Universidad de la República
Facultad de Ingeniería

Octubre 2008

PgVirtex 2

Martín Ignacio Dubourdieu Hourcade
Alejandro Romio Ravazzani
Rodrigo Martín Taborda Bonilla

Tutor: Juan Pablo Oliver

Contenido

1. Agradecimientos **6**

2. Resumen **7**

3. Introducción **8**

4. La placa PG-VIRTEX **10**

I Ethernet sin Leon **13**

5. Protocolo Ethernet **14**

6. Estudio de los chips **18**
6.1. Chip LXT974 . . . 18
6.2. Pruebas básicas de funcionamiento . . . 21

7. Diseño **22**
7.1. Requerimientos . . . 22
7.2. Elección del core MAC . . . 23
7.2.1. GRETH 10/100 Mbit Ethernet MAC . . . 23
7.2.2. Ethernet IP Core de Opencores . . . 25
7.3. Wishbone . . . 26
7.4. Puertos de Entrada/Salida del Ethernet IP Core de Opencores . . . 33
7.5. Registros del core MAC . . . 37
7.5.1. Buffer descriptors . . . 37
7.5.2. Registro Moder . . . 39
7.5.3. Registros de interrupción . . . 41
7.6. Testbench del IP Core de Opencores . . . 42
7.7. RAM utilizada . . . 43
7.8. Diseño del bloque MAC_eth_controller . . . 43
7.8.1. Ciclos de escritura/lectura a la RAM . . . 44
7.8.2. Ciclo de escritura/lectura al core MAC . . . 46
7.9. Bloque principal . . . 47
7.9.1. Conexionado del Bloque Principal . . . 48
7.9.2. Señales de reset y reloj . . . 51

8. Simulación **51**

9. Síntesis y pruebas **55**

II Ethernet con Leon y Linux **58**

10. Processador LEON2 **59**
10.1. Arquitectura . . . 59
10.2. Configuración . . . 64

11. Antecedentes **64**

12. Snapgear Linux para LEON **65**

13. Elección del tipo de procesador **66**

14. Configuración del LEON **67**

15. Compilación y simulación de Linux **68**
15.1. Aspectos generales del sistema 69

16. Implementación del sistema **73**
16.1. Síntesis del LEON sin Ethernet, prueba con grmon. 74
16.2. Síntesis del LEON con Ethernet, verificación con wireshark 76

III PCI **77**

17. Estudio de alternativas **78**
17.1. Manejo del puerto USB 78
17.2. PCI con LEON2 78
17.3. PCI - Ethernet 79

18. Estudio del Chip PCI **79**

19. Utilización del CPLD **81**

20. Software necesario **81**
20.1. PLXmon 81
20.2. Windriver 82

21. Problemas Encontrados **83**
21.1. Problemas de Software 83
21.1.1. Problemas con PLXmon 83
21.1.2. Problemas con el Windriver 83
21.2. Problemas de hardware 84

22. Modificación del diseño inicial **85**
22.1. Introducción 85
22.2. Compatibilidad de protocolos 86
22.3. Modificación del bloque principal 86
22.3.1. Interfaz hacia el bus local y CPLD 87
22.3.2. Proceso de escritura de datos 88
22.4. Modificación de core de programación del FPGA 89
22.5. Árbitro del bus 89
22.6. Reloj utilizado 90

23. Pruebas **91**

IV Conclusiones generales **93**

Referencias **95**

V Anexos **96**

24. Anexo 1: Chip LAN91C111 **97**

25. Anexo 2: Protocolo AMBA **100**
25.1. Advanced High-Performance Bus 100
25.2. Advanced System Bus 102
25.3. Advanced Peripheral Bus 103
25.4. Elección del bus del sistema 104

26. Anexo 3: Plan inicial de proyecto **106**
26.1. Descripción de los entregables intermedios. 106
26.2. Descripción del proyecto 106
26.3. Objetivo general del proyecto 107
26.4. Actores, supuestos y restricciones 107
26.5. Especificación funcional del proyecto 108
26.6. Definición detallada del alcance 108
26.7. Análisis de riesgos 109

27. Anexo 4: Diagramas de Gantt y estimación de horas: Plan vs. Real **111**

Lista de Figuras

1. Diagrama de bloques de la placa PGVIRTEX[8] . . . 10
2. Placa PG-VIRTEX[8] . . . 11
3. Conexión de Interfaz MII[9] . . . 20
4. Arquitectura del Greth Ethernet Core[6] . . . 24
5. Arquitectura del Ethernet IP Core[10] . . . 25
6. Conexión Master-Salve Wishbone[5] . . . 29
7. Ciclo simple lectura/escritura Wishbone[5] . . . 31
8. Lectura Wishbone en bloque[5] . . . 32
9. Escritura Wishbone en bloque[5] . . . 32
10. Escritura en RAM . . . 45
11. Escritura de un Buffer Descriptor . . . 47
12. Arquitectura del Bloque Principal . . . 48
13. Arquitectura del testbench . . . 52
14. Lectura de la RAM desde el core MAC (acceso DMA) . . . 53
15. Envío de paquete desde el core MAC al PHY (inicio) . . . 55
16. Envío de paquete desde el core MAC al PHY (fin) . . . 55
17. Arquitectura del microprocesador LEON2 . . . 60
18. Espacio de direcciones AHB[7] . . . 63
19. Mapeo de direcciones del controlador de memoria[7] . . . 63
20. Archivo de configuración de LEON2 . . . 69
21. Configuración de Linux . . . 70
22. Elección del procesador . . . 70
23. Aspectos generales del sistema. . . . 71
24. Configuración del Kernel . . . 71
25. Configuraciones de usuario . . . 71
26. Elección del comando ifconfig . . . 72
27. Prueba de Sistema LEON - Linux con Ethernet . . . 73
28. Comunicación con el sistema a través de Grmon . . . 75
29. Ejecución del comando ping desde Snapgear . . . 76
30. Arquitectura del chip PCI 9054[11] . . . 80
31. Conexión PCI 9054 - CPLD - FPGA . . . 85
32. Ciclo de escritura de cuatro bytes del bus local[11] . . . 88
33. Arquitectura del chip LAN91C111[12] . . . 97
34. Diagrama de Gantt: plan (1ª parte) . . . 111
35. Diagrama de Gantt: real (1ª parte) . . . 111
36. Diagrama de Gantt: plan (2ª parte) . . . 112
37. Diagrama de Gantt: real (2ª parte) . . . 113
38. Diagrama de Gantt: plan (3ª parte) . . . 114
39. Diagrama de Gantt: real (3ª parte) . . . 114

