves possibilitats i limitacions. Per sobre d'Ethernet, que només correspon a les capes física i de control d'accés al medi, generalment s'hi poden trobar altes protocols, com ara: ARP, TCP/IP, Token Ring, Token Bus, etcètera. Aquests protocols són els que realment treballen amb les dades que es volen intercanviar entre els dos (o més) punts a la xarxa que estiguin comunicats.

En la present obra s'estudia i analitza com es poden unir tots dos mons, i quins són els avantatges i inconvenients. Essent que TCP/IP és una pila de protocols de comunicació global i molt utilitzada, existeixen moltes implementacions de la mateixa sobre moltes arquitectures diferents. En aquest document s'empra una de les implementacions més conegudes, lwIP. Generalment aquesta implementació s'utilitza en sistemes incrustats, amb microprocessadors restringits en espai de memòria. Aquesta implementació es compara contra una implementació molt més bàsica, feta per l'autor, que permet intercanviar dades entre dos nodes en una xarxa ja existent, que respon només a algun dels protocols més bàsics d'una possible pila de protocols per establir una connexió entre dos equips connectats en una xarxa.

1.1. Objectius

Alguns dels objectius que s'han perseguit amb aquest treball són:

- Conèixer en profunditat el funcionament d'Ethernet a nivell de bit d'informació.
- Conèixer en profunditat alguns dels protocols més emprats a les comunicacions d'avui en dia (Internet).
- Dissenyar un sistema incrustat sobre una FPGA.
- Conèixer amb el màxim detall possible com funciona una implementació pràctica de la pila de protocols d'Internet per sistemes incrustats.
- Intentar desenvolupar una pila de protocol bàsica que permet respondre a peticions d'eco en una xarxa.
- Mesurar els temps que tarda una pila i l'altra en processar un paquet d'eco i comparar resultats.

2. FPGA

2.1. Estructura

Aquests dispositius, el nom dels quals traduït literalment és: formació de portes (lògiques) programables al camp (*in situ*); són un tipus de dispositius electrònics que permeten la generació de funcions lògiques, i aplicacions més complexes, mitjançant la reprogramació de l'estat dels seus blocs lògics i l'estat de les interconnexions entre aquests. Una imatge qualitativa de l'estructura interna d'una FPGA podria ser la de la Figura 1.

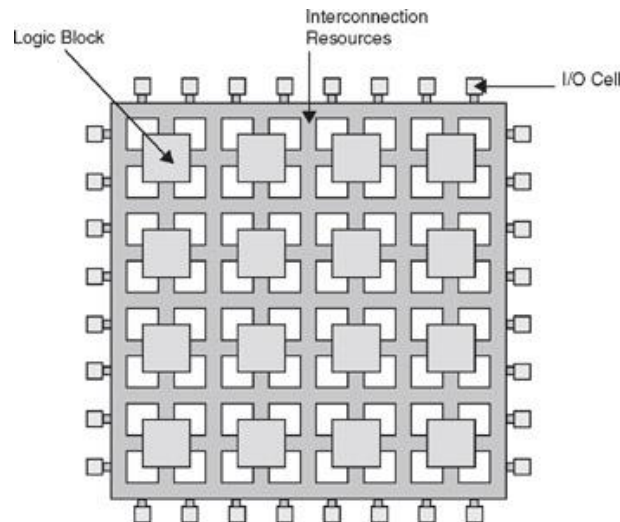


Figura 1. Estructura de la FPGA, altament simplificada

Els blocs lògics, coneguts també com cel·les lògiques són els elements capaços de realitzar funcions lògiques. Un exemple d'una cel·la es pot trobar a la Figura 2.

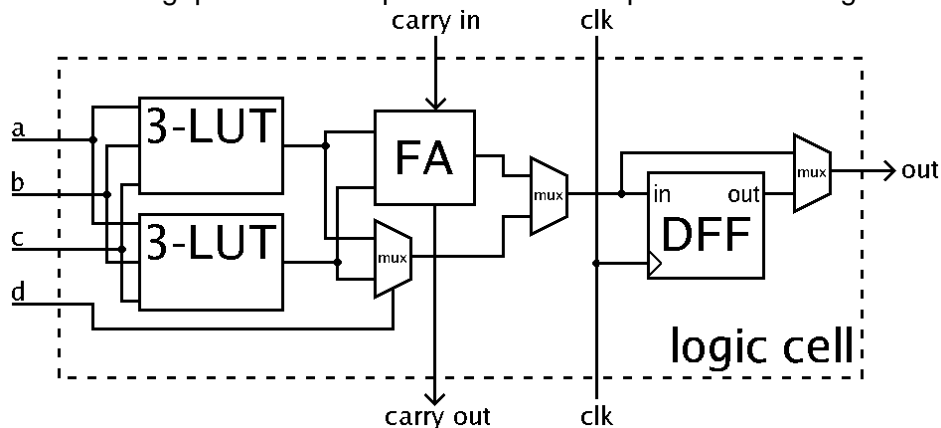


Figura 2. Cel·la lògica individual simplificada d'una FPGA

Segons es pot veure en aquesta Figura, cada cel·la d'una FPGA consisteix de 4 bits d'entrada (a, b, c i d), que permeten entrar informació a la cel·la a través de les taules d'entrada (LUT, Look Up Table), sigui del món exterior, com de la resta de cel·les. L'altra entrada (carry in), permetria encadenar cel·les per poder fer circuits sumadors més complexes utilitzant els sumadors complets (FA, Full-Adder de les cel·les), per exemple. En aquestes cel·les, el que se selecciona en el moment de la programació de la pròpia FPGA és l'estat dels multiplexors, per aconseguir que la cel·la es comporti d'una determinada manera. A la sortida hi ha un biestable de tipus D (DFF, D Flip-Flop), governat per un senyal de rellotge, que és, en general, global a totes o la majoria de cel·les. Aquest biestable és molt important degut a què els circuits que s'hagin de generar a les FPGA normalment han de ser síncrons. Cal notar que una

cel·la real sol ser més complexa, amb més entrades i probablement amb més elements lògics al seu interior.

Altres elements presents a les FPGA són els recursos d'interconnexió. Aquests permeten encaminar les connexions entre els blocs lògics i els blocs d'entrada/sortida. Aquests recursos d'interconnexió consisteixen d'interruptors programables que permeten seleccionar quins camins han de seguir les pistes d'interconnexió dels blocs lògics de la FPGA. Una representació altament simplificada es pot veure a la Figura 3.

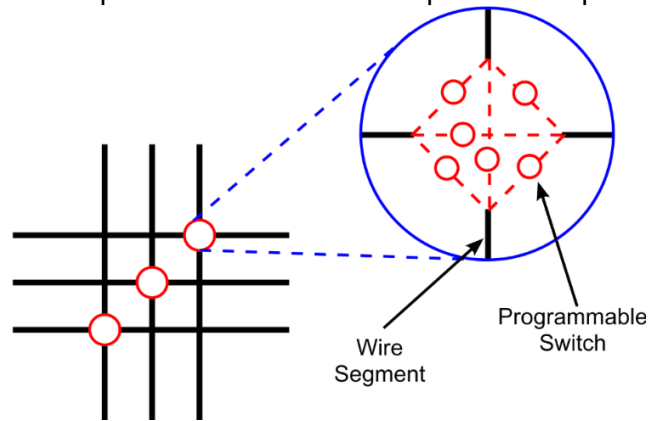


Figura 3. Representació simplificada d'un dels interruptors d'interconnexió d'una FPGA

Per últim, els elements que falta descriure d'una FPGA són els blocs d'entrada/sortida. Aquests blocs són trossos d'electrònica que permeten configurar les connexions de la lògica generada pels blocs lògics, i connectada pels recursos d'interconnexió, amb els pins que connecten la FPGA al món exterior. Generalment solen incorporar electrònica per poder fer que un pin en concret sigui entrada/sortida, o estigui en estat d'alta impedància (control tri-estat).

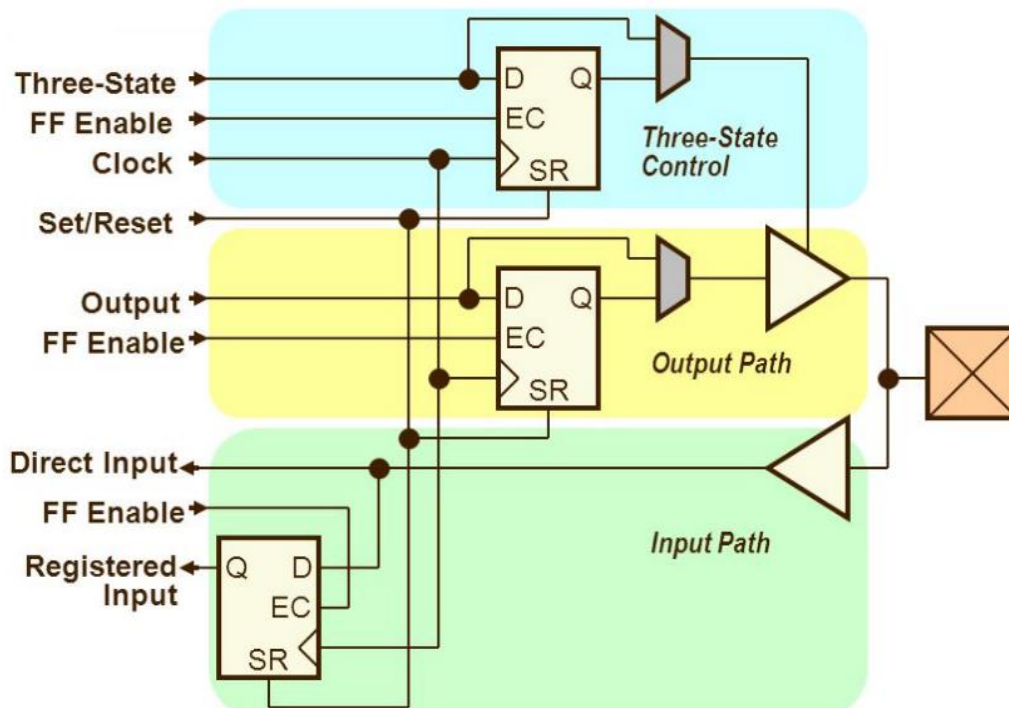


Figura 4. Representació simplificada d'un dels blocs d'entrada/sortida de la FPGA

2.2. Nuclis de propietat intel·lectual

Degut a l'àmplia disponibilitat de portes lògiques dintre d'una FPGA, aquestes es poden programar perquè es comportin com altres sistemes sencers. Per exemple, trossos de la lògica disponible a la FPGA es poden programar per comportar-se com un microcontrolador, amb uns perifèrics determinats, a triar per l'usuari segons necessitats de l'aplicació. La resta de la lògica disponible es podria fer servir per implementar, per exemple, funcions lògiques que necessiten ser executades molt ràpidament, com ara les d'una memòria d'accés aleatori disponible pel microcontrolador, o algun tipus de processat de senyal digital que s'executi en paral·lel amb el microcontrolador, de forma molt més ràpida.

Aquest tipus de programació aporta una flexibilitat molt gran, només limitada per la quantitat d'elements lògics disponibles. És per això que els grans fabricants de FPGA desenvolupen i permeten fer servir, de vegades sota llicència, els anomenats nuclis de propietat intel·lectual (de l'anglès Intellectual Property core). Els IP cores són implementacions en llenguatge de descripció de hardware (generalment VHDL o Verilog) d'algun dispositiu dins la lògica de la FPGA. Poden ser de diferents tipus: un controlador de memòria RAM DDR, un controlador d'accés al medi per Ethernet, un perifèric SPI, un microcontrolador sencer, etcètera. Quan aquests IP cores són implementats en llenguatge de descripció de hardware s'anomenen soft-cores. Més endavant, a la secció 4, es descriuran amb més detall els IP cores utilitzats en aquest treball.

Existeixen també versions permanents dels soft-cores, que són incrustats al silici de la pròpia FPGA, generalment en forma de microprocessador. Aquests últims s'anomenen hard-cores, i existeix una gran varietat dels mateixos. Generalment s'utilitzen de forma híbrida en conjunt amb la resta de la lògica de la FPGA, fent servir algun bus d'interconnexió entre totes dues parts. Els grans fabricants es decanten per un o un altre tipus. Per exemple, el fabricant Xilinx ofereix models de FPGA amb un microprocessador PowerPC incrustat, mentre que per altra banda Altera ofereix molts models amb un ARM incrustat.

Per aquest treball, degut a la disponibilitat al departament, s'utilitza una placa amb FPGA d'un dels principals fabricants del mercat, Xilinx. La placa en qüestió és la Avnet Spartan-6 LX9 MicroBoard. Aquesta placa duu una FPGA XC6SLX9 de Xilinx. Aquesta FPGA és un dels models bàsics del fabricant Xilinx, i l'usuari disposa entre d'altres, de 9152 blocs lògics amb taules d'ent i un màxim de 200 entrades/sortides. Aquesta placa duu incorporada connectivitat Ethernet fins a 100 Mb/s, que s'utilitzarà en aquest treball.



Figura 5. Foto de la placa utilitzada en aquest treball

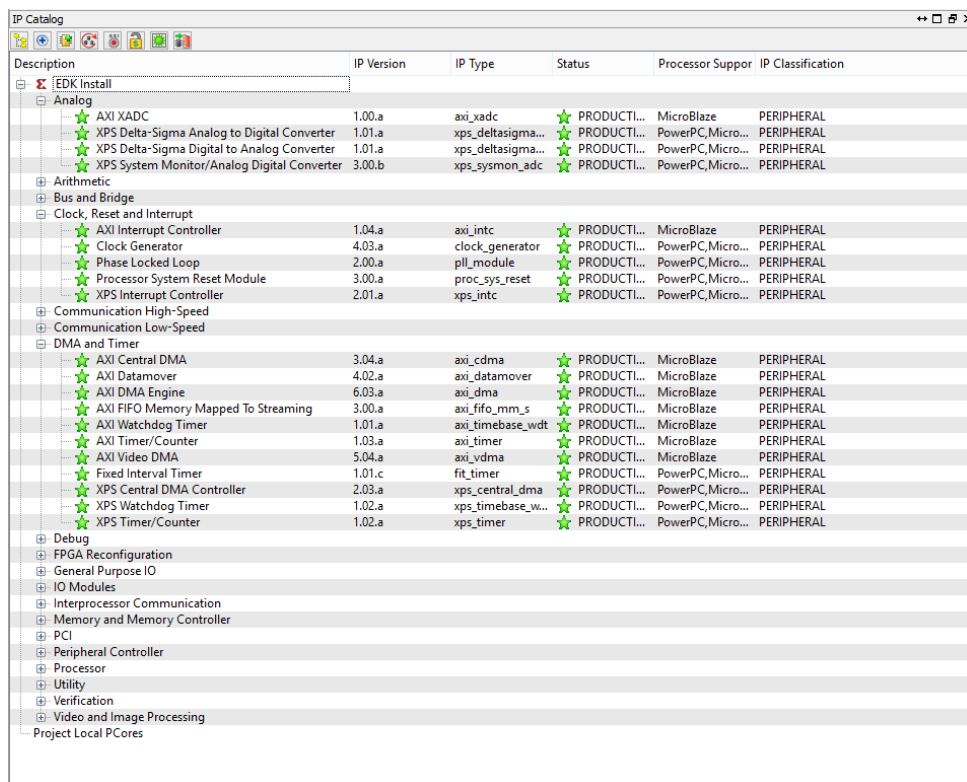
2.3. MicroBlaze

MicroBlaze és el nom que rep la implementació de microcontrolador soft-core del fabricant Xilinx. Aquest és un microcontrolador de tipus RISC (de l'anglès Reduced Instruction Set Computing), amb un nombre reduït d'instruccions de codi màquina. Aquests tipus de microcontroladors estan dissenyats per ser el més ràpid possibles amb la filosofia de tenir instruccions més senzilles d'executar sobre hardware suficientment potent com per executar-les fent servir el mínim nombre de cicles de rellotge. El MicroBlaze fa servir un bus d'interconnexió AXI (Advanced eXtensible Interface) entre els seus perifèrics i memòria, igual que els microcontroladors ARM més moderns. Això fa que sigui relativament fàcil de programar per aquest microcontrolador, utilitzant un llenguatge d'alt nivell¹ com podria ser C.