Lista de Tablas

1. Trama IEEE 802.3[2] 16
2. Paquete UDP 17
3. Señales Wishbone del core MAC[10] 34
4. Señales del core MAC hacia el PHY[10] 35
5. Detalle del buffer descriptor de transmisión[10] 38
6. Detalle del buffer descriptor de recepción[10] 39
7. Señales de MAC_eth_controller 44
8. Señales de entrada/salida del bloque principal 50
9. Señales de interfaz bloque principal - bus local/CPLD 87

1. Agradecimientos

- A Leonardo Etcheverry por darnos algunos consejos muy útiles acerca del LEON.

- A Alejo "Gringo" Rubinovicz por brindarnos material que resultó vital en el desarrollo del proyecto.

- A Alfredo "Chispa" Sánchez del grupo Linux de Paysandú, por sus consejos acerca de la compilación de un kernel de Linux.

- A Melissa Bisio, por su ayuda con el diseño y confección del poster del proyecto.

- A Juan Pablo Oliver, tutor de nuestro proyecto, por sus consejos útiles.

2. Resumen

Nuestro proyecto nace con la finalidad de poner en funcionamiento algunos de los periféricos existentes en la placa PgVirtex. Esta placa fue desarrollada en el Instituto de Ingeniería Eléctrica por un grupo de proyecto de fin de carrera en el año 2005. Cuenta con un FPGA de la familia VIRTEX-E de Xilinx y con algunos periféricos e interfaces como puerto serie, IRDA, USB, puertos Ethernet y PCI (Peripheral Component Interconnect). Las pruebas hechas para corroborar el funcionamiento de la placa fueron muy básicas y si bien se utilizó en un proyecto de fin de carrera anterior al nuestro, alguno de los puertos de ésta nunca habían sido utilizados. Este es el caso de los puertos Ethernet, USB y bus PCI.

En la primera parte del proyecto se hicieron funcionar 4 bocas Ethernet de la placa mediante un core implementado en el FPGA que se encarga de enviar y recibir paquetes.

Luego se decidió hacer funcionar los mismos puertos Ethernet, pero esta vez implementando un SOC[1] compuesto por un procesador y corriendo un sistema operativo Linux sobre el mismo que se encarga del envío y recepción de paquetes.

En la última etapa, se logró diseñar un core que es capaz de recbir paquetes a través del puerto PCI de la placa, y utilizando el core diseñado en el primera parte del proyecto, enviarlos a través de los puertos Ethernet.

Esta documentación está dividida en tres partes principales además de las conclusiones y los anexos. En cada una de las partes se pretende explicar lo realizado en cada una de las etapas del proyecto, lo cual incluye estudios de distintos chips, protocolos y diseño de cores en VHDL. Además el estudio y en algunos casos modificación de diseños no realizados por nosotros en VHDL y Verilog para adaptarlos a nuestros requerimientos.

[1] System On a Chip

3. Introducción

Un FPGA (field-programmable gate array) es un dispositivo semiconductor que contiene lógica programable llamada "bloques lógicos" e interconexiones programables[1]. Los bloques lógicos pueden ser programados para la ejecución de funciones básicas como AND ó XOR, ó también para ejecutar funciones combinatorias mas complejas como decodificadores o funciones matemáticas. En la mayoría de los FPGAs los bloques lógicos incluyen memorias simples como los flip-flops o bloques más completos de memoria.

Para lograr el funcionamiento de cuatro de las bocas Ethernet de la placa PgVirtex, se utilizó el FPGA de la misma en el cual se grabó un core que implementa la subcapa de acceso al medio para el envío y recepción de paquetes. El core además incluye una RAM donde se guardan los paquetes a ser enviados además de los que se reciben, y otro bloque que se encarga del control del sistema. Los detalles de esta implementación se profundizan más adelante en esta documentación.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un SOC. En nuestro caso, para la segunda parte del proyecto se implementó un SOC que incluye un procesador LEON2 de Gaisler Research y que se comunica con los puertos Ethernet a través de un sistema operativo Linux SnapGear[2]. La comunicación de la placa con una PC para cargar y ejecutar comandos en el sistema operativo se hizo mediante el puerto serie de la misma y utilizando dos UARTS. Cabe recalcar que la comunicación serie para el manejo del Snapgear fue necesaria ya que la placa no cuenta con una salida VGA o similar.

En la parte final del proyecto se rediseñó el core de la primera etapa para lograr enviar paquetes desde una PC a través del puerto PCI de la PgVirtex para luego ser enviados a través de los puertos Ethernet de la misma. Como antecedente teníamos que el grupo que desarrolló la placa había logrado un diseño capaz de programar el FPGA. Este diseño consiste en un core que se graba en un CPLD existente en la placa, el cual esta debidamente comunicado con el FPGA. En nuestro diseño además se debió modificar este core para que fuera posible comunicarnos con el

[2]La versión de Snapgear utilizada es dispuesta por Gaisler Research, especificamente para correr sobre un procesador LEON

PCI desde el FPGA. Esta comunicación, como se detalla oportunamente, no se hace directamente sobre el puerto sino a través de la interfaz de un chip que se encuentra en la placa, el cual está diseñado para el manejo del puerto PCI.

4. La placa PG-VIRTEX

En esta sección se explicará de manera muy breve el diseño y funcionamiento de la placa PgVirtex, para una mejor comprensión de la documentación de nuestro proyecto. Si se desea profundizar en el conocimiento de esta placa se puede consultar la documentación del proyecto que estuvo a cargo de su diseño y desarrollo[8].

Esta placa fue diseñada en el IIE por un grupo de proyecto de fin de carrera en el año 2005. Los realizadores de este proyecto fueron Silvia Gómez, Jimena Saporiti y Agustín Villavedra. El objetivo fue diseñar y fabricar una placa basada en lógica programable que pueda ser utilizada como plataforma de desarrollo y evaluación de circuitos.