Generació de Hardware

La generació de hardware per una FPGA del fabricant Xilinx es fa mitjançant una eina del mateix fabricant (Xilinx Platform Studio) que permet incorporar IP cores a un disseny de hardware com si es tractés d'una llista seleccionable. Aquesta eina consta de tot el necessari per triar els IP cores que l'usuari necessita i fer tot el disseny, generació d'arxius i compilació perquè el següent pas sigui programar pel microcontrolador MicroBlaze en llenguatge C/C++.

Un exemple dels IP cores que es poden seleccionar en aquest programari es pot trobar a la Figura 6. Com es pot veure, existeixen gran varietat de IP cores: controladors d'interrupcions, controladors d'accés directe a memòria (DMA) i IP cores per depurar codi sobre MicroBlaze, entre d'altres.



Description	IP Version	IP Type	Status	Processor Support	IP Classification
Analog					
AXI XADC	1.00.a	axi_xadc	PRODUCT...	MicroBlaze	PERIPHERAL
XPS Delta-Sigma Analog to Digital Converter	1.01.a	xps_deltasigma...	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS Delta-Sigma Digital to Analog Converter	1.01.a	xps_deltasigma...	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS System Monitor/Analog Digital Converter	3.00.b	xps_sysmon_adc	PRODUCT...	PowerPC, Micro...	PERIPHERAL
Arithmetic					
Bus and Bridge					
Clock, Reset and Interrupt					
AXI Interrupt Controller	1.04.a	axi_intc	PRODUCT...	MicroBlaze	PERIPHERAL
Clock Generator	4.03.a	clock_generator	PRODUCT...	PowerPC, Micro...	PERIPHERAL
Phase Locked Loop	2.00.a	pll_module	PRODUCT...	PowerPC, Micro...	PERIPHERAL
Processor System Reset Module	3.00.a	proc_sys_reset	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS Interrupt Controller	2.01.a	xps_intc	PRODUCT...	PowerPC, Micro...	PERIPHERAL
Communication High-Speed					
Communication Low-Speed					
DMA and Timer					
AXI Central DMA	3.04.a	axi_cdma	PRODUCT...	MicroBlaze	PERIPHERAL
AXI Datamover	4.02.a	axi_datamover	PRODUCT...	MicroBlaze	PERIPHERAL
AXI DMA Engine	6.03.a	axi_dma	PRODUCT...	MicroBlaze	PERIPHERAL
AXI FIFO Memory Mapped To Streaming	3.00.a	axi_fifo_mm_s	PRODUCT...	MicroBlaze	PERIPHERAL
AXI Watchdog Timer	1.01.a	axi_timebase_wdt	PRODUCT...	MicroBlaze	PERIPHERAL
AXI Timer/Counter	1.03.a	axi_timer	PRODUCT...	MicroBlaze	PERIPHERAL
AXI Video DMA	5.04.a	axi_vdma	PRODUCT...	MicroBlaze	PERIPHERAL
Fixed Interval Timer	1.01.c	fit_timer	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS Central DMA Controller	2.03.a	xps_central_dma	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS Watchdog Timer	1.02.a	xps_timebase_w...	PRODUCT...	PowerPC, Micro...	PERIPHERAL
XPS Timer/Counter	1.02.a	xps_timer	PRODUCT...	PowerPC, Micro...	PERIPHERAL
Debug					
FPGA Reconfiguration					
General Purpose IO					
IO Modules					
Interprocessor Communication					
Memory and Memory Controller					
PCI					
Peripheral Controller					
Processor					
Utility					
Verification					
Video and Image Processing					
Project Local PCores					

Figura 6. Extracte d'una llista de IP cores disponibles per l'usuari

¹ Alt nivell en comparació amb el llenguatge d'assemblador.

Un cop es tenen seleccionats els IP cores que necessita l'usuari pel seu disseny aquests es mostren en una altra part del programari on es pot veure com estaran organitzats segons els busos d'interconnexió entre els mateixos (busos hardware). En aquesta altra vista també es poden configurar paràmetres dels mòduls individualment, triar en quines adreces de memòria estaran situats, etcètera. Aquesta part del programari es pot veure a la Figura 7.

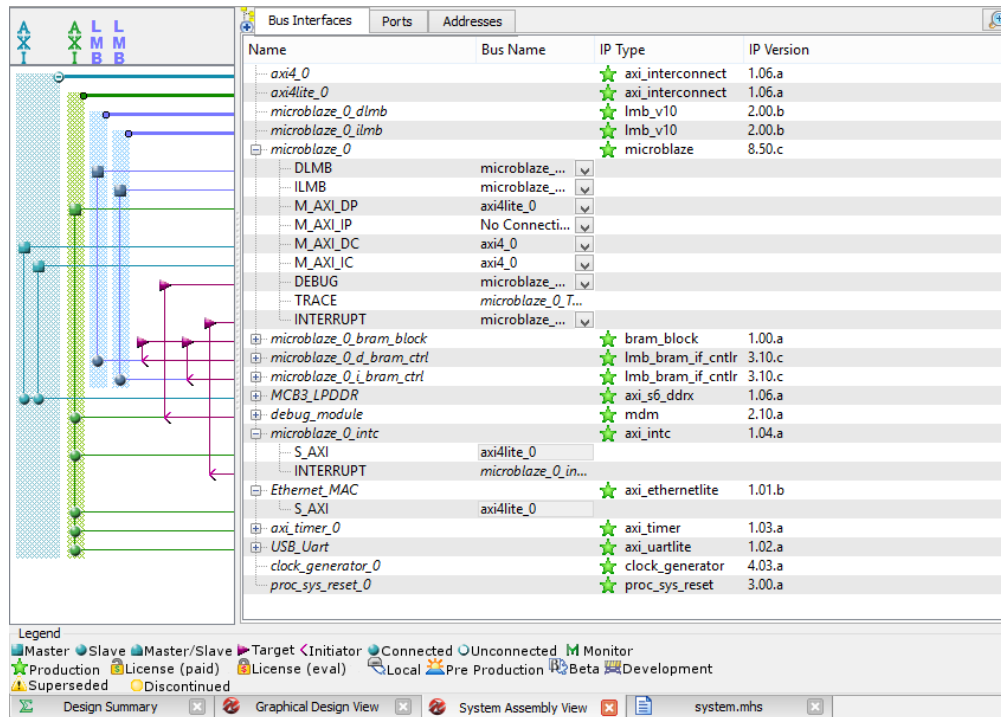


Figura 7. Vista de configuració dels mòduls IP core triats pel projecte en curs

Una vista útil més del programari és la d'assignació d'adreces de memòria del disseny. En aquest cas, com la placa utilitzada té una memòria LPDDR de 64 MB i MicroBlaze no té cap problema per executar codi des de memòria RAM, s'utilitza el IP core per controlar aquesta memòria per tenir un espai pràcticament il·limitat pel codi del programa (tenint en compte que l'aplicació en qüestió és *petita*).

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	microblaze_0_dl...	<input type="checkbox"/>
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	microblaze_0_il...	<input type="checkbox"/>
USB_Uart	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
Ethernet_MAC	C_BASEADDR	0x40E00000	0x40E0FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
microblaze_0_intc	C_BASEADDR	0x41200000	0x4120FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
debug_module	C_BASEADDR	0x41400000	0x4140FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
axi_timer_0	C_BASEADDR	0x41C00000	0x41C0FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
MCB3_LPDDR	C_S0_AXI_BASE...	0xA4000000	0xA7FFFFFF	64M	S0_AXI	axi4_0	<input type="checkbox"/>

Figura 8. Assignació d'adreces de l'espai de memòria de MicroBlaze

Un cop tot el disseny ha estat configurat correctament, per poder utilitzar-lo per carregar-lo a una FPGA l'usuari ha de generar un arxiu conegut com *bitstream*. Aquest és l'arxiu de configuració de la lògica de la FPGA. Un cop es té aquest arxiu es pot carregar a la FPGA i el hardware es configurarà tal com l'usuari l'ha dissenyat. Aleshores, com es té un microcontrolador al sistema, aquest mateix s'ha de programar

per executar el codi de l'usuari. Abans de poder fer aquesta programació, però, s'ha d'exportar el projecte necessari per fer servir aquest hardware generat des de l'entorn de desenvolupament de software.

Programació de Software

Un cop es té generat el programa que farà que la FPGA quan sigui programada es comporti a nivell electrònic segons s'ha dissenyat, cal desenvolupar el codi pel microcontrolador que s'ha decidit incloure al disseny. Això es fa des d'un altre paquet del programari que ofereix el fabricant (Xilinx Software Development Kit). Dins d'aquest programari es requereix importar el projecte que s'ha exportat del programari descrit prèviament. Això porta a l'usuari a tenir un projecte que defineix una plataforma hardware. Un cop es té aquest projecte dins l'espai de treball es requereix crear un altre tipus de projecte que dona accés a tota la propietat intel·lectual necessària (llibreries) per controlar els perifèrics dels IP cores des del software que executarà el MicroBlaze. Aquest segon projecte s'anomena Board Support Package (BSP). Un cop creat el BSP, l'usuari pot crear tants projectes d'aplicació com necessiti, on programarà el seu software com per qualsevol altre microcontrolador, en llenguatge C/C++. Es pot veure una captura de pantalla d'aquest programari a la Figura 9. En aquesta figura, a la pestanya Project Explorer de l'esquerra es poden veure els tres projectes descrits anteriorment: *app*, *standalone_bsp_0* i *xps_hw_platform*.

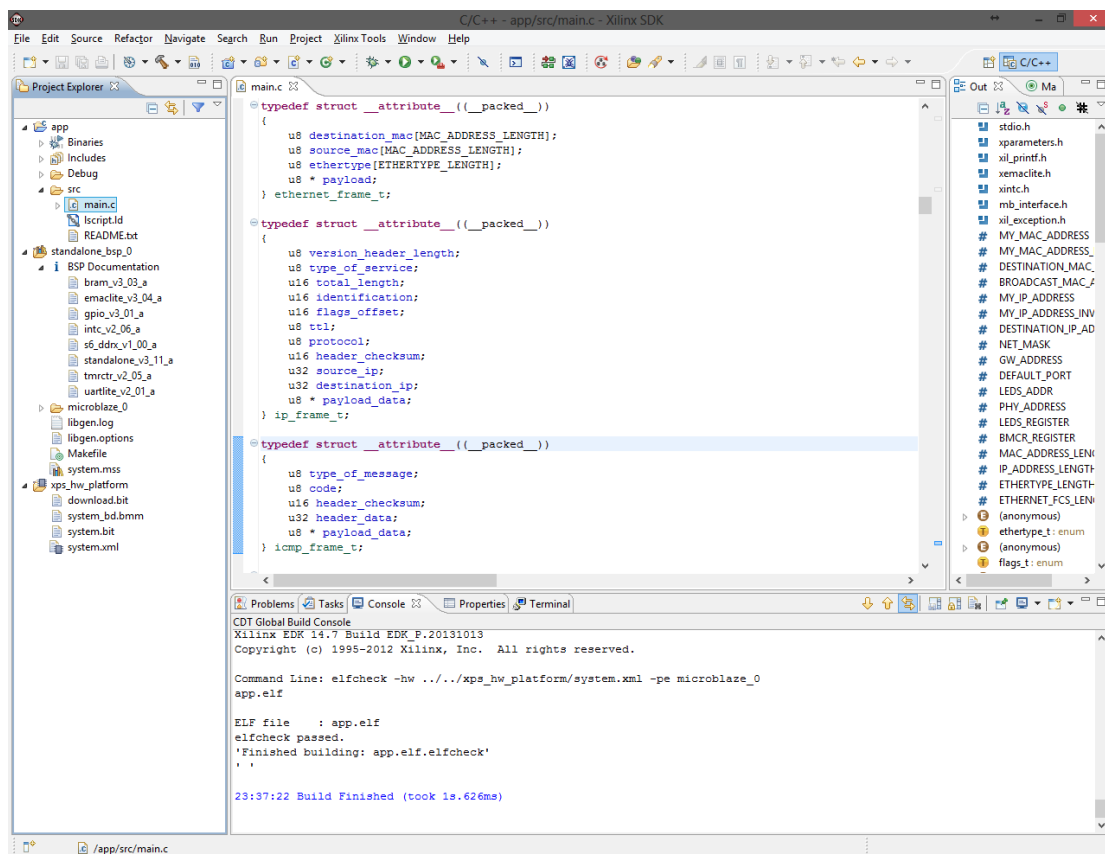


Figura 9. Captura de pantalla del programari Software Development Kit

3. ETHERNET

3.1. Pila de protocol per capes

Com s'ha dit prèviament, Ethernet en sí és un estàndard que només s'aplica sobre les capes física i d'accés al medi en una xarxa d'equips interconnectats. Això vol dir que Ethernet no tracta les dades *útils* de cap manera. Generalment, són els protocols de nivells superiors els que tracten amb les dades. Tots aquests protocols, dividits per les anomenades capes, estan englobats dins de l'estàndard conegut com model OSI (de l'anglès Open Systems Interconnection) de l'Organització Internacional per a l'Estandardització (ISO). Aquest model defineix les capes de protocol que es necessita per connectar-se a la xarxa de xarxes, Internet. Una representació del model OSI es pot trobar a la Figura 10.

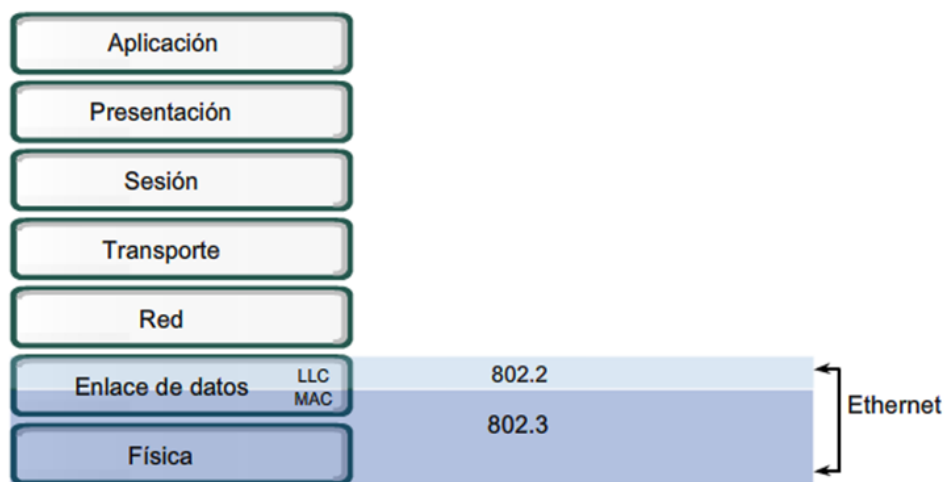


Figura 10. Representació del model OSI

Ethernet és l'encarregat de lligar els sistemes físicament i a nivell de paquets de dades, tal com es descriu a les normatives IEEE 802.3 i 802.2. A la capa física s'hi troba el tipus de connexió que s'ha d'utilitzar per tenir connectivitat Ethernet. Generalment la connectivitat física d'Ethernet són cables amb parells diferencials trenats. Segons la taxa de bits que es vol fer servir, hi ha diferents estàndards dintre de la pròpia connectivitat Ethernet. Generalment els estàndards són de 10/100/1000 Mbit/s. El nombre de parells trenats també depèn de si es vol comunicació full-duplex o n'hi ha suficient amb half-duplex. Aquests estàndards requereixen de diferents tipus de cablejat. Per exemple, a la Figura 11 es pot veure una comparativa entre cables de diferents categories, que serveixen per les diferents velocitats. Les diferències entre uns cables i uns altres generalment són: la quantitat de parells diferencials disponibles, la millora en aïllament entre aquests i la densitat del trenat dels mateixos. Per exemple, per funcionar a 10 Mbit/s no fan falta més que cables de categoria 3, amb 2 parells trenats, mentre que per funcionar a 100 Mbit/s es necessiten cables de categoria 5 com a mínim. També existeixen cables de categories superiors, que permeten velocitats de transmissió superiors als 1000 Mbit/s, arribant en alguns casos fins a 10 Gbit/s.

Category Cable Wiring

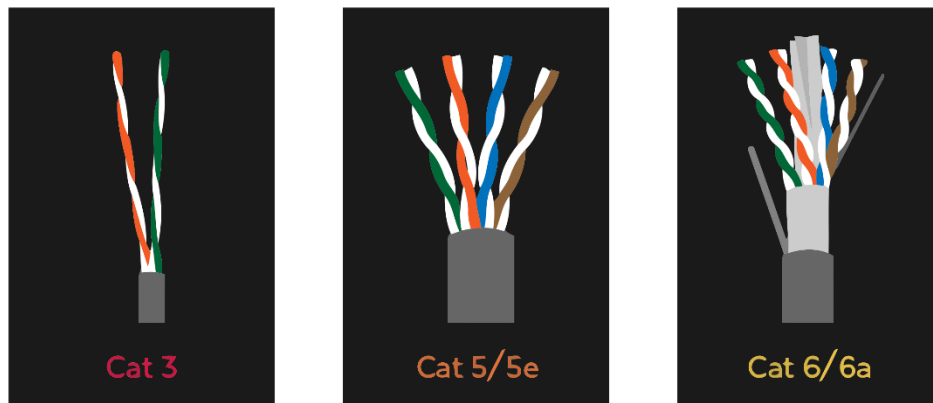


Figura 11. Diferents categories de cablejat per Ethernet que es poden trobar

En aquesta figura es pot veure que els parells dels cables de categoria 3 estan trenats molt poc densament i no tenen cap tipus d'aïllament electromagnètic, ni entre ells ni respecte a l'ambient. A l'altre extrem es troben els cables de categoria 6a, que estan aïllats front a interferències electromagnètiques mitjançant apantallament del cable amb malla de coure, i també entre ells fent servir una pel·lícula metal·litzada d'alumini-mylar.

En una xarxa Ethernet tots els equips connectats tenen la seva pròpia adreça física, també coneguda com adreça d'accés al medi (MAC). Aquesta adreça ha de ser única segons el tipus de dispositiu. Es tracta d'una adreça de 6 bytes on els 3 primers identifiquen el fabricant del dispositiu connectat a la xarxa, i els altres 3 identifiquen el model del dispositiu fabricat per aquell fabricant. Normalment s'expressa en notació hexadecimal (base 16), amb el valor dels bytes separats per dos punts. Per exemple, en el cas de Xilinx, l'adreça de fabricant que li correspon és: 00:35:0a.

Generalment, les xarxes Ethernet segueixen una topologia en estrella, on tots els equips estan connectats a un enrutador que dirigeix els paquets. Però això no és cap requisit. Es pot tenir, per exemple, una connexió punt a punt entre dos equips, o una connexió en anell si hi ha varis equips a connectar. A la Figura 12 es mostren algunes topologies utilitzades en les xarxes.

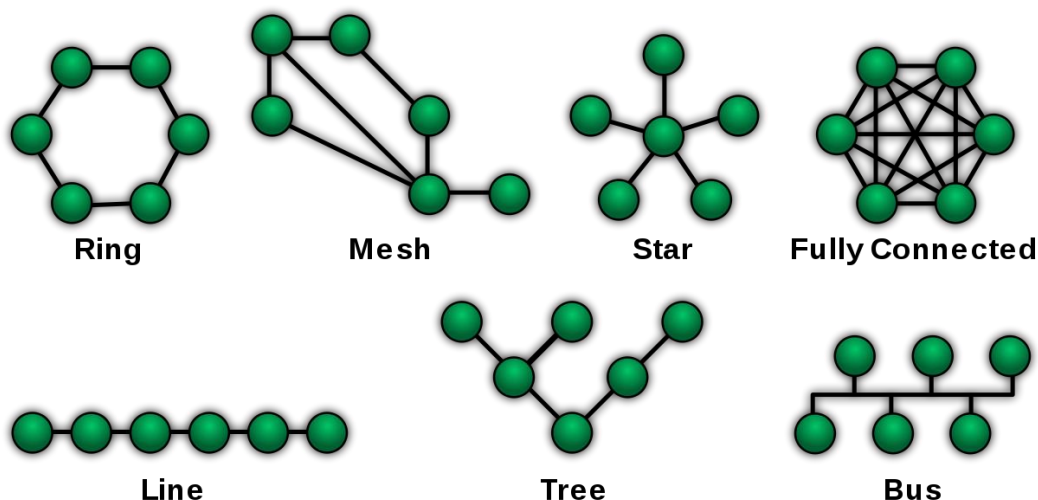


Figura 12. Representació de diverses topologies de xarxes possibles

3.2. Funcionament d'Ethernet a la placa utilitzada

El funcionament d'Ethernet a la placa utilitzada és similar al de tots els dispositius que compleixin l'estàndard. Es tracta bàsicament d'un circuit integrat que interpreta els senyals que arriben a través dels parells trenats del cable. A més d'aquest xip, es necessita una capa de hardware/software que gestiona l'accés al medi (MAC, de l'anglès Medium Access Control). En aquest cas, la capa MAC està integrada a la lògica de la FPGA a través d'un IP core de Xilinx, EmaLite. Aquesta capa és qui es comunica amb el circuit integrat. Una descripció gràfica es pot veure a la Figura 16.

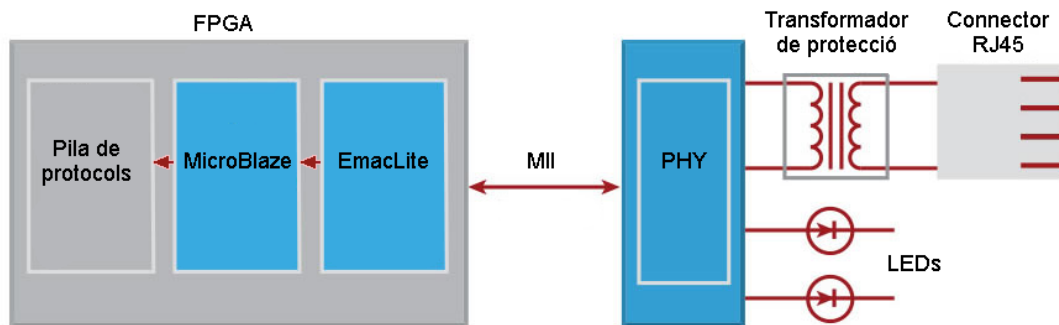


Figura 13. Diagrama que representa el funcionament d'Ethernet a la placa utilitzada

El circuit integrat que fa de PHY (de l'anglès PHYsical layer, en referència a la capa física del model OSI) en la placa utilitzada és el DP83848J de Texas Instruments. Segons el datasheet d'aquest xip, és capaç de comunicar-se amb la xarxa a un màxim de 100 Mbit/s, i gestiona automàticament la negociació de velocitat de transmissió amb la resta de dispositius de la xarxa, segons la normativa IEEE 802.3. Aquest xip es comunica amb la FPGA a través d'un bus sèrie anomenat MII, de l'anglès Media Independent Interface. Aquest és un bus sèrie de 4 bits en paral·lel per transmissió i 4 bits en paral·lel per recepció de dades. Això fa que no necessiti freqüències massa altes per aconseguir aquestes taxes de bits. Utilitzant rellotges de 25 MHz n'hi ha suficient per aconseguir 100 Mbit/s. A més d'aquests 8 bits també es fan servir una sèrie de senyals de gestió d'aquesta comunicació, per gestionar-la el més eficientment i ràpida possible. Al nivell del programa del MicroBlaze, aquesta comunicació la gestiona una llibreria proporcionada per Xilinx. Així doncs, l'usuari és responsable de posar en marxa els perifèrics i configurar-los correctament. Aquest funcionament es descriurà amb més detall al capítol 4.

3.3. Protocols utilitzats

En aquest treball es fan servir varis protocols que són transportats per la xarxa Ethernet. Com s'ha dit anteriorment, Ethernet no treballa amb les dades i per tant només és un *mitjà de transport* de les trames necessàries per intercanviar dades entre els equips connectats a la xarxa. Els protocols que es fan servir estan encapsulats dins d'aquestes trames. Cal notar que aquests protocols no són exclusius d'Ethernet, i que són els que també es fan servir a través de la xarxa de xarxes, Internet. Alguns dels protocols més coneguts i utilitzats són:

- IP (Internet Protocol), engloba tots els protocols necessaris per tal que la xarxa de xarxes pugui funcionar.
- ARP (Address Resolution Protocol), que serveix per descobrir les adreces físiques (adreces MAC) que té un dispositiu a la xarxa.
- DHCP (Dynamic Host Configuration Protocol), que serveix per tal de que els clients en una xarxa puguin ser configurats automàticament per l'amfitrió per poder connectar-se a Internet i ser identificats pels altres equips d'aquesta xarxa.
- TCP (Transfer Control Protocol), és un dels protocols d'intercanvi de dades a Internet més utilitzats. Conté funcions de reenviament i detecció/control d'errors en les comunicacions. És un protocol orientat a connexió, és a dir, tots dos equips s'han de posar d'acord per comunicar-se, i tots dos han de confirmar les dades rebudes.
- UDP (User Datagram Protocol), un altre dels protocols més utilitzats a Internet. No és orientat a connexió i permet enviar i rebre dades a més altes velocitats que TCP, degut a què les dades no són crítiques. Aquest protocol és més eficient que TCP perquè *desaprofita* menys bytes transmesos degut a què no es controlen errors ni recepció de dades.
- ICMP (Internet Control Message Protocol), és un dels protocols més bàsics que permet regular el tràfic a la xarxa. És el protocol fonamental per poder, per exemple, comunicar missatges d'error, notificar que un servei determinat no està disponible, o provar si un equip es troba a la xarxa o no.

A continuació es descriuen amb cert detall els protocols que s'utilitzaran més endavant al treball, en la secció 4.

Trames Ethernet

Per tal de poder transmetre dades a través del medi físic es requereix que les trames Ethernet tinguin un format específic. En aquest cas, és un format simple on s'ha d'enviar:

- Un preàmbul que serveix per sincronitzar els equips que s'estiguin comunicant en aquell moment.
- L'adreça física (MAC) de destí de les dades encapsulades en aquella trama.
- L'adreça física (MAC) d'origen de les dades encapsulades en aquella trama.
- El tipus de dades (EtherType) encapsulades en aquella trama. Els valors són estandarditzats. Per exemple el valor 0x0800 indica que les dades encapsulades corresponen a IPv4, mentres que 0x0806 indica que les dades són ARP.
- Les dades encapsulades. Si la llargària total del paquet encapsulat no arriba als 46 bytes s'ha de omplir amb valors nuls.
- Una seqüència de control de trama (FCS, de l'anglès Frame Check Sequence), consistent del resultat d'executar un algorisme de comprovació de redundància cíclic (CRC) de 32 bits (4 bytes) sobre tota la trama. Generalment aquest algorisme és transparent a l'usuari i s'executa a nivell de hardware a la capa

MAC del dispositiu connectat a la xarxa.

A la Figura 13 es pot veure una representació gràfica d'aquesta trama.

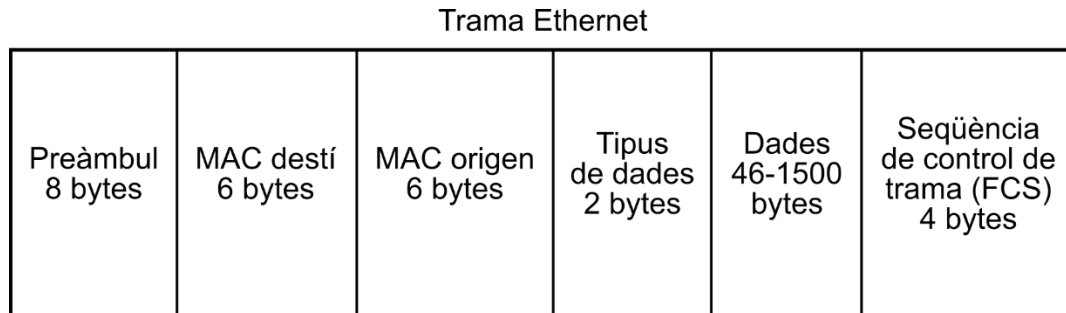


Figura 14. Diagrama que representa la estructura d'una trama Ethernet

A continuació es descriuen els protocols que van encapsulats dins de la trama Ethernet, i que permeten intercanviar dades *útils* entre els equips connectats a la xarxa.

Paquets IPv4

El protocol bàsic que s'utilitza per l'intercanvi de dades tant dins d'una xarxa local com a la xarxa de xarxes és IP versió 4. Aquest és un protocol extens que permet intercanvi de dades dintre d'una xarxa on tots els equips connectats estan adreçats amb una adreça IP, diferent de l'adreça física, que permet identificar-los independentment del dispositiu, i permet organització lògica dels equips, mitjançant formació de xarxes i subxarxes. L'adreça IP consta de 4 bytes de llargària. En aquest protocol els paquets de dades se solen separar entre una capçalera, que dóna informació sobre l'origen i destí del paquet entre d'altres, i les dades encapsulades en aquest paquet. La capçalera dels paquets IPv4 ha de contenir, com a mínim:

- La versió de protocol IP que s'està utilitzant (4 o 6), 4 bits.
- La llargària de la capçalera IP, comptada en paraules de 4 bytes (32 bits), 4 bits. Això dóna un màxim de 15, per tant $15 \cdot 32 \text{ bits} = 480 \text{ bits} = 60 \text{ bytes}$. És a dir, una capçalera de IPv4 mai hauria de ser més llarga de 60 octets.
- Un camp anomenat Serveis Diferenciats de 6 bits. Aquest camp permet distingir entre tipus de trames IPv4, per tractar-les amb major o menor prioritat, per exemple.
- Un camp anomenat Notificació de Congestió Explícita, de 2 bits. Aquest camp serveix per, si hi ha congestió a la xarxa, es pugui notificar als equips de destí i origen, sense pèrdua de paquets. Generalment aquests protocols no s'utilitzen amb freqüència.
- La llargària total del paquet IP, incloent les dades, comptant bytes. Aquest és un camp de 16 bits, i per tant permetria que els paquets tinguin una llargària màxima de fins a 65535 bytes.
- Un camp anomenat Identificació, que permet identificar els paquets, de 16 bits. Com que aquesta identificació hauria de ser única pel que duri la connexió i transmissió entre els dos equips de la xarxa, això limita altament les velocitats de transmissió, així que generalment no s'utilitza per res.
- El camp anomenat Flags, de 3 bits. Aquest camp té un bit que indica si el paquet IPv4 ha hagut de ser fragmentat en vàries transmissions diferents (MF, de l'anglès More Fragments), si les dades no cabessin en un sol paquet, i per tant han de venir més fragments. També té un bit (DF, de l'anglès Don't Fragment) segons el qual l'originador del paquet pot demanar que no se separin les dades durant la transmissió, saltant pels diferents punts de la xarxa. El tercer bit està reservat, segons la especificació.