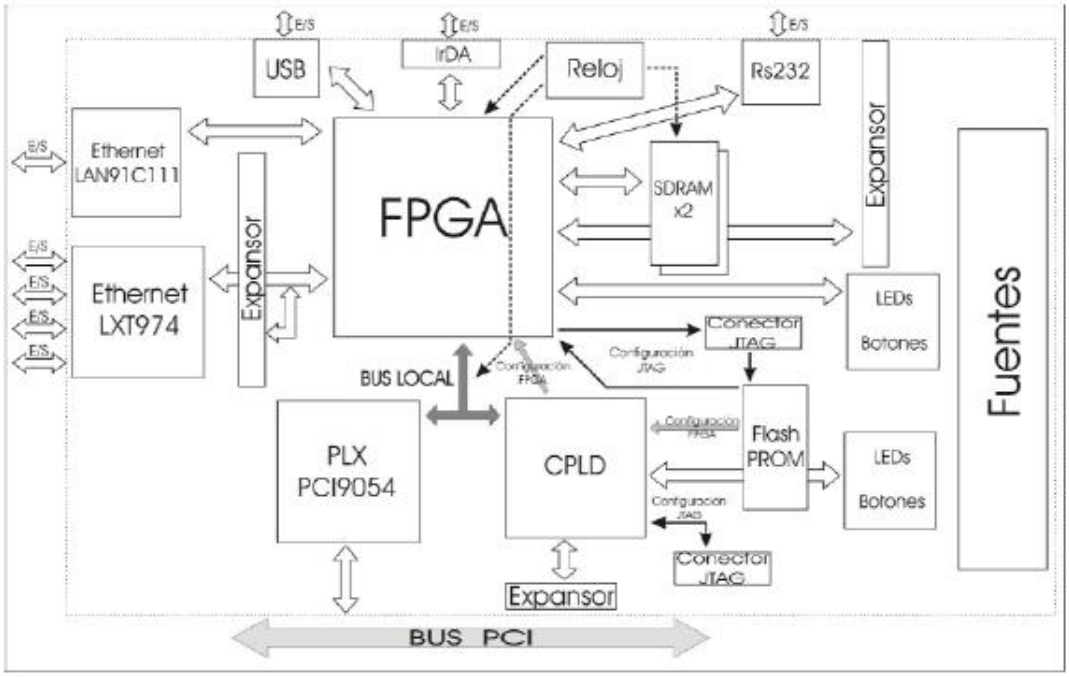


Figura 1: Diagrama de bloques de la placa PGVIRTEX[8]

La placa cuenta con un FPGA de la familia Virtex-E de Xilinx. Este FPGA tiene poco más de 330.000 compuertas lógicas y 404 puertos de entrada/salida, distribuidos en un encapsulado BGA (Ball Grid Array)

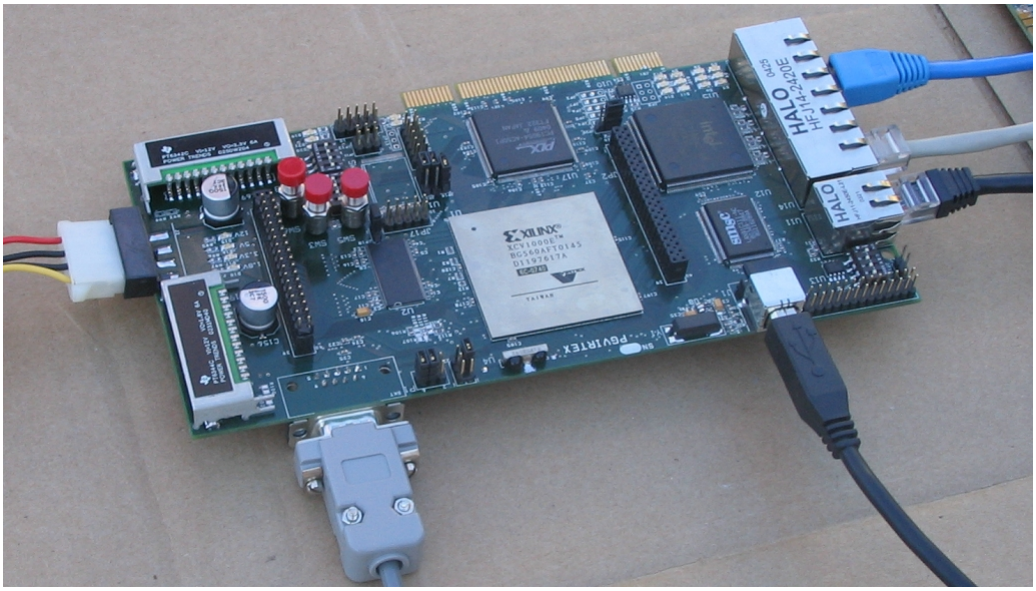


Figura 2: Placa PG-VIRTEX[8]

de 560 pines.

Una de las varias formas de comunicación con el mundo exterior con la que cuenta la placa es a través del bus PCI. El encargado de traducir los comandos, datos y direcciones del bus PCI al bus local de la placa es un integrado de la empresa PLX: PCI 9054 . Además incluye un CPLD que permite configurar el FPGA a través del bus PCI.

Otras interfaces de entrada/salida que tiene la placa son RS-232, USB y 5 puertos Ethernet. Todas estas interfaces están implementadas a través de integrados que llevan a cabo parte del stack de protocolos necesario en cada caso. Cuenta además con un receptor y transmisor infrarrojo. Los elementos de almacenamiento de datos con los que cuenta cada placa son dos SDRAM de 16MB cada una.

Para la alimentación utiliza conversores AC-DC para generar las fuentes de 3.3V y 2.5V utilizadas por lo integrados. Permite conectar una fuente de alimentación externa, para utilizar la placa sin estar conectada al bus PCI.

El FPGA puede ser programado utilizando una EPROM de programación

EPC2, para autoconfigurarse mediante el encendido de la alimentacióno o utlizando el protocolo JTAG configuración serie pasiva (PS) mediante el cable de programación ByteBlasterMV.

El reloj de las placas se obtiene de un generador de reloj que regenera y multiplica la frecuencia de oscilación que se le da como entrada. Para algunos circuitos como ser las interfaces USB y Ethernet se utilizan cristales y osciladores independientes. Por último, cuenta con expansores, llaves, botones y LEDs de propósito general, que permiten ampliar las funcionalidades de la placa.