- El camp que indica, si haguessin d'haver més fragments, quin nombre de fragment s'està tractant en el paquet actual. Aquest camp té una llargària de 13 bits, però el seu número es mesura en unitats de 8 bytes. El primer paquet de la transmissió sempre comença tenint un 0.
- El *temps de vida* que li queda al paquet, 8 bits. Aquest temps de vida es mesura restant d'un nombre màxim. Es resta un per cada salt que fa aquest paquet a la xarxa, entenent com salt passar d'un equip a un altre. Això impedeix que un paquet quedi donant voltes per la xarxa de forma indefinida.
- Un camp que defineix el protocol de les dades encapsulades en aquest paquet IPv4, de 8 bits. Alguns exemples són: 0x01 pel protocol ICMP, 0x11 per UDP, 0x06 per TCP.
- El camp que comprova la suma de verificació de la capçalera del paquet en qüestió, de 16 bits. Aquest valor s'obté calculant el complement a u, de la suma de tota la capçalera en complement a u, assumint com si la capçalera estigués formada per paraules de 16 bits.
- L'adreça IP d'origen del paquet, de 32 bits (4 bytes).
- L'adreça IP de destí del paquet, de 32 bits (4 bytes).
- Per últim, el camp de les dades encapsulades en aquest paquet, de llargària variable.

A la Figura 14 es pot veure una representació gràfica d'aquest paquet.

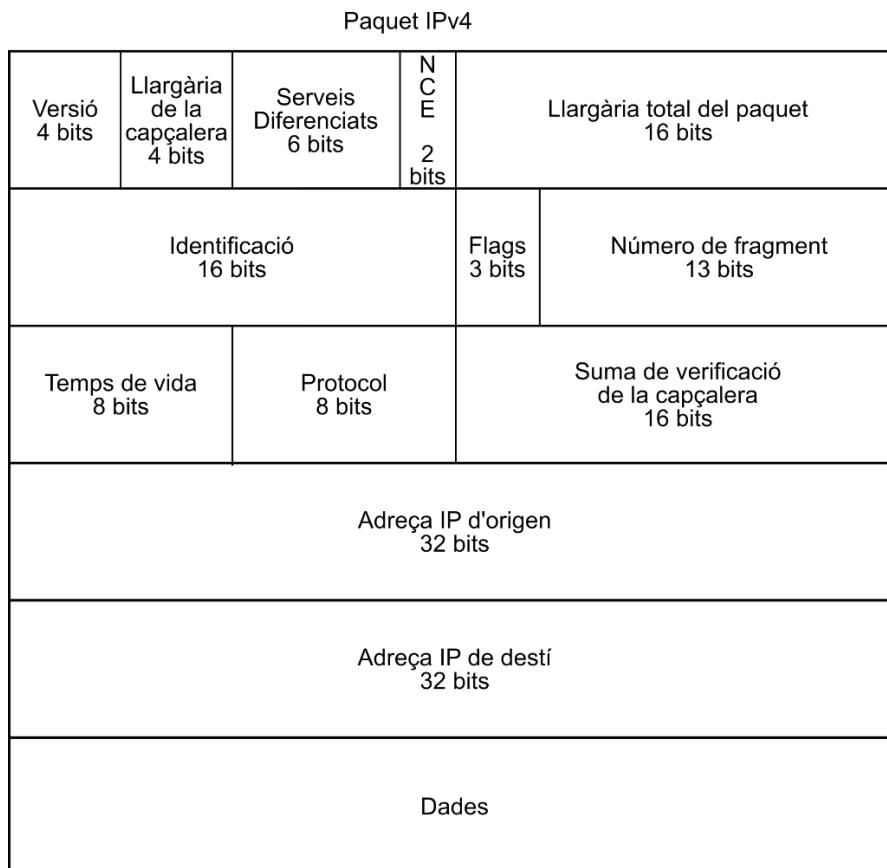


Figura 15. Diagrama que representa la estructura d'un paquet IPv4

A continuació es descriuen els protocols que es faran servir més endavant, encapsulats dins del paquet IPv4.

Paquet ARP

El protocol ARP serveix per resoldre les adreces físiques dels dispositius connectats a una xarxa. Per utilitzar aquest protocol, l'equip que necessiti saber la direcció física d'un altre pot emetre un paquet Ethernet broadcast (dirigit a tots els dispositius d'aquella xarxa física), fent servir com a adreça MAC de destí l'adreça FF:FF:FF:FF:FF:FF. Els paquets ARP es componen de les següents dades:

- Medi a què s'està accedint, 16 bits. Per una xarxa Ethernet, aquest valor ha de ser 0x0001.
- Tipus de protocol que s'utilitza a la xarxa a què s'accedeix, 16 bits. Per IPv4, el valor ha de ser 0x0800. Normalment s'utilitzen els mateixos valors que el EtherType de la trama Ethernet.
- Llargària de les adreces físiques (MAC), comptant bytes, 8 bits. Per Ethernet, aquest camp ha de ser 6.
- Llargària de les adreces lògiques (IP), comptant bytes, 8 bits. Per IPv4, aquest camp ha de ser 4.
- Codi d'operació, 16 bits. Codi que defineix què ha de fer el receptor amb aquest paquet. Pot ser una petició (1) o una resposta (2).
- Adreça física de qui envia el paquet, llargària 6 bytes.
- Adreça lògica de qui envia el paquet, llargària 4 bytes.
- Adreça física del destinatari del paquet, llargària 6 bytes. Quan no se sap i es vol descobrir l'adreça física del dispositiu ubicat a una determinada adreça lògica, aquest camp es plena amb valors nuls.
- Adreça lògica del destinatari del paquet, llargària 4 bytes.

A la Figura 14 es pot veure una representació gràfica d'aquest paquet.

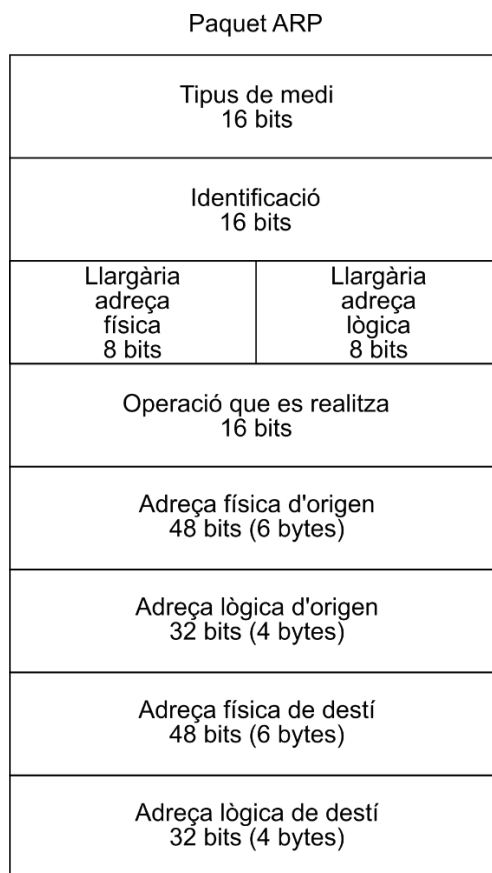


Figura 16. Diagrama que representa la estructura d'un paquet ARP

Paquet ICMP

El protocol ICMP serveix per regular el tràfic a la xarxa. Això es refereix a, per exemple, notificar a l'originador d'una transmissió de la pèrdua del seu paquet, o que el paquet no ha pogut arribar al destinatari en el nombre de salts (temps de vida) establert. Entre d'altres, aquest protocol permet fer peticions i respostes de paquets anomenats eco. Això permet comprovar l'estat de la xarxa, i en quin temps es poden enviar i rebre paquets d'un lloc a un altre. Aquesta funció serà la que s'utilitzarà extensament als capítols 4 i 5 per fer les proves comparant entre les diferents implementacions de piles de protocol. El paquet ICMP està compostat per les següents dades:

- Tipus de missatge que es fa servir, 8 bits. Alguns d'aquests tipus són, per exemple: 0 per una resposta a un paquet eco, 3 quan el destinatari d'algun paquet és inabastable, 8 per una petició d'eco, i 11 per informar que un paquet ha arribat a 0 del seu temps de vida.
- Codi que acompanya el tipus de missatge, 8 bits. Aquest codi dona més detalls sobre el tipus de missatge del que s'està informant.
- Suma de verificació del paquet ARP, 16 bits. És el mateix algoritme que el que comprova la capçalera IPv4, però en aquest cas s'aplica a tot el paquet ARP.
- Identificador i nombre de seqüència, 32 bits, generalment utilitzats per associar una petició d'eco amb una resposta d'eco.
- Altres dades. Aquí s'hi podria incloure una data i hora d'enviament del missatge (si es tractés d'una petició d'eco), per exemple. El receptor de la petició d'eco retorna aquestes dades sense modificació. Això fa que quan l'originador de la petició d'eco inicial rebí les dades sàpiga quant de temps ha passat, degut a què la seva referència de data i hora no ha canviat, així que les pot comparar directament.

Paquet ICMP

Tipus 8 bits	Codi 8 bits	Suma de verificació del paquet 16 bits
Identificador 16 bits		Número de seqüència 16 bits
Dades		

Figura 17. Diagrama que representa la estructura d'un paquet ICMP

4. IMPLEMENTACIÓ PRÀCTICA

Abans de descriure les parts de software de la implementació pràctica d'aquest treball caldria establir un escenari comú de hardware utilitzat, per tal de poder fer comparacions entre les diferents implementacions de software sense afavorir cap d'ambdues. Així doncs, a continuació es detallarà el projecte de creació de hardware amb el programari anteriorment descrit (secció 2.3.1).

El primer pas a seguir per tal d'aconseguir crear un sistema de hardware amb IP cores de Xilinx és crear un projecte nou des del programari XPS, com mostra la Figura 18.

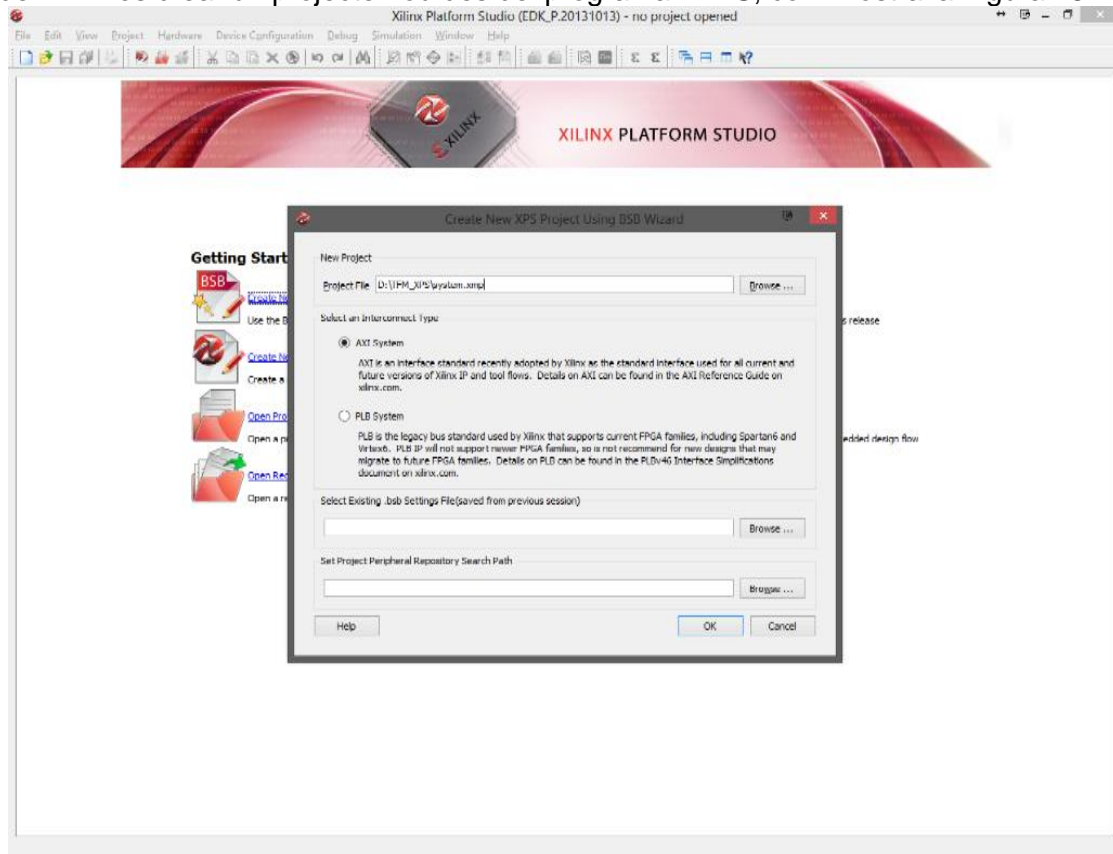


Figura 18. Creació d'un projecte nou al programari XPS

Un cop creat el projecte per aquest hardware que es vol dissenyar, el programari demana a l'usuari que triï la placa de desenvolupament (si fos necessari) que vol fer servir, o si fa servir una FPGA sobre un sistema dissenyat apart. En el cas d'aquest treball es pot seleccionar directament la placa del fabricant Avnet utilitzada. Això vol dir que el XPS configura el projecte com si es tractés de la FPGA Spartan6, del model utilitzat en la placa LX9 MicroBoard de Avnet, amb una freqüència de rellotge de referència de la FPGA de 66.67 MHz.

En el cas del treball se selecciona un sistema format per un sol MicroBlaze, tot i que sobre aquesta FPGA es poden implementar sistemes de fins a 2 MicroBlaze funcionant en paral·lel. A més d'això, com a estratègia d'optimització dels recursos de la FPGA se selecciona Throughput (traduït literalment de l'anglès rendiment, en aquest cas es refereix a capacitat de procés), perquè:

1. Per l'aplicació que es vol fer el codi s'executarà des d'una memòria RAM externa, i per tant no és necessari que quedi espai pel codi de MicroBlaze.
2. És imprescindible que el codi s'executi el més ràpid possible, encara que sigui a costa d'utilitzar més recursos de sistema.

Així doncs, aquest punt del procés es mostra a la Figura 19.

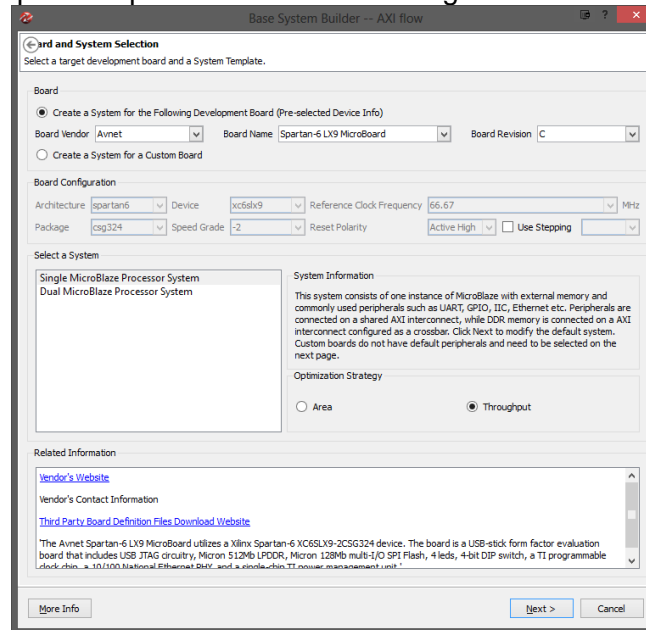


Figura 19. Selecció de la FPGA utilitzada i el sistema que s'hi vol implementar

En el pas següent, el XPS presenta a l'usuari totes les opcions que té preconfigurades per la placa utilitzada pel projecte, si n'hagués algunes. En aquest cas, mostra tots els perifèrics que hi ha a la placa i es podrien utilitzar, com ara el cas d'Ethernet, la memòria RAM LPDDR externa a la FPGA i un canal de comunicació sèrie (UART) sobre USB, per exemple. Per aquest treball es trien els següents dispositius:

- Ethernet, per poder fer servir la connectivitat.
- LPDDR, per no tenir limitacions de codi de software.
- UART sobre USB, per poder mostrar missatges a l'ordinador. Aquest mòdul no és necessari per un funcionament correcte del software que es desenvoluparà, és només per mostrar missatges.
- Un mòdul timer, necessari per la llibreria LwIP que s'explicarà més endavant.