Parte I

Ethernet sin Leon

En esta etapa del proyecto se diseñó un core para ser programado en el FPGA de la placa PgVirtex que sea capaz de manejar los puertos Ethernet de la misma, en particular el envío y recepción de paquetes. Este core implementa la subcapa MAC[3] (capa de enlace de datos) que se conecta con el chip PHY[4] existente en la placa, además de una RAM donde se almacenan tanto los paquetes para ser enviados como los que se reciben.

5. Protocolo Ethernet

A continuación se explica el protocolo Ethernet, del cual es necesario tener conocimiento para comprender lo que se realizó en nuestro proyecto. Los datos proporcionados son en su mayoría sacados de la web[2], aunque se utilizó otra fuente la cuál es oportunamente mencionada. Al final de la sección se explica brevemente el protocolo UDP, el cuál fue utilizado para las pruebas de esta etapa del proyecto.

El protocolo Ethernet refiere a la familia de las redes de área local (LAN) cubiertas por la norma IEEE 802.3. En los estándares de Ethernet existen dos modos de operación: half duplex (significa que el método de envío de información es bidireccional pero no simultáneo)y full duplex (envío y recepción simultáneos). En el modo half duplex los datos son transmitidos usando el protocolo CSMA/CD (carrier sense multiple access / collision detection) en un medio compartido.

El sistema Ethernet consiste en 3 elementos básicos:

- El medio físico para transportar señales.
- Un conjunto de reglas integradas de acceso al medio en cada interfaz Ethernet que permite el acceso a varias computadoras al canal compartido.
- Un marco Ethernet que consta de un conjunto estandarizado de bits utilizados para transportar datos a lo largo del sistema.

Al igual que con todos los protocolos IEEE 802, la capa de enlace de datos se divide en dos subcapas IEEE 802, la subcapa MAC (Media Access Control) y la subcapa MAC-client. La capa física IEEE 802.3 corresponde a la capa física del modelo ISO.
La subcapa MAC tiene dos resposabilidades principales:

[3]Media Access Controller
[4]dispositivo de capa física del modelo OSI

- Encapsulación de datos, incluyendo montaje de tramas previo a la transmisión y análisis y detección de errores durante y después de la recepción.

- Control de acceso al medio, incluído el inicio de la transmisión de tramas y recuperación de la transmisión fallida.

Todas las estaciones conectadas a través de Ethernet están conectadas a un medio de señales compartidas. Para enviar datos, una estación primero escucha al canal y luego cuando este esta inactivo, la estación transmite sus datos en forma de tramas Ethernet o paquetes. Luego de cada transmisión, todas las estaciones de la red quedan a la espera de la próxima transmisión en igualdad de condiciones.

El acceso al canal compartido esta determinado por la MAC, mecanismo integrado en la interfaz Ethernet situada en cada estación. El mecanismo de acceso al medio se basa en un sistema llamado Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Las velocidades de envío de paquetes utilizando la tecnología Ethernet son de 10 Mbps (Ethernet estándar), 100 Mbps (Fast Ethernet – 100BASEX) y de 1000 Mbps utilizando el Gigabit Ethernet.

Como cada trama Ethernet es enviada al mismo canal compartido, todas las interfaces Ethernet miran la dirección de destino. Si esta dirección de destino coincide con una de las direcciones de las interfaces, la trama será entregada completamente al software de red de esa computadora. Las demás interfaces de la red dejarán de leer la trama cuando descubran que la dirección de destino no coincide con su propia dirección. Para asignar una dirección, los fabricantes de hardware de Ethernet adquieren bloques de direcciones MAC y las asignan en secuencia conforme fabrican el hardware de interfaz Ethernet, de esta manera no existen dos unidades de hardware de interfaz que tengan la misma dirección MAC.

Por lo general, las direcciones MAC se colocan en el hardware de interfaz anfitrión de las máquinas de tal forma que se puedan leer. Debido a que el direccionamiento Ethernet se da entre dispositivos de hardware, a estos se les llama direccionamientos o direcciones físicas. La trama de Ethernet es de una longitud variable pero no es menor a 64 bytes ni rebasa los 1518 bytes (encabezado, datos y CRC), cada trama contiene un campo con la

información de la dirección de destino.

Tabla 1: Trama IEEE 802.3[2]

Preámb	**SOF**	**Destino**	**Origen**	**Tipo**	**Dato**	**Relleno**	**FCS**
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	0 a 1500 bytes	0 a 46 bytes	4 bytes

- Preámbulo (PRE) - 7 bytes. El preámbulo es un patrón de alternancia de unos y ceros que le dice a las estaciones receptoras que una trama viene, y que proporciona un medio para sincronizar las porciones de la recepción de la trama de las capas física con el flujo de bits.
- Inicio de trama delimitador (SOF) - 1 byte. El SOF es un patrón de alternancia de unos y ceros, terminando con dos bits 1 consecutivos indicando que el próximo es el bit más significativo en el primer byte de la dirección de destino.
- Dirección de destino (DA) - 6 bytes. Este campo identifica que estación(es) deben recibir la trama.
- Dirección de Origen (SA) - 6 bytes. Este campo identifica la estación que envía.
- Largo/Tipo- 2 bytes. Este campo indentifica el protocolo de red de alto nivel asociado con el paquete ó la longitud del campo de datos.
- Datos - es una secuencia de n bytes (46 = n = 1500) de cualquier valor. (El valor mínimo de una trama Ethernet es 64 bytes).
- Trama de verificación de secuencia (FCS) - 4 bytes. Esta secuencia contiene un chequeo de control de redundancia cíclica (CRC) de 32-bits, que es creado por la MAC que envía y se vuelve a calcular por la MAC que recibe para comprobar si hay tramas dañadas.

100BaseT es la especificación de IEEE para la implementación física de Fast Ethernet 100-Mbps utilizando para el cableado un par trenzado[3]. La capa de acceso de control al medio (MAC) especificada por este estándar es compatible con la capa MAC definida por IEEE 802.3. Los componentes que se incluyen la norma para la conexión física 100BaseT son los siguientes:

- Medio físico: se elige entre 3 posibilidades; 100BaseTX, 100BaseFX, 100BaseT4.