Aquest punt del procés es pot veure a la Figura 20.

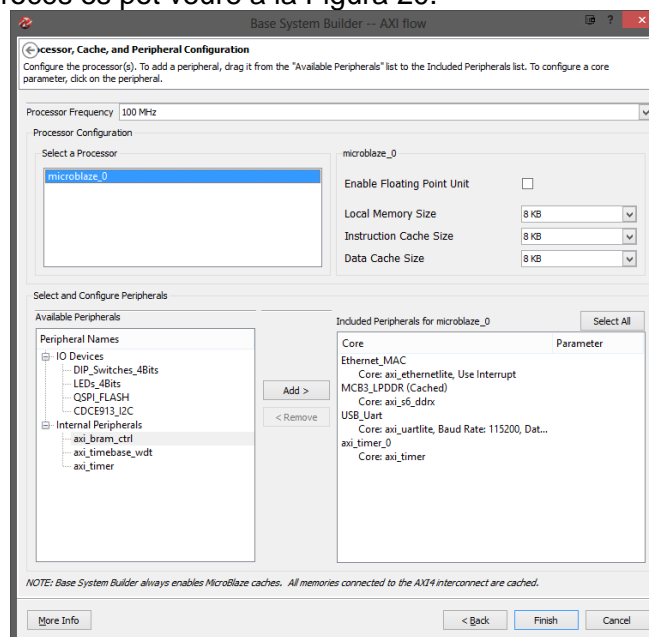


Figura 20. Selecció dels mòduls predefinits per la placa en qüestió que s'està utilitzant

Un cop passat aquest punt ja s'està pràcticament a punt per generar els fitxers necessaris. L'usuari arriba al mateix punt que el que es mostrava a les Figures 7 i 8. En aquest cas no cal afegir altres IP cores, ni fer modificacions sobre els que ja hi ha, així que es pot procedir a exportar el bitstream i tots els arxius necessaris per poder programar el software per MicroBlaze des del programari Xilinx Software Development Kit, tal com s'havia mencionat a la secció 2.3.2. Un cop el pas de generar el bitstream i preparar l'entorn de desenvolupament ha finalitzat l'usuari es troba amb una vista similar a la de la Figura 21, amb el programari Software Development Kit.

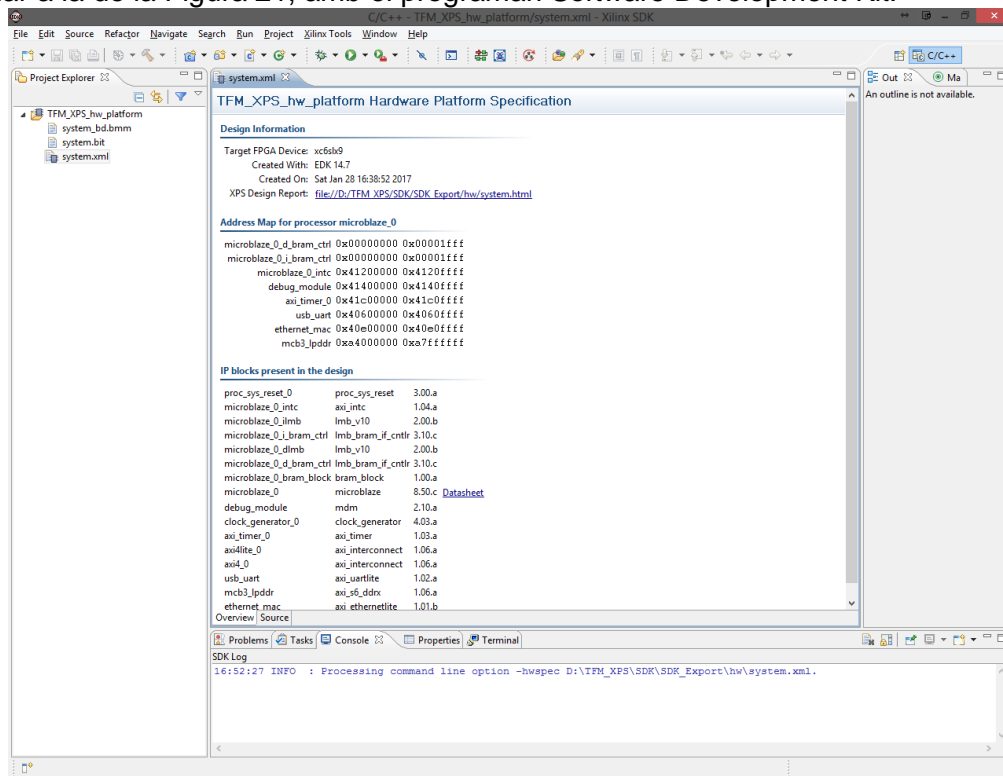


Figura 21. Vista després d'exportar el disseny del XPS

A partir d'aquí, els passos a seguir són pocs per començar a crear el codi de l'aplicació. En primer lloc, com s'havia dit a la secció 2.3.2., s'ha de crear un Board Support Package, per poder utilitzar llibreries de software que controlin els IP cores del disseny. Aquest pas es pot veure a la Figura 22.

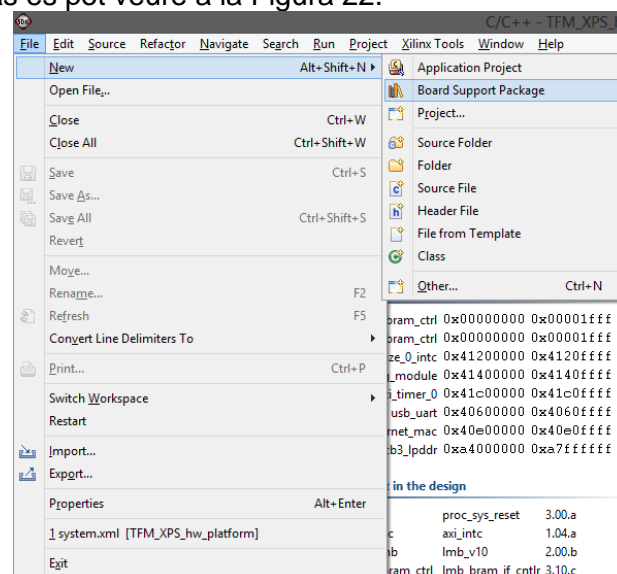


Figura 22. Creació d'un nou Board Support Package

Seguint aquest pas a l'usuari li apareix la finestra de la Figura 23. Aquí es permet triar un nom pel BSP i fer servir una capa de software per sobre de MicroBlaze. La opció standalone és una opció bàsica, que permet accedir als recursos del processador i poca cosa més. La opció xilkernel, en canvi, és una espècie de nucli de sistema operatiu que permet fer planificació de tasques, sincronització, fils d'execució, etcètera. Aquesta opció no interessa pel treball, així que s'ha de seleccionar la opció standalone.

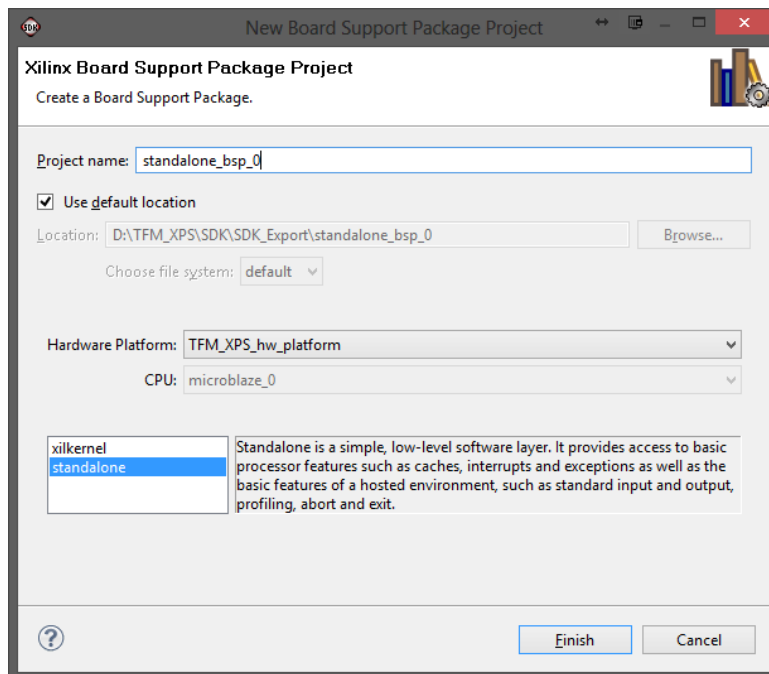


Figura 23. Selecció de la capa de software d'un BSP

Un cop seleccionada la capa de software que duu aquest BSP, l'usuari es troba amb la finestra de la Figura 24. Aquesta figura permet incloure algunes llibreries estàndard que dóna el fabricant, com per exemple la seva implementació de LwIP que serà la que s'utilitzarà a la secció següent. Si no es necessiten fer més modificacions al BSP, aquest ja es pot crear prement OK.

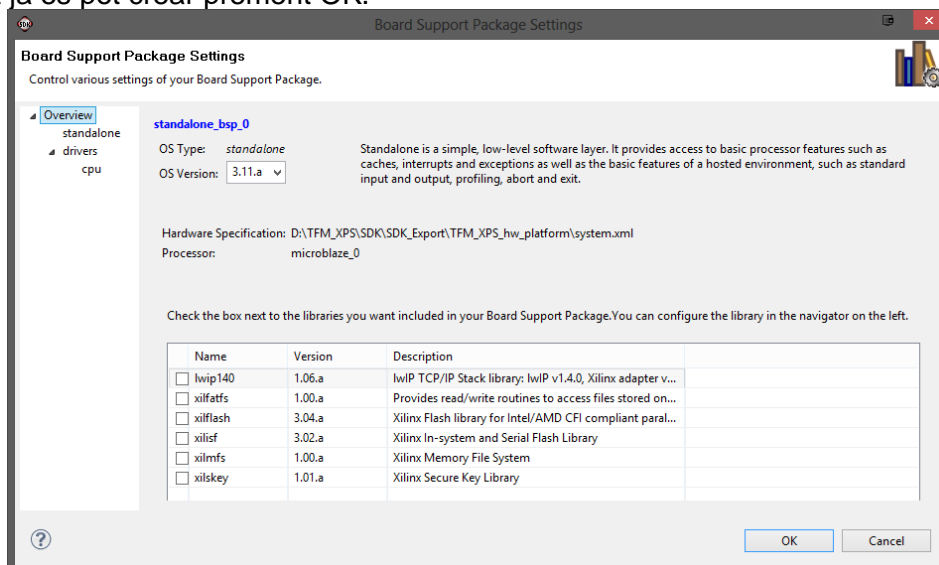


Figura 24. Configuració dels elements que integren un BSP

Un cop creat el BSP, l'única cosa que falta per poder començar a programar l'aplicació és crear un projecte d'aplicació. Això es fa des del mateix menú que el que s'havia creat el BSP, com es pot veure a la Figura 25.

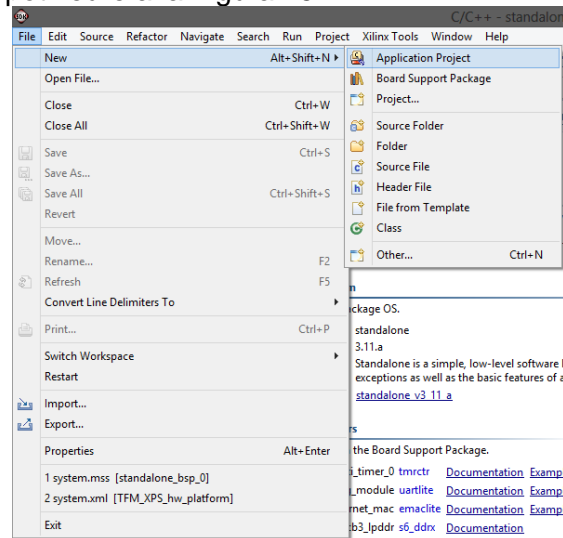


Figura 25. Creació d'un nou projecte d'aplicació

Un cop creat aquest nou projecte d'aplicació apareix la finestra de la Figura 26, on l'usuari ha de donar un nom al projecte, i triar quina plataforma de hardware vol fer servir, i quin BSP vol fer servir amb aquesta plataforma de hardware.

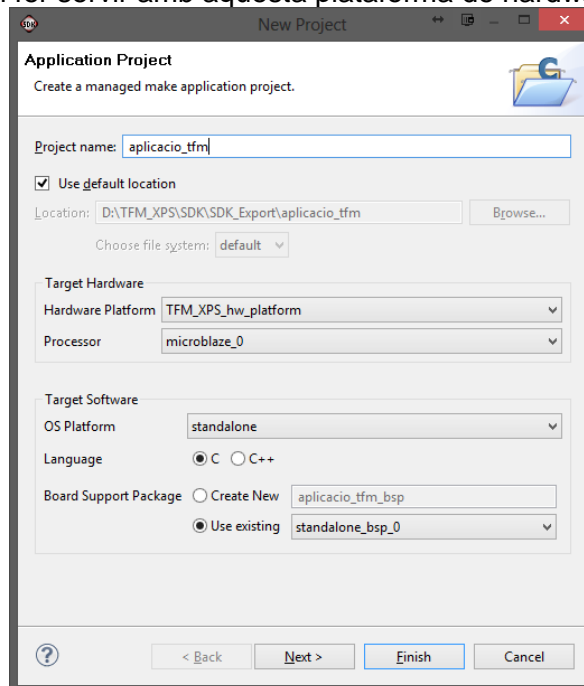


Figura 26. Selecció de BSP que es vol fer servir pel projecte d'aplicació

Quan ja s'ha donat un nom al projecte d'aplicació i s'ha triat fer servir el BSP que s'havia creat anteriorment, prement Next l'usuari es troba amb una pantalla de l'assistent on pot triar una plantilla d'aplicació de les que hi ha disponibles. D'aquestes opcions, per aquest treball, es tria Empty Application perquè el codi que s'afegirà no s'assembla a cap de les opcions oferides.

4.1. LwIP

La llibreria LwIP és una llibreria de codi obert àmpliament coneguda que implementa la pila de protocols necessària per tal de poder establir una connexió amb Internet (TCP/IP) generalment des d'un sistema incrustat amb poca memòria disponible (tant memòria RAM com memòria de programa). És una llibreria escrita en codi C altament optimitzat que, per si sola, requereix al voltant de 40 kilobytes d'espai de memòria de programa i una desena de kilobytes de memòria RAM (segons la utilització i requeriments de la xarxa). Aquesta llibreria és totalment independent del medi utilitzat per connectar-se a la xarxa, així que només processa els paquets amb informació *útil* per l'aplicació en qüestió. Entre d'altres, permet fer servir pràcticament tots els protocols utilitzats a Internet:

- IP, de l'anglès Internet Protocol, sigui la versió 4 o la versió 6.
- ICMP, de l'anglès Internet Control Message Protocol.
- UDP, de l'anglès User Datagram Protocol.
- TCP, de l'anglès Transfer Control Protocol.
- ARP, de l'anglès Address Resolution Protocol.
- IGMP, de l'anglès Internet Group Management Protocol.
- PPPoE, de l'anglès Point-to-Point Protocol over Ethernet.
- DNS, de l'anglès Domain Name System.
- DHCP, de l'anglès Dynamic Host Configuration Protocol.