- MDI[5]: interfaz eléctrica y mecánica entre el medio de transmisión y el dispositivo de la capa física (PHY).
- Dispositivo de capa física (PHY).
- MII[6]: interfaz que comunica el dispositivo PHY con el dispositivo o implementación MAC.
- Dispositivo de subcapa de acceso de control al medio (MAC)

.

User Datagram Protocol (UDP)

UDP es un protocolo de nivel de transporte basado en la transferencia de datagramas. En la familia de protocolos de Internet, UDP proporciona una sencilla interfaz entre la capa de red y la capa de aplicación. UDP no otorga garantías para la entrega de sus mensajes y el orígen UDP no retiene estados de los mensajes UDP que han sido enviados a la red. UDP solo añade multiplexado de aplicación y suma de verificación de la cabecera y la carga útil. Cualquier tipo de garantías para la transmisión de la información, deben ser implementadas en capas superiores.

Tabla 2: Paquete UDP

+	**Bits 0-15**	**16-31**
0	Puerto origen	puerto destino
32	Longitud del mensaje	Suma de verificación
64	Datos	

La cabecera UDP consta de 4 campos de los cuales 2 son opcionales (puerto orígen y suma de verificación). Los campos de los puertos fuente y destino son campos de 16 bits que identifican el proceso de orígen y recepción. Ya que UDP carece de un servidor de estado y el orígen UDP no solicita respuestas, el puerto orígen es opcional. En caso de no ser utilizado, debe ser puesto a cero. A los campos del puerto orígen y destino le sigue un campo obligatorio que indica el tamaño en bytes del datagrama UDP incluidos los datos.

El valor mínimo es de 8 bits. El campo de la cabecera restante es un checksum de 16 bits que abarca la cabecera, una pseudo-cabecera con las

[5]Medium-Dependent Interface
[6]Media Independent Interface

IP de orígen y destino, el protocolo, la longitud del datagrama y 0's hasta completar un múltiplo de 16, pero no los datos. El checksum también es opcional, aunque en la práctica generalmente se utiliza.

Este protocolo se utiliza cuando se necesita transmitir voz o video y resulta más importante transmitir con velocidad que garantizar el hecho de que lleguen absolutamente todos los bytes.

6. Estudio de los chips

La placa PgVirtex tiene 5 bocas de red y 2 chips controladores Ethernet. Uno de los chips existentes en la placa es el LXT974. Este chip es un transciever de 4 puertos Fast Ethernet que soporta redes de 10 Mbps y 100 Mbps y cumple con todos los requerimientos del estándar IEEE 802.3[9]. Este chip está conectado a 4 conectores RJ-45, los cuales ofrecen la conexión con el medio físico. A estos conectores además se le anexaron unos LEDs que permiten determinar entre otras cosas el estado del enlace de capa física de cada boca de red. Igualmente los LEDs son configurables.

La quinta boca de red también es un conector RJ-45 (este encapsulado incluye los LEDs antes mencionados) que se conecta al otro chip, el LAN91C111. Este chip fue diseñado para facilitar la implementación de conexiones de Fast Ethernet para aplicaciones embebidas. Es un dispositivo que mezla señales analógicas y digitales e implementa las subcapas PHY y MAC del protocolo CSMA/CD para 10 Mbps y 100 Mbps[12].

6.1. Chip LXT974

Cuando se pone en funcionamiento, el chip utiliza la detección de auto negociación en cada puerto para determinar automáticamente las condiciones de operación. Si el dispositivo PHY en el otro lado del enlace soporta auto negociación, el LXT974 autonegocia con el otro chip para determinar la velocidad de conexión. De lo contrario, el chip detecta la presencia de "link pulses" (10 Mbps PHY) ó "idle symbols" (100 Mbps PHY) con lo cual setea la correcta condición de operación.

El LXT974 soporta cuatro MIIs[7] (una por puerto). Esta interfaz a su vez

[7]Media Independent Interface

consiste en una interfaz de datos y otra de control. La interfaz de datos de la MII se encarga de pasar datos entre el chip LXT974 y una o más MACs. Se tienen señales separadas para la transmisión y recepción de datos. Esta interfaz puede operar tanto a 10 Mbps como a 100 Mbps. La velocidad queda determinada por las condiciones de operación.

Se disponen las siguientes nueve señales para la recepción de datos:

- RXD[3:0] - bus de datos de recepción
- RX-CLK - señal de clock de recepción
- RX-DV - señal que indica cuando se recibe un dato válido. Se setea en el primer nibble del preámbulo y permanece seteada hasta el final del mismo.
- RX-ER - Error en recepción. Siempre que se recibe un símbolo erróneo de la red se setea esta señal y se escribe el dato "1110" en el bus de datos (RXD).
- COL - señal asíncrona que indica colisión de paquetes. Se activa por alto siempre que se está trabajando en modo half duplex y la transmisión y recepción están activos a la misma vez.
- CRS - señal de salida asíncrona de detección de portadora. Se setea cada vez que se recibe un paquete y en algunos casos en transmisión, solo si se opera en modo half-duplex.

Para la transmisión de datos se dispone de las siguientes siete señales:

- TXD[3:0] - bus de datos de transmisión. Su frecuencia se setea automáticamente según la velocidad de operación: 2.5 MHz a 10 Mbps y 25 MHz a 100Mbps. Las señales de datos y de control deben ser sincronizadas con este reloj por la subcapa MAC. Normalmente, el LXT974 muestrea las señales en el flanco de subida.
- TX-CLK - señal de clock de transmisión.
- TX-EN - Transmit Enable. La subcapa MAC debe setear esta señal junto con el envío del primer nibble del preámbulo y mantenerla en