Així doncs, al ser una llibreria de codi obert és extensament utilitzada en aplicacions de tot tipus. La filosofia de desenvolupament d'aquesta llibreria és que sigui totalment modular. Amb un nucli basat fortament en IP (sigui versió 4 o 6) s'hi munten mòduls que gestionen la resta de protocols. LwIP pretén ser una llibreria relativament senzilla d'utilitzar, i que un cop configurada (en temps de compilació) es gestiona més o menys *automàticament*. Per tant, el bucle principal d'una aplicació programada en C per un sistema incrustat amb un sol processador i un sol fil d'execució que faci servir LwIP, podria arribar a ser tan simple com:

```
while(1)
{
    /* consulta el driver de la xarxa (en aquest cas Ethernet), rep
    trames que podrien no haver estat llegides fins ara, reserva
    memòria per aquestes i les gestiona */
    poll_driver(netif);
    /* comprova si algun dels temps d'espera ha vençut i ha
    d'executar tasques d'algun dels protocols nucli */
    sys_check_timeouts();
}
```

Codi 1. Codi d'un possible bucle mínim d'aplicació que utilitza la llibreria LwIP

Com es pot veure al Codi anterior, aquesta llibreria es basa en tenir un temporitzador que s'activa cada certa quantitat de temps. Aquesta quantitat ha de ser suficient perquè el processador tingui temps per resoldre totes les tasques que la llibreria té pendents (paquets que encara esperen ser processats, i tasques periòdiques similars), però tampoc pot ser massa gran perquè sinó les funcions de codi que gestionen algunes cues o alguns processos interns tardarien molt en tornar a executar-se la següent vegada, introduint retards en les funcions de xarxa del dispositiu en qüestió. En aquest cas, la llibreria fa servir dos tipus de temporitzadors:

- Un temporitzador ràpid, que generalment es recomana sigui de com a molt 250 ms de període.
- Un temporitzador lent, que generalment té un període de 500 ms (el doble del temporitzador ràpid). El fet que sigui un múltiple del temporitzador ràpid no és

casualitat, doncs així es fa servir només un temporitzador hardware. Això permet que els altres temporitzadors disponibles a la plataforma es facin servir per altres coses.

En aquest cas, la plataforma hardware (MicroBlaze) ja s'ha dissenyat sabent això, i només té un temporitzador hardware disponible. Aleshores només cal escriure el codi per fer servir aquesta llibreria. A continuació es descriurà el codi en C escrit per aconseguir que l'aplicació arrenqui i sigui capaç de respondre a peticions d'eco des d'un ordinador de la mateixa xarxa.

Funció principal

A la funció principal, abans d'executar codi de la llibreria LwIP s'han d'inicialitzar els perifèrics necessaris del processador. En aquest cas són els perifèrics en si (la capa Ethernet MAC i el temporitzador) i les interrupcions de sistema pels mateixos perifèrics. Un cop donats aquests dos passos, cal posar en marxa la interfície de xarxa de la llibreria LwIP. Fet això la llibreria *funciona sola*. El procés de contestar a una petició d'eco és dut a terme per les capes baixes de la llibreria i això fa que sigui transparent per l'usuari de la llibreria (que en canvi és responsable de gestionar les comunicacions TCP, si existissin). En el Codi 2 a continuació es pot veure la funció principal (main) de l'aplicació.

```
int main(void)
{
    /* neteja la consola UART i imprimeix missatge */
    xil_printf("%c[2J",27);
    xil_printf("----- TFM - Marko Peshevski - versio LwIP -----\\r\\n");

    inicialitza_temporitzador();
    inicialitza_interrupcions();
    inicialitza_lwip();

    xil_printf("Arrenca aplicacio que respon a eco... ");
    arrenca_app();
    xil_printf("Aplicacio en marxa\\r\\n");

    /* activa les interrupcions a nivell de processador */
    Xil_ExceptionEnable();
    while (1)
    {
        /* rep dades de la interfície de xarxa (driver ethernet MAC) */
        xemacif_input(&interficie_xarxa);

        /* consulta temporitzadors i executa tasques periodiques */
        if (temporitzador_tcp_rapid)
        {
            tcp_fasttmr();
            temporitzador_tcp_rapid = 0;
        }
        if (temporitzador_tcp_lent)
        {
            tcp_slowtmr();
            temporitzador_tcp_lent = 0;
        }
    }
    return 0;
}
```

Codi 2. Funció principal de l'aplicació amb llibreria LwIP

Inicialització del temporitzador

La funció que inicialitza el temporitzador es pot consultar al Codi 3 a continuació.

```
void inicialitza_temporitzador(void)
{
    /* especifica quin nombre de cicles ha de comptar el temporitzador abans
    d'interrompre. amb un rellotge de 100 MHz -> 0.01us per cada periode per
    tant, per 250ms -> 250000000 periodes */
    XTmrCtr_SetLoadReg(XPAR_TMRCTR_0_BASEADDR, 0, 250000000);

    /* neteja el bit d'interrupcio i carrega el valor a comptar al registre
    del temporitzador */
    XTmrCtr_SetControlStatusReg(XPAR_TMRCTR_0_BASEADDR, 0,
                                XTC_CSR_INT_OCCURED_MASK |
                                XTC_CSR_LOAD_MASK);

    /* inicia el temporitzador i la interrupcio i compta descendent */
    XTmrCtr_SetControlStatusReg(XPAR_TMRCTR_0_BASEADDR, 0,
                                XTC_CSR_ENABLE_TMR_MASK |
                                XTC_CSR_ENABLE_INT_MASK |
                                XTC_CSR_DOWN_COUNT_MASK);
}
```

Codi 3. Funció d'inicialització del temporitzador

Gestió d'interrupció del temporitzador

La funció que gestiona el temporitzador es pot consultar al Codi 4 a continuació.

```
void handler_temporitzador(void *p)
{
    /* neteja el bit d'interrupcio i carrega el valor a comptar al registre
    del temporitzador */
    XTmrCtr_SetControlStatusReg(XPAR_TMRCTR_0_BASEADDR, 0,
                                XTC_CSR_INT_OCCURED_MASK |
                                XTC_CSR_LOAD_MASK);

    /* inicia el temporitzador i la interrupcio i compta descendent */
    XTmrCtr_SetControlStatusReg(XPAR_TMRCTR_0_BASEADDR, 0,
                                XTC_CSR_ENABLE_TMR_MASK |
                                XTC_CSR_ENABLE_INT_MASK |
                                XTC_CSR_DOWN_COUNT_MASK);

    /* neteja el bit d'interrupcio del registre */
    XIntc_AckIntr(XPAR_INTC_0_BASEADDR, (1 << XPAR_INTC_0_TMRCTR_0_VEC_ID));

    /* actualitza valors dels temporitzadors rapid i lent */
    temporitzador_tcp_rapid = 1;
    both_timers = !both_timers;
    if (both_timers)
    {
        temporitzador_tcp_lent = 1;
    }
}
```

Codi 4. Funció de gestió d'interrupció del temporitzador

Inicialització de les interrupcions

La funció que inicialitza el perifèric d'interrupcions es pot consultar al Codi 5 a continuació.

```
void inicialitza_interrupcions(void)
{
    /* inicialitza el periferic Intc de MicroBlaze */
    XIntc_Initialize(&intc, XPAR_INTC_0_DEVICE_ID);

    /* arrenca el periferic en mode real (no simulacio) */
    XIntc_Start(&intc, XIN_REAL_MODE);

    /* especifica la funcio que gestiona les interrupcions del temporitzador.
    la del ethernet MAC es fa des de lwip */
    XIntc_RegisterHandler(XPAR_INTC_0_BASEADDR,
                        XPAR_INTC_0_TMRCTR_0_VEC_ID,
                        (XInterruptHandler) handler_temporitzador,
                        &intc);

    /* habilita la interrupcio del temporitzador */
    XIntc_EnableIntr(XPAR_INTC_0_BASEADDR,
                    (1 << XPAR_INTC_0_TMRCTR_0_VEC_ID));

    /* habilita la interrupcio del ethernet MAC */
    XIntc_EnableIntr(XPAR_INTC_0_BASEADDR,
                    XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK);

    /* activa les interrupcions del timer i ethernet MAC */
    XIntc_Enable(&intc, XPAR_INTC_0_TMRCTR_0_VEC_ID);
    XIntc_Enable(&intc, XPAR_INTC_0_EMACLITE_0_VEC_ID);
}
```

Codi 5. Funció d'inicialització del controlador d'interrupcions

Inicialització de LwIP

La funció que inicialitza la llibreria LwIP es pot consultar al Codi 6 a continuació.

```
void inicialitza_lwip(void)
{
    /* inicialitzacio interna de la lwip */
    lwip_init();

    /* especifica les adreces IP a utilitzar manualment, no fa servir DHCP */
    IP4_ADDR(&direccio_ip, 192, 168, 1, 200);
    IP4_ADDR(&mascara_xarxa, 255, 255, 255, 0);
    IP4_ADDR(&gateway, 192, 168, 1, 1);
    imprimeix_configuracio_ip(&direccio_ip, &mascara_xarxa, &gateway);

    /* afegeix la interficie de xarxa a la llista de les disponibles de la
    llibreria */
    xemac_add(&interficie_xarxa, &direccio_ip, &mascara_xarxa, &gateway,
            direccio_mac, XPAR_EMACLITE_0_BASEADDR);

    /* configura la interficie de xarxa per defecte i la posa en marxa */
    netif_set_default(&interficie_xarxa);
    netif_set_up(&interficie_xarxa);
}
```

Codi 6. Funció d'inicialització la llibreria LwIP

Arrencant l'aplicació

La funció que arrenca l'aplicació per poder respondre a peticions TCP es pot consultar al Codi 7 a continuació.

```
void arrenca_app(void)
{
    struct tcp_pcb *pcb;
    err_t err;

    /* crea i reserva memoria per una estructura protocol control block (PCB)
    nova */
    pcb = tcp_new();
    if (!pcb)
    {
        xil_printf("Error creant PCB, falta memoria\r\n");
        return;
    }

    /* lliga aquest PCB al port 80 (per respondre a peticions HTTP, per exemple)
    */
    err = tcp_bind(pcb, IP_ADDR_ANY, 80);
    if (err != ERR_OK)
    {
        xil_printf("No he pogut lligar al port 80. err = %d\r\n", err);
        return;
    }

    /* diu a la llibreria que quan cridi a les funcions callback d'aquest servei
    no retorni arguments */
    tcp_arg(pcb, NULL);

    /* comença a escoltar per si hi han connexions */
    pcb = tcp_listen(pcb);
    if (!pcb)
    {
        xil_printf("M'he quedat sense memoria quan volia començar a
        escoltar\r\n");
        return;
    }

    /* especifica la funcio callback per connexions vinents */
    tcp_accept(pcb, callback_connexio_acceptada);
}
```

Codi 7. Funció que arrenca l'aplicació

Callbacks de l'aplicació

Les funcions que són cridades quan s'accepta una connexió nova i quan es reben dades es poden consultar als Codis 8 i 9 a continuació. En aquest cas, aquestes funcions només han de notificar la llibreria que el paquet s'ha rebut.

```
err_t callback_connexio_acceptada(void *arg, struct tcp_pcb *newpcb, err_t err)
{
    /* quan hi ha una connexio per acceptar l'accepta i lliga el callback de
    dades rebudes a aquest paquet */
    tcp_recv(newpcb, callback_paquet_rebut);

    return ERR_OK;
}
```

Codi 8. Funció callback de connexió acceptada

```
err_t callback_paquet_rebut(void *arg, struct tcp_pcb *tpcb, struct pbuf *p,
                             err_t err)
{
    /* tanca la connexio si el transmissor ha enviat el paquet FIN de TCP */
    if (p == NULL) {
        tcp_close(tpcb);
        return ERR_OK;
    }

    /* avisa a la llibreria que el paquet ha estat rebut */
    tcp_recved(tpcb, p->tot_len);

    /* no s'ha de fer res mes amb el paquet, aixi que allibera l'espai de
    memoria que feia servir aquest */
    pbuf_free(p);

    return ERR_OK;
}
```

Codi 9. Funció callback de paquet rebut

Funcions d'utilitat per imprimir adreces IP

Les funcions cridades des de la inicialització de LwIP per imprimir les adreces IP configurades es poden consultar al Codi 10 a continuació.

```
void imprimeix_direccio_ip(char *msg, struct ip_addr *ip)
{
    xil_printf(msg);
    xil_printf("%d.%d.%d.%d\r\n",
               ip4_addr1(ip),
               ip4_addr2(ip),
               ip4_addr3(ip),
               ip4_addr4(ip));
}

void imprimeix_configuracio_ip(struct ip_addr *ip,
                               struct ip_addr *mask,
                               struct ip_addr *gw)
{
    imprimeix_direccio_ip("Direccio IP: ", ip);
    imprimeix_direccio_ip("Mascara   : ", mask);
    imprimeix_direccio_ip("Gateway   : ", gw);
}
```

Codi 10. Funcions d'utilitat per imprimir adreces IP i configuració de la interfície

Tot el codi junt amb comentaris es pot consultar als annexes d'aquest document.

4.2. Pila programada per l'autor

La pila de protocols bàsica per respondre a peticions d'eco programada per l'autor d'aquest treball se centra en ser el més senzilla i ràpida d'executar possible, per necessitar de la menor quantitat de memòria possible i introduir la menor latència entre paquets possible. Això s'aconsegueix gràcies a optimitzacions petites en el codi C utilitzat. Aquesta opció es presenta en el treball perquè ofereix una manera mínima de funcionar en una xarxa amb altres dispositius i programar protocols propis, dissenyats específicament per l'aplicació que s'estigui programant. Aquests protocols idealment, prescindint de llibreries grans, permetrien que una aplicació fos més lleugera i pogués executar-se amb menys recursos. Per tant els avantatges, si aquest mètode demostra ser més ràpid o igual que l'anterior, són evidents.

A l'hora d'implementar aquesta pila bàsica en C sobre MicroBlaze s'han trobat varis problemes. Sense cap ordre concret, alguns d'ells són:

- A la xarxa Ethernet, els paquets es transmeten començant pel byte més significatiu. Això pot confondre molt degut a què en el MicroBlaze les dades es guarden en l'ordre invers. Això vol dir que quan s'ha de fer alguna operació sobre aquestes, s'ha de tindre en compte aquesta inversió.
- A l'hora de contestar a peticions d'eco, i en general a qualsevol paquet que tingui comprovació d'errors, la operació que més temps consumeix és el càlcul de la suma de verificació. Per millorar això existeixen dues opcions: fer aquesta implementació el més òptima que es pugui (sigui utilitzant codi molt optimitzat de C o utilitzant codi d'assemblador), o utilitzar propietats matemàtiques de la suma de verificació. Segons el document [RFC 1071 \(millorar per enllaç bibliogràfic \[x\]\)](#), quan es fan poques modificacions a un paquet (com podria ser el cas de respondre a una petició d'eco, només es canvia un camp), no és estrictament necessari recalcular tota la suma de verificació. Això vol dir que si tenim la suma de verificació original, només cal restar-hi els valors que eliminem del paquet, i sumar-hi els que afegim. Sempre respectant l'ordenació de bytes descrita al punt anterior.
- Fer servir les interrupcions sobre MicroBlaze resulta bastant confús perquè requereix de varis passos diferents, i el fabricant no ofereix molta documentació al respecte.
- A l'hora de crear estructures amb elements de llargàries diferents (per exemple variables de 8 i 32 bits) cal vigilar amb com ordenarà això a l'espai de memòria el compilador del programa. Pot ser que per defecte tingui habilitada la funció d'alinear les dades a la llargària predeterminada per aquell sistema. En aquest cas sí passa, i el compilador alinea les dades a 32 bits. Generalment aquesta solució és desitjada perquè d'aquesta manera el compilador ha de traduir els accessos a memòria en un menor numero d'instruccions d'assemblador, resultant en un codi més òptim. Però per l'aplicació en qüestió és un problema perquè els paquets que es volen tractar fan servir variables de llargàries diferents. Aquest problema es pot solucionar indicant al compilador que respecti l'ordre i mida de les variables que es volen guardar en l'estructura, encara que això resulti en una implementació pitjor dels accessos a memòria. En el llenguatge C, la majoria de compiladors accepten l'atribut d'estructura `__packed__`. Així, totes les estructures que defineixen tipus de paquets s'han declarat amb: `typedef struct __attribute__((__packed__))`; solucionant així aquest problema.
- Sembla ser que, un cop carregat el codi a la memòria RAM de la placa de la FPGA, si es fa un reset des del botó d'usuari la placa deixa de respondre a les trames d'Ethernet. En canvi, arrencant l'aplicació de nou des de l'entorn de desenvolupament no té aquest efecte. No s'ha pogut descobrir perquè succeeix.

A continuació es descriurà el codi per parts, igual que en la secció anterior.

Funció principal

A la funció principal, en aquest cas, només cal inicialitzar els perifèrics (el controlador d'interrupcions i la capa Ethernet MAC). Inicialitzats aquests perifèrics després ja es poden processar les trames d'Ethernet tal com arribin. Així doncs el codi per inicialitzar serà més curt que en la secció anterior. En el Codi 11 a continuació es pot veure la funció principal (main) de l'aplicació.

```
int main(void)
{
    /* neteja la consola UART i imprimeix missatge */
    xil_printf("%c[2J",27);
    xil_printf("----- TFM - Marko Peshevski - versio NO-LwIP -----\\r\\n");

    inicialitza_interrupcions();
    inicialitza_emaclite();

    /* activa les interrupcions a nivell de processador */
    Xil_ExceptionEnable();

    while (1)
    {
        if (sys.paquet_rebut)
        {
            /* neteja el flag */
            sys.paquet_rebut = FALSE;

            /* inverteix les direccions MAC, respon d'on ha vingut el
            paquet */
            memcpy(trama_ethernet->mac_desti,
                trama_ethernet->mac_origen,
                LLARGARIA_MAC);
            memcpy(trama_ethernet->mac_origen,
                direccio_mac,
                LLARGARIA_MAC);

            /* mira si el paquet es ARP. necessita girar els bytes */
            if(INVERTEIX_BYTES_16(trama_ethernet->ethertype) == ARP)
            {
                /* per espai aquest codi es troba més endavant */
            }
            /* mira si es un paquet IPv4. necessita girar els bytes */
            else if(INVERTEIX_BYTES_16(trama_ethernet->ethertype) == IPv4)
            {
                /* per espai aquest codi es troba més endavant */
            }
        }
        if(sys.paquet_enviat)
        {
            /* neteja el flag */
            sys.paquet_enviat = FALSE;
        }
    }
    return 0;
}
```

Codi 11. Funció principal de l'aplicació sense llibreria LwIP

Inicialització de les interrupcions

La funció que inicialitza el perifèric d'interrupcions es pot consultar al Codi 12 a continuació.

```
void inicialitza_interrupcions(void)
{
    /* inicialitza el periferic Intc de MicroBlaze */
    xil_printf("Inicialitzant periferic Intc... ");
    if (XIntc_Initialize(&intc, XPAR_INTC_0_DEVICE_ID) != XST_SUCCESS)
    {
        xil_printf("No s'ha pogut completar!\r\n");
        return;
    }
    xil_printf("OK!\r\n");

    /* arrenca el periferic en mode real (no simulacio) */
    xil_printf("Arrencant periferic Intc... ");
    if (XIntc_Start(&intc, XIN_REAL_MODE) != XST_SUCCESS)
    {
        xil_printf("No s'ha pogut completar!\r\n");
        return;
    }
    xil_printf("OK!\r\n");

    /* especifica la funcio que gestiona les interrupcions del Emaclite */
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                                (Xil_ExceptionHandler) XIntc_InterruptHandler,
                                &intc);

    /* habilita les interrupcions del Emaclite */
    XIntc_EnableIntr(XPAR_MICROBLAZE_0_INTC_BASEADDR,
                    XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK);

    /* activa les interrupcions del Emaclite */
    XIntc_Enable(&intc, XPAR_INTC_0_EMACLITE_0_VEC_ID);

    /* connecta el senyal d'interrupcio del Emaclite amb el periferic Intc */
    xil_printf("Connectant senyal d'interrupcio del Emaclite... ");
    if (XST_SUCCESS != XIntc_Connect(&intc,
                                     XPAR_INTC_0_EMACLITE_0_VEC_ID,
                                     (XInterruptHandler) XEmacLite_InterruptHandler,
                                     (void *) &intc))
    {
        xil_printf("No s'ha pogut completar!\r\n");
    }
    xil_printf("OK!\r\n");
}
```

Codi 12. Funció d'inicialització del controlador d'interrupcions

Inicialització del perifèric Emaclite

La funció que inicialitza el perifèric Emaclite es pot consultar al Codi 13 a continuació.

```
void inicialitza_emaclite(void)
{
    /* inicialitza el Emaclite i el xip PHY de la placa */
    xil_printf("Inicialitzant Emaclite i PHY... ");
    if (XST_SUCCESS != XEmacLite_Initialize(&emaclite,
                                            XPAR_ETHERNET_MAC_DEVICE_ID))
    {
        xil_printf("No s'ha pogut completar!\r\n");
        return;
    }
    xil_printf("OK!\r\n");

    XIntc_RegisterHandler(XPAR_MICROBLAZE_0_INTC_BASEADDR,
                        XPAR_INTC_0_EMACLITE_0_VEC_ID,
                        (XInterruptHandler)XEmacLite_InterruptHandler,
                        &emaclite);

    /* configura la direccio MAC */
    xil_printf("Configurant la següent direccio MAC: ");
    print_mac_address(direccio_mac);
    xil_printf("... ");
    XEmacLite_SetMacAddress(&emaclite, (u8 *) direccio_mac);
    xil_printf("OK!\r\n");

    /* neteja els buffers de recepcio */
    XEmacLite_FlushReceive(&emaclite);

    /* assigna funcions callback i habilita interrupcions */
    xil_printf("Assignant funcions de callback per les interrupcions... ");
    XEmacLite_SetRecvHandler(&emaclite,
                            &emaclite,
                            (XEmacLite_Handler) recv_callback);
    XEmacLite_SetSendHandler(&emaclite,
                            &emaclite,
                            (XEmacLite_Handler) sent_callback);
    xil_printf("Habilitant interrupcions d'Emaclite... ");
    if (XEmacLite_EnableInterrupts(&emaclite) != XST_SUCCESS)
    {
        xil_printf("No s'ha pogut completar!\r\n");
        return;
    }
    xil_printf("OK!\r\n");
}
```

Codi 13. Funció d'inicialització del perifèric Emaclite

Callbacks de l'aplicació

Les funcions que són cridades quan s'ha rebut o s'ha enviat un paquet es poden consultar als Codis 14 i 15 a continuació.

```
void callback_rebut(XEmacLite * callbackReference)
{
    /* neteja la interrupcio del sistema */
    XIntc_AckIntr(XPAR_MICROBLAZE_0_INTC_BASEADDR,
                  XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK);

    /* llegeix la llargaria de les dades rebudes */
    sys.llargaria_paquet_rebut = XEmacLite_Recv(&emac_lite, &buffer[0]);

    /* posa un flag per notificar al bucle principal */
    sys.paquet_rebut = TRUE;
}
```

Codi 14. Funció callback de paquet rebut

```
void callback_enviat(XEmacLite * callbackReference)
{
    /* neteja la interrupcio del sistema */
    XIntc_AckIntr(XPAR_MICROBLAZE_0_INTC_BASEADDR,
                  XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK);

    /* posa un flag per notificar al bucle principal */
    sys.paquet_enviat = TRUE;
}
```

Codi 15. Funció callback de paquet enviat

Tipus de paquets

Les definicions de tipus d'estructures utilitzades es troben als Codis 16 (trama Ethernet), 17 (paquet IPv4), 18 (ARP), i 19 (ICMP) a continuació. Aquestes definicions són d'especial importància per la simplificació del tractament del paquet que donen.

```
typedef struct __attribute__((__packed__))
{
    u8 mac_desti[LLARGARIA_MAC];
    u8 mac_origen[LLARGARIA_MAC];
    u16 ethertype;
    u8 * dades;
} trama_ethernet_t;
```

Codi 16. Definició de tipus d'estructura per trames Ethernet

```
typedef struct __attribute__((__packed__))
{
    u8 versio_llargaria_header;
    u8 tipus_de_servei;
    u16 llargaria_total;
    u16 identificacio;
    u16 flags_fragments;
    u8 temps_de_vida;
    u8 protocol;
    u16 suma_verificacio;
    u32 ip_origen;
    u32 ip_desti;
    u8 * dades;
} paquet_ip_t;
```

Codi 17. Definició de tipus d'estructura per paquets IPv4

```
typedef struct __attribute__((__packed__))
{
    u16 tipus_de_medi;
    u16 identificacio;
    u8 llargaria_direccio_fisica;
    u8 llargaria_direccio_logica;
    u16 operacio;
    u8 mac_origen[LLARGARIA_MAC];
    u32 ip_origen;
    u8 mac_desti[LLARGARIA_MAC];
    u32 ip_desti;
} paquet_arp_t;
```

Codi 18. Definició de tipus d'estructura per paquets ARP

```
typedef struct __attribute__((__packed__))
{
    u8 tipus_de_missatge;
    u8 codi;
    u16 suma_verificacio;
    u32 dades_header;
    u8 * dades;
} paquet_icmp_t;
```

Codi 19. Definició de tipus d'estructura per paquets ICMP

Variables i funcions d'ajuda

Les variables globals que es fan servir al programa es poden veure al Codi 20 a continuació. Les funcions d'ajuda per imprimir adreces IP i MAC es poden trobar al Codi 21.

```
u8 direccio_mac[LLARGARIA_MAC] = {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};
u32 direccio_ip = (192) | (168<<8) | (1<<16) | (200<<24);
XEmacLite emaclite;
XIntc intc;
static u8 buffer[2048] = {'\0'};
static volatile variables_sistema sys;
static trama_ethernet_t * trama_ethernet = (trama_ethernet_t *) &buffer[0];
```

Codi 20. Variables globals utilitzades al programa

```
void imprimeix_direccio_mac(u8 * addr)
{
    xil_printf("%02x:%02x:%02x:%02x:%02x:%02x",
               addr[0],
               addr[1],
               addr[2],
               addr[3],
               addr[4],
               addr[5]);
}
```

Codi 21. Funció d'utilitat per imprimir adreces MAC

Trossos de codi del bucle principal

El codi que tracta els paquets ARP i ICMP quan arriben (trossos que faltaven al Codi 11) es pot trobar a continuació, en els codis 22 i 23.

```
/* mira si el paquet es ARP. necessita girar els bytes */
if (INVERTEIX_BYTES_16(trama_ethernet->ethertype) == ARP)
{
    /* es un cast que interpreta les dades de memoria per facilitar */
    paquet_arp_t * paquet_arp = (paquet_arp_t *) &trama_ethernet->dades;
    /* mira si el paquet anava dirigit per la nostra IP */
    if(paquet_arp->ip_desti == direccio_ip)
    {
        if(paquet_arp->operacio == ARP_REQUEST_LINUX)
        {
            paquet_arp->operacio = ARP_REPLY_LINUX;
        }
        else if(paquet_arp->operacio == ARP_REQUEST_WINDOWS)
        {
            paquet_arp->operacio = ARP_REPLY_WINDOWS;
        }
        /* inverteix adreces IP i MAC del paquet */
        paquet_arp->ip_desti = paquet_arp->ip_origen;
        paquet_arp->ip_origen = direccio_ip;
        memcpy(paquet_arp->mac_desti, paquet_arp->mac_origen,
            LLARGARIA_MAC);
        memcpy(paquet_arp->mac_origen, direccio_mac,
            LLARGARIA_MAC);
        /* envia la resposta */
        XEmaclite_Send(&emaclite, buffer,
            sys.llargaria_paquet_rebut - LLARGARIA_FCS);
    }
}
```

Codi 22. Codi que tracta els paquets ARP

```
/* mira si es un paquet IPv4. necessita girar els bytes */
else if (INVERTEIX_BYTES_16(trama_ethernet->ethertype) == IPv4)
{
    /* es un cast que interpreta les dades de memoria per facilitar */
    paquet_ip_t * paquet_ip = (paquet_ip_t *) &trama_ethernet->dades;
    /* mira si es un paquet de tipus ICMP */
    if(paquet_ip->protocol == ICMP)
    {
        /* es un cast que interpreta les dades de memoria per facilitar */
        paquet_icmp_t * paquet_icmp = (paquet_icmp_t *) &paquet_ip->dades;
        /* mira si es una petició d'eco */
        if(paquet_icmp->tipus_de_missatge == ECHO_REQUEST)
        {
            /* canvia la suma de verificacio */
            paquet_icmp->suma_verificacio += ECHO_REQUEST;
            /* -ECHO_REPLY, pero no cal porque es 0 */
            /* modifica el tipus de missatge */
            paquet_icmp->tipus_de_missatge = ECHO_REPLY;
            /* inverteix les adreces del paquet IPv4 */
            paquet_ip->ip_desti = paquet_ip->ip_origen;
            paquet_ip->ip_origen = direccio_ip;
            /* envia la resposta */
            XEmaclite_Send(&emaclite, buffer,
                sys.llargaria_paquet_rebut - LLARGARIA_FCS);
        }
    }
}
```

Codi 23. Codi que tracta els paquets ICMP de petició d'eco

5. RESULTATS EXPERIMENTALS

5.1. Mètode d'estudi

Per fer les proves experimentals per poder posar a prova les implementacions de pila de protocol al respondre a una petició d'eco s'ha decidit fer-ho des d'una màquina Linux. La principal raó és que la utilitat ping de Linux dona molta més informació que la de Windows. Per tots els temps per sota d'un milisegon la utilitat de Windows simplement diu <1ms, mentre que la de Linux dona la informació completa.

Per poder automatitzar les proves el màxim possible el que s'ha fet és un petit script de bash (terminal de Linux) que executa els pings i guarda el resultat en un arxiu. En aquest script l'únic important que hi ha és la línia que executa el ping. Es pot veure aquest script al Codi 24. Els arguments amb què s'executa la utilitat ping són:

- `sudo`, permisos de superusuari requerits per poder tenir un interval per sota de 200 ms.
- `192.168.1.200`, adreça IP de la placa utilitzada, la mateixa en tots els casos.
- `-c 100`, quantitat de paquets que es volen intercanviar, en aquest cas 100.
- `-s $i`, quantitat de bytes que contindrà cada paquet d'eco, anirà incrementant segons el valor del comptador del bucle. És a dir, des de 20 fins a 1000 bytes, en salts de 10, tornant un total de 99 valors per prova.
- `-q`, la opció `--quiet`, que només retorna el resultat estadístic de la prova, sense retornar els resultats individuals de cada eco
- `-i 0.01`, l'interval en segons entre cada petició d'eco des de l'ordinador. En aquest cas són 10 ms, perquè no s'espera cap temps d'anada i tornada del paquet d'eco més llarg que aquest temps.
- `-w 0.01`, la quantitat de temps màxima durant la qual l'ordinador espera resposta. Passat aquest temps dona la petició com a perduda i compta un paquet perdut.

```
#!/bin/bash
for i in `seq 20 10 1000`;
do
    echo $i
    sudo ping 192.168.1.200 -c 100 -s $i -q -i 0.01 -w 0.01 >> arxiu.txt
    sleep 0.5
done
```

Codi 24. Script de bash utilitzat per realitzar totes les proves

Aleshores, un cop es té l'arxiu de text corresponent a la prova, d'aquesta es poden extreure les dades i utilitzar-les en un full de càlcul per fer més càlculs o gràfics per poder comparar. Un resultat d'una de les proves fetes podria assemblar-se a les següents línies a la terminal de Linux:

```
PING 192.168.1.200 (192.168.1.200) 420(448) bytes of data.
```

```
--- 192.168.1.200 ping statistics ---
```

```
100 packets transmitted, 100 received, 0% packet loss, time 989ms
```

```
rtt min/avg/max/mdev = 1.556/1.597/1.622/0.057 ms
```

Codi 25. Possible resposta d'una prova d'eco

En aquestes línies, es poden identificar varis paràmetres d'utilitat per les proves que interessin per aquest treball:

- 420, número de bytes continguts com a dades en el paquet ICMP.
- 100 packets transmitted, el nombre de paquets transmesos.
- 100 packets received, el nombre de paquets rebuts.