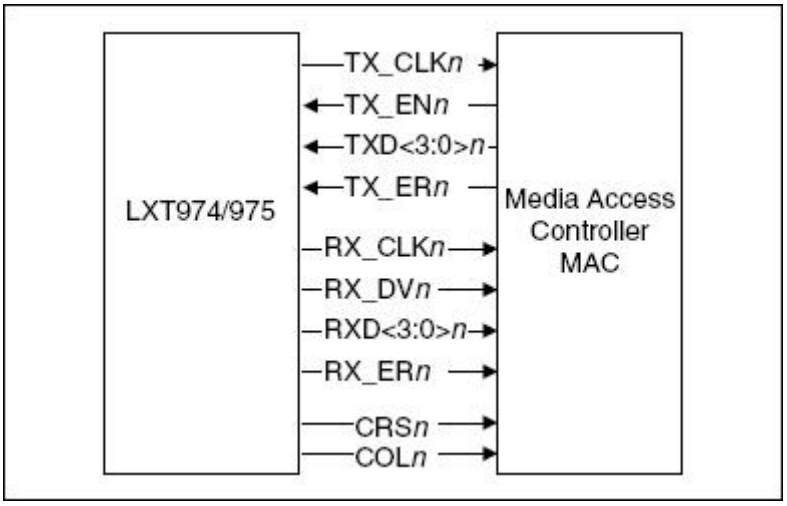


Figura 3: Conexión de Interfaz MII[9]

'1' hasta el último bit de cada paquete.

- TX-ER - Error en transmisión. Cada vez que la subcapa MAC setea esta señal, el chip despliega "1111" en el bus de datos.

Interfaz de datos de la MII:

Tanto el canal de transmisión como el de recepción tienen su propia señal de reloj, señales de control y bus de datos. El chip brinda ambas señales de reloj, y señales separadas para detección de la portadora y colisiones. La transmisión de datos a través de la MII es implementada de a nibbles (los buses de datos son de ancho 4).

Interfaz de control de la MII:

El chip LXT974 soporta la interfaz de control de la MII especificada por IEEE 802.3. Esta interfaz permite que los dispositivos de las capas y subcapas superiores puedan monitorear y controlar el estado del chip. Consiste de solo dos señales: una de reloj (MDC[8]) y una bi-direccional de datos (MDIO[9]). Esta señal bi-direccional es una combinación de 3

[8]Management Data Clock
[9]Management Data Input/Output Interface

señales, una de entrada de datos (Mdi), una de salida de datos (Mdo), y una señal de enable (MdoEn). Está hecho de esta forma para poder implementar el mismo core Ethernet tanto en un FPGA como en un ASIC[10].

La operación de esta interfaz es controlada por el pin de entrada MDDIS. Cuando este pin está en '1', la MDIO opera como una interfaz de solo lectura. Cuando el pin está en '0', se habilitan la lectura y la escritura.

6.2. Pruebas básicas de funcionamiento

De la documentación del proyecto PgVirtex[8] pudimos obtener información acerca de las pruebas que se realizaron sobre los chips de la placa.

Para cada uno de los chips se realizo un core que permitía configurarlos de manera adecuada. De esta forma, al ser conectados a través de un patch cord a otro puerto Ethernet, ambos chips efectúan el algoritmo de auto negociación y se establece el enlace de capa física entre cada chip y su par remoto correspondiente.

La velocidad del enlace y el modo de transferencia de datos (half duplex ó full duplex) depende de las configuraciones de cada dispositivo y su par remoto.

Como se explica claramente en dicha documentación, estas pruebas permiten exclusivamente verificar que exista un enlace a nivel de capa física, lo cual se verifica observando que se enciendan los LEDs en la placa que justamente indican el estado de los enlaces.

Las primeras pruebas que realizamos sobre la placa, fueron precisamente estas mismas, para verificar que realmente al menos se podía establecer un enlace de capa física en cada puerto Ethernet. Las pruebas realizadas para los cuatro puertos conectados al chip LXT974 fueron satisfactorias.

En cambio, con las pruebas del puerto Ethernet conectado al chip LAN91C111 pudimos advertir que estos chips no estaban funcionando en ninguna de las dos placas. Adjunto con las placas se nos entregó una nota perteneciente

[10] Application-Specific Integrated Circuit

al grupo del proyecto que desarrolló la placa que informaba que efectivamente el puerto Ethernet había dejado de funcionar en una de las placas y que en el otra nunca había funcionado. En la nota incluso se informa que se sustituyó el chip en una de las placas sin resultados positivos, y que se sospechaba del conector RJ-45 cuyos contactos podían estar gastados.

Cabe acotar que nuestro grupo ya había sido advertido acerca del posible no funcionamiento de este puerto Ethernet en una de las placas al comenzar el proyecto. Debido a esto, en primera instancia nos enfocamos más al diseño de un core que utilizara el chip LXT974. Aparte podíamos presuponer que el diseño para el LAN91C111 podía ser más sencillo, ya que este chip implementa la subcapa MAC además de la PHY.

Según nuestro análisis, los conectores del puerto de una boca se encuentran en buen estado en ambas placas y no pudimos determinar la causa del problema con estos puertos. Dado que recibimos las placas bastante más tarde de lo que esperábamos, en el momento de comenzar a implementar nuestro diseño nos vimos muy ajustados por el calendario por lo que decidimos descartar en ese momento el diseño del core para el chip LAN91C111 y abocarnos de lleno al diseño de un core para el chip LXT974.

7. Diseño

El propósito de nuestro diseño era, para el caso del chip LXT974, poder implementar en un core la subcapa MAC y poder transferir datos a través de los puertos Ethernet hacia otro terminal Ethernet.

Para este propósito, detallaremos en las siguientes secciones, los requerimientos de nuestro diseño, y las opciones que fuimos eligiendo para lograr el resultado deseado.

7.1. Requerimientos

Además del diseño de la subcapa MAC, era necesario diseñar la interfaz MII que permite la conexión entre esta subcapa y la capa física (PHY)

que implementa el chip.

Igualmente, una vez que lográramos esta implementación, de alguna forma teníamos que generar los paquetes que se quieran transferir, o bien almacenar los paquetes que se recibieran en alguno de los puertos. Por este motivo, decidimos que en una primera instancia además del bloque MAC, deberíamos implementar otro bloque que lo pudiera manejar y que a su vez manejase una memoria RAM donde guardar la información transmitida o recibida por los puertos Ethernet.

Durante la investigación de como implementar el core MAC y la interfaz MII que permitiera conectarnos con el chip, surgió la posibilidad de utilizar y/o adaptar cores ya diseñados que se encontraban disponibles para bajar, principalmente de la web de Opencores[11].

Resumiendo, deberíamos diseñar e implementar un bloque capaz de manejar la subcapa MAC, una RAM donde almacenar datos a ser enviados y recibidos, y decidir entre diseñar la capa MAC y la comunicación con el PHY o implementar un core ya diseñado y realizar las adaptaciones necesarias a nuestro diseño.

7.2. Elección del core MAC

Luego de optar por dos cores como posibles candidatos, comenzamos a evaluarlos por separado para tomar la decisión de utilizar uno de ellos ó diseñar uno propio.

7.2.1. GRETH 10/100 Mbit Ethernet MAC

El core Greth[6] implementa la subcapa de control de acceso al medio (MAC) para 10/100 Mbit/s Ethernet con interfaces AMBA[12]. El core implementa las especificaciones del estándar 802.3-2002 de Ethernet y soporta interfaces tanto MII como RMII[13]. Utiliza DMA[14] para recibir y

[11] www.opencores.org
[12] Advanced Microcontroller Bus Architecture
[13] Reduced Media Independent Interface
[14] Direct Memory Access

transmitir datos entre el core y el bus AMBA AHB[15].

Mediante el uso de descriptores de transmisión y recepción se pueden enviar y recibir múltiples paquetes sin la necesidad del uso de un CPU durante el proceso. El control de los registros del core Greth se realiza mediante una interfaz APB[16].

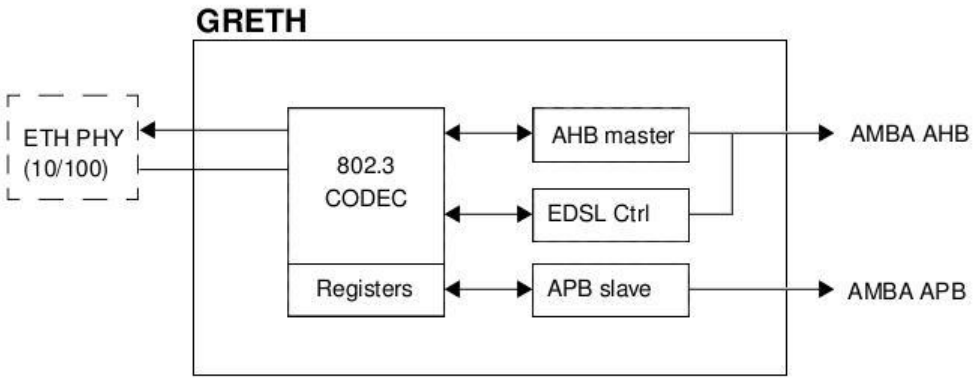


Figura 4: Arquitectura del Greth Ethernet Core[6]

De un estudio superficial de las características de este core, pudimos deducir que iba a ser necesario que nuestro diseño tuviera interfaces AMBA para poder comunicarse con el core, por lo que tendríamos que invertir tiempo en el estudio de este protocolo de comunicación. Para la utilización de este core, ya que utiliza DMA para el envío y recepción de datos, queda para nuestro diseño solamente manejar los registros de control del core mediante una interfaz AMBA (master), y dotar a la RAM de nuestro diseño de una interfaz AMBA (slave) para ser conectada al core. La conexión con el PHY parecería bastante simple en primera instancia ya que tanto el chip LXT974 como la MII implementada por el core son compatibles con la especificación del estándar 802.3.

[15] Advanced High-performance Bus
[16] Advanced Peripheral Bus

7.2.2. Ethernet IP Core de Opencores

El Ethernet IP Core de Opencores[10] implementa una MAC. Se conecta por un lado al chip PHY Ethernet y por el otro brinda una interfaz Wishbone para ser conectada al host. El core ha sido diseñado para ofrecer la mayor flexibilidad posible a todo tipo de aplicaciones.

Sus principales características son la de implementar la subcapa MAC y la MII según la especificación IEEE 802.3. Implementa una RAM interna que permite almacenar hasta 128 descriptores de transmisión y recepción lo que permite múltiples procesos de envío y recepción sin la necesidad del uso del CPU. Cuenta además con generación de preámbulo en los paquetes, generación de CRC, 10 y 100 Mbps de velocidad y permite trabajar full duplex, incluyendo el control de flujo, ó en half duplex, con retransmisión automática en caso de colisión.

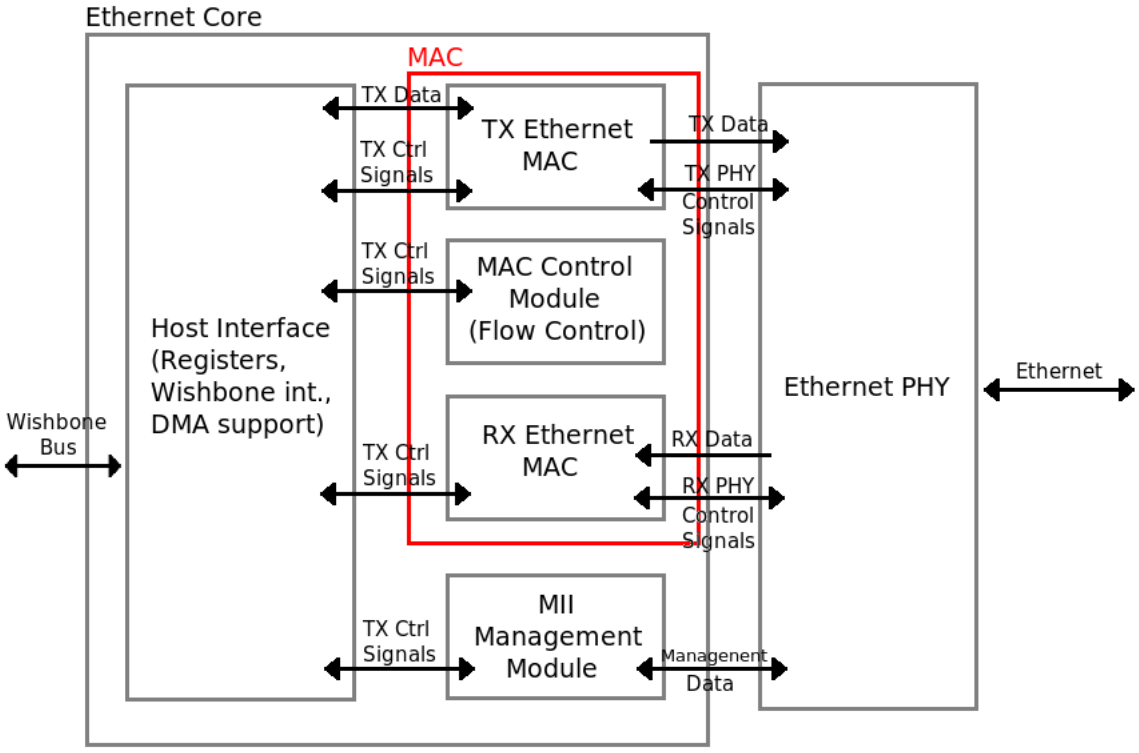


Figura 5: Arquitectura del Ethernet IP Core[10]