- 0% packet loss, el percentatge de paquets perduts (sense resposta).
- rtt min/avg/max/mdev, les dades estadístiques de temps de la prova. rtt es refereix a temps d'anada i tornada, des que el paquet surt del PC fins que torna en forma de resposta del receptor. Els altres valors són mínim, mitjana, màxim i desviació estàndard de tots els valors, respectivament.

D'aquestes dades el que es farà és extreure els mínim, mitjana, màxim i desviació estàndard dels resultats. En totes les proves realitzades no s'ha vist cap en què hagués pèrdua de paquets, així que aquesta informació no és útil. Aquest fet és d'esperar perquè es tracta d'una xarxa Ethernet domèstica on no hi ha congestió ni una densitat elevada de tràfic.

Un cop establert el mètode per fer proves, el que cal és definir les proves que es volen fer. En aquest treball s'han decidit fer les següents proves:

- Prova amb ambdues implementacions, fent servir un cable Cat3.
- Prova amb ambdues implementacions, fent servir un cable Cat6.
- Prova amb la implementació de l'autor, utilitzant mètodes diferents per calcular la suma de verificació, per intentar veure l'impacte que té aquesta.
- Prova amb la implementació de l'autor, utilitzant un hardware que executa codi des de la memòria RAM LPDDR de la placa i un altre que executa codi des de la memòria interna de MicroBlaze, directament sobre la FPGA. Aquesta prova no es pot dur a terme amb la implementació LwIP degut a què el codi resultant és més gran que el màxim de memòria de programa que hi cap a la FPGA (32 kilobytes).

5.2. Resultats de les proves

Proves amb cable Cat3

Com s'ha mencionat anteriorment, una de les proves que s'ha realitzat és la d'executar el script del Codi 25, amb la implementació de protocols corresponent a la FPGA, utilitzant un cable Cat3 (amb només 2 parells diferencials). Els resultats d'aquestes proves es poden veure a les Taules 1 i 2 i la Figura 27. En les taules només s'han extret valors cada 100 bytes per economia d'espai, però a la figura es troben representat tots els valors, des de 20 bytes fins a 1000.

Taula 1. Resultats de la prova amb cable Cat3 per la llibreria LwIP

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.988	1.015	1.051	0.025
200	1.136	1.190	1.218	0.035
300	1.349	1.372	1.397	0.033
400	1.534	1.559	1.585	0.033
500	1.718	1.742	1.770	0.045
600	1.912	1.929	2.005	0.055
700	2.094	2.117	2.143	0.023
800	2.285	2.315	2.355	0.053
900	2.470	2.497	2.534	0.049
1000	2.660	2.686	2.718	0.044

Taula 2. Resultats de la prova amb cable Cat3 per la implementació de l'autor

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.628	0.656	0.688	0.029
200	0.748	0.776	0.808	0.031
300	0.871	0.899	0.932	0.025
400	0.989	1.005	1.040	0.019
500	1.122	1.126	1.144	0.034
600	1.240	1.261	1.295	0.045
700	1.366	1.392	1.416	0.048
800	1.498	1.518	1.539	0.044
900	1.614	1.650	1.674	0.043
1000	1.749	1.777	1.804	0.026

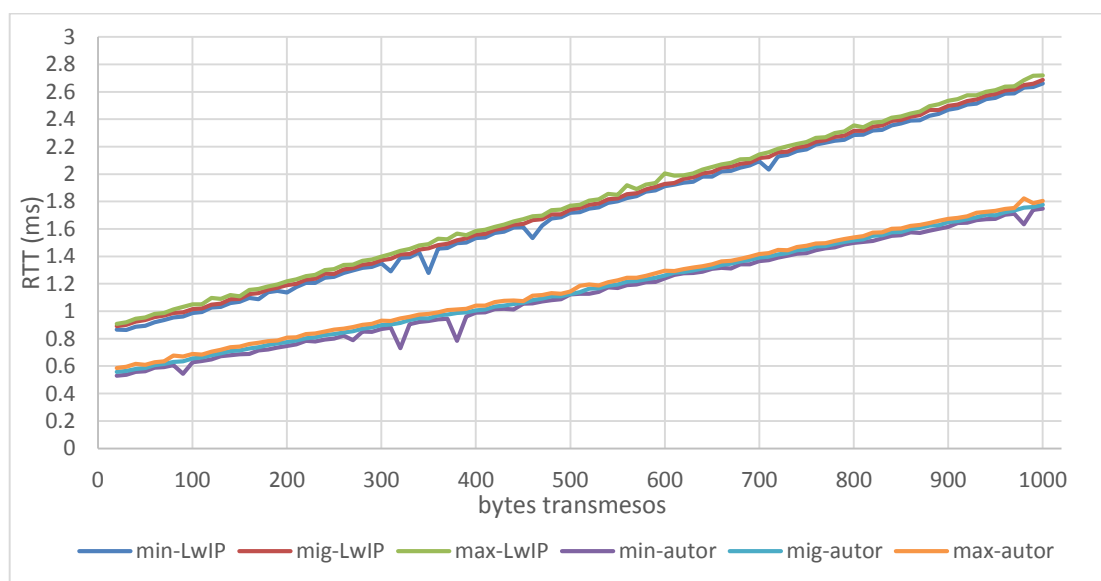


Figura 27. Proves d'eco amb les diferents implementacions sobre un cable Cat3

Proves amb cable Cat6

La segona prova que s'ha realitzat és similar a l'anterior, variant el cable utilitzat. Els resultats d'aquesta prova es poden veure a les Taules 3 i 4 i la Figura 28. Segons els valors de les taules i la figura, no sembla que hagi una diferència apreciable entre utilitzar cable de Cat3 i cable de Cat6, sempre i quan la comunicació sigui a 100 Mbit/s.

Taula 3. Resultats de la prova amb cable Cat6 per la llibreria LwIP

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.870	1.017	1.046	0.022
200	1.165	1.196	1.220	0.028
300	1.356	1.375	1.396	0.027
400	1.535	1.564	1.590	0.054
500	1.717	1.743	1.775	0.044
600	1.899	1.931	1.968	0.062
700	2.095	2.124	2.153	0.060
800	2.291	2.323	2.346	0.011
900	2.465	2.502	2.528	0.062
1000	2.660	2.683	2.709	0.010

Taula 4. Resultats de la prova amb cable Cat6 per la implementació de l'autor

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.644	0.667	0.695	0.025
200	0.754	0.785	0.809	0.040
300	0.868	0.903	0.930	0.031
400	0.984	1.007	1.044	0.027
500	1.115	1.136	1.164	0.033
600	1.237	1.264	1.284	0.012
700	1.368	1.400	1.423	0.044
800	1.492	1.520	1.544	0.034
900	1.622	1.647	1.683	0.015
1000	1.756	1.780	1.847	0.045

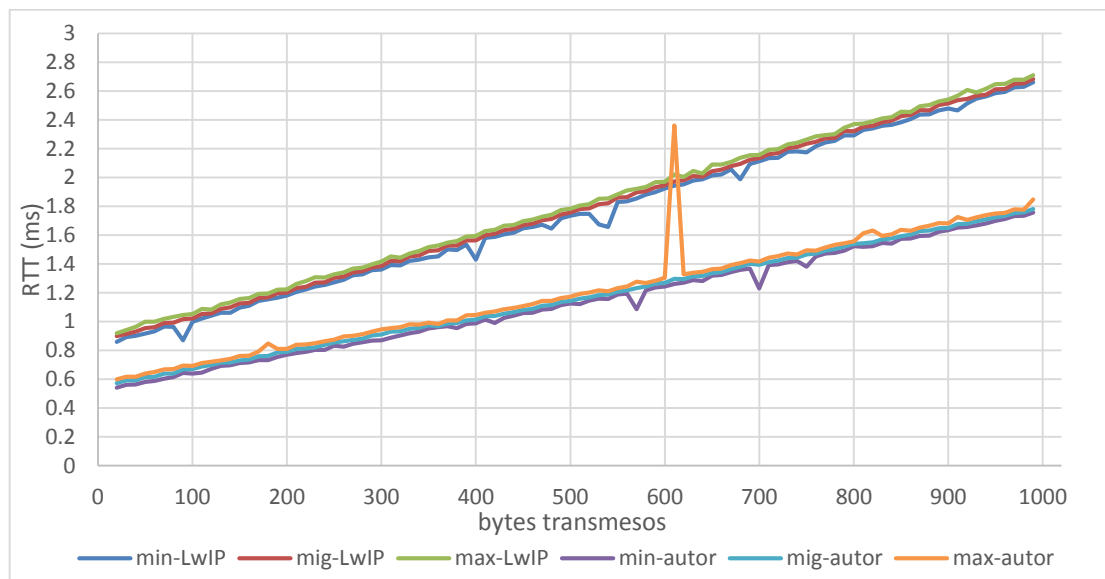


Figura 28. Proves d'eco amb les diferents implementacions sobre un cable Cat6

Proves amb diferents sumes de verificació

La tercera prova que s'ha realitzat varia de les anteriors. Se segueix fent servir el cable Cat6, degut a què no s'han observat diferències entre Cat3 i Cat6. En aquest cas, el que canvia és la implementació de la suma de verificació del paquet ICMP de petició d'eco. La implementació *ràpida* és la que es podia trobar al Codi 23. En aquest cas, la implementació *lenta* consistirà en recalcular la suma de verificació de tot el paquet fent servir l'algoritme sense cap optimització. El Codi 26 mostra la funció que calcula la suma de verificació. Els resultats d'aquesta prova es poden veure a les Taules 4, 5 i 6 i la Figura 29.

Taula 5. Resultats de la prova amb la suma de verificació *ràpida*

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.630	0.659	0.691	0.026
200	0.748	0.775	0.807	0.033
300	0.888	0.909	0.931	0.042
400	0.987	1.011	1.037	0.013
500	1.122	1.128	1.163	0.041
600	1.230	1.262	1.287	0.021
700	1.372	1.387	1.411	0.021
800	1.495	1.520	1.543	0.032
900	1.618	1.644	1.666	0.050
1000	1.754	1.772	1.805	0.024

Taula 6. Resultats de la prova amb la suma de verificació *lenta*

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.865	0.898	0.917	0.013
200	1.186	1.208	1.241	0.048
300	1.509	1.537	1.561	0.012
400	1.726	1.853	1.890	0.049
500	2.163	2.187	2.221	0.031
600	2.497	2.522	2.549	0.049
700	2.748	2.851	2.874	0.058
800	3.156	3.178	3.207	0.043
900	3.472	3.512	3.547	0.075
1000	3.824	3.851	3.871	0.068

Taula 7. Resultats de la prova amb la implementació LwIP

Bytes transmesos	RTT mínim (ms)	RTT mitjà (ms)	RTT màxim (ms)	RTT desv. est. (ms)
100	0.990	1.017	1.046	0.021
200	1.161	1.193	1.233	0.031
300	1.350	1.381	1.405	0.013
400	1.438	1.557	1.584	0.039
500	1.719	1.748	1.774	0.043
600	1.900	1.932	1.956	0.031
700	2.044	2.114	2.137	0.056
800	2.282	2.307	2.341	0.046
900	2.477	2.503	2.527	0.055
1000	2.533	2.684	2.851	0.065

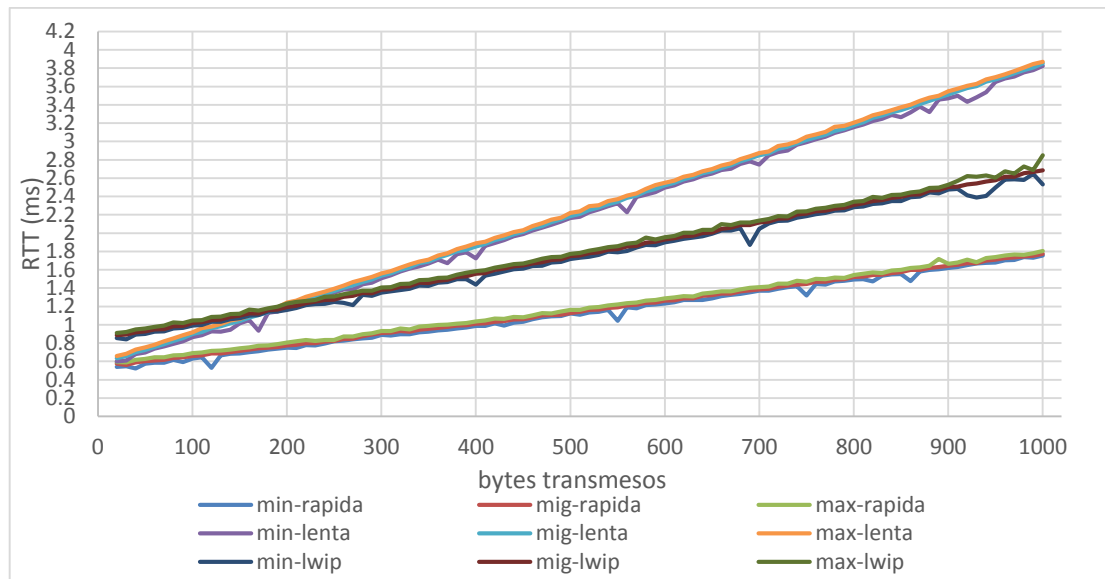


Figura 29. Proves d'eco amb les diferents sumes de verificació

Com es pot veure a la Figura 29, la diferència entre els diferents mètodes és clara. Sembla ser que per un nombre de bytes petit les 3 implementacions s'assemblen, però a mesura que creix el nombre de bytes la diferència és cada cop més gran. També es pot apreciar que el grup de línies de la implementació LwIP (línies fosques del centre), i les línies de la implementació de l'autor *ràpida* (línies més baixes) són relativament paral·leles. Això sembla indicar que hi ha algun factor que suma una certa espera de temps a la llibreria LwIP que a la implementació *ràpida* de l'autor no hi és. La implementació *lenta* és, evidentment, cada cop més lenta a mesura que creix el nombre de bytes, perquè s'ha de recórrer un nombre de posicions de memòria més gran.

```
void calcula_suma_verificacio(paquet_icmp_t * frame,
                             u8 * direccio_inici,
                             u8 * direccio_final)
{
    u8 * i = 0;
    u32 suma = 0;
    u16 valor_calculat = 0;

    for (i = direccio_inici; i < direccio_final; i=i+2)
    {
        suma += (*i<<8) | (*(i+1));
    }

    valor_calculat = ~(((suma & 0xFF0000)>>16) + (suma & 0x00FFFF));
    frame->header_checksum = INVERTEIX_BYTES_16(valor_calculat);
    return;
}
```

Codi 26. Funció que implementa l'algoritme de càlcul de la suma de verificació *lenta*

5.3. Comparació entre les piles

6. CONCLUSIONS I TREBALL FUTUR

6.1. Conclusió

6.2. Treball futur

7. BIBLIOGRAFIA I ANNEXES

AGRAÏMENTS