En esencia es muy similar al core Greth Ethernet, con la diferencia de que utiliza, del lado del host, interfaces Wishbone en lugar de AMBA lo que a priori podría ser una ventaja dado que Wishbone es un protocolo bastante simple y con especificaciones muy generales. Al igual que el otro core, el Ethernet IP Core de Opencores trabaja con DMA para la lectura y escritura de la RAM.

Además de la subcapa MAC, el core incluye un testbench que implementa un core de una capa PHY Ethernet muy similar a la del chip existente en la placa. En este punto de nuestro proyecto, aún no contábamos con la placa, por lo que este testbench nos sería de gran utilidad para por lo menos simular el comportamiento del chip real y así comprender un poco más acerca del manejo de las señales, sobre todo de la MII, y por ende la configuración de los registros del core MAC.

Finalmente, debido al protocolo en las interfaces hacia el host, y con el agregado de este testbench decidimos optar por utilizar este core ya que teníamos que diseñar y comenzar a realizar pruebas sin tener la placa.

A medida que fuimos estudiando estos cores, fuimos descartando la idea de diseñar e implementar nuestro propio core MAC, lo que seguramente nos habría insumido más tiempo debido a la complejidad y nuestra inexperiencia en este tipo de diseños.

7.3. Wishbone

"El objetivo de la especificación Wishbone es crear una interfaz común entre IP Cores.

Define una forma estándar de intercambiar datos entre módulos IP Core. No pretende especificar el funcionamiento del core, solo la forma de comunicarse con él.

La arquitectura WISHBONE es análoga al bus de un microprocesador. Ofrece una solución flexible de integración, varios tipos de ciclos y anchos de bus para resolver distintas situaciones, y es versátil al desarrollo de una misma aplicación por varios fabricantes."[5]

Las características más sobresalientes de la especificación son:

- Basado en protocolos estándares de transferencia de datos
  - ciclos READ/WRITE
  - ciclos de transferencia en BLOQUE
  - ciclos RMW
- Soporta varios tipos de interconexionamientos
  - punto a punto
  - bus compartido
  - switch de interconexiones
- Protocolo de Handshake para regular la velocidad de transferencia de datos
- Soporta varias terminaciones de ciclos
  - normal
  - con retry
  - por error
- Basado en arquitectura Maestro/Esclavo

La especificación Wishbone[5] define:

- Reglas, las cuales deben ser seguidas para asegurar una compatibilidad entre interfaces.
- Recomendaciones, que se aconsejan seguir. El no adoptarlas puede implicar una pérdida de performance. Muchas recomendaciones están basadas en la experiencia acumulada de los diseñadores que escribieron la especificación.

- Consejos, que pueden ser considerados por el diseñador. Pueden ser útiles, pero no vitales para el funcionamiento de la interfaz.

- Permisos, que indican si algo puede hacerse o no, la decisión de implementarlos puede quedar a cargo del diseñador.

- Observaciones, que usualmente son textos que clarifican alguna situación, pero no aportan nada más que eso.

La convención para el nombre de la señales indica que todas las señales tienen "_I" y "_O" para indicar si son salidas o entradas al core. Los buses son indicados por nombres seguidos de "()". Por ejemplo DAT_I(). Pueden utilizarse señales especiales definidas por el usuario, por ejemplo PAR_O. Estas señales son llamadas tags. Los tags tiene asociados un TAG TYPE que indican en que momento del ciclo tienen un valor válido.

La idea es que las señales permitan las conexiones punto a punto, bus compartido, etc. Permiten tres ciclos básicos: Read, Write y RMW. Pero no hay obligación de soportarlos a todos. Permiten un handshake para adecuar la velocidad de transferencia de datos, e indicar errores y retry. Las señales no son bi-direccionales, siempre son entradas o salidas. Esto es así pues muchas veces el diseño puede llegar a querer implementarse en hardware que no soporta internamente señales bi-direccionales, por ejemplo los FPGAs de Altera. Además todas las señales son activas por nivel alto.

Algunas señales comunes a master y slave:

- CLK_I: entrada de reloj.
- RST_I: reset del diseño.
- DAT_I() y DAT_O(): buses de entrada y salida de datos.

Señales master:

- ADR_O(N..n); N: límite superior dado por el ancho del bus de direcciones; n: límite inferior dado por la granularidad del bus de datos.

- CYC_O: indica que se esta llevando a cabo un ciclo de bus válido. Se activa al comienzo del ciclo y permanece hasta el final.

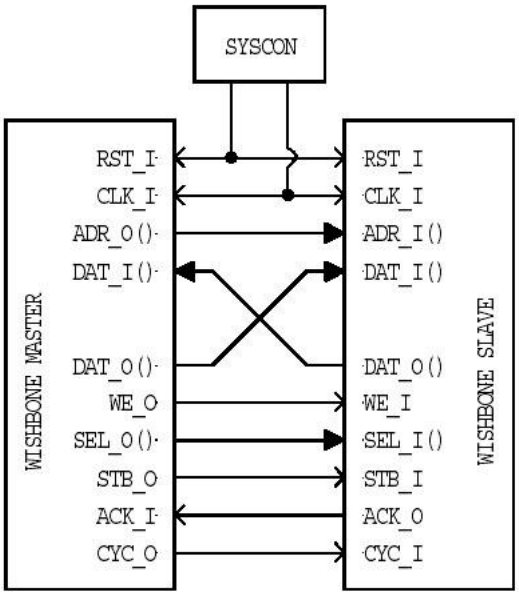


Figura 6: Conexión Master-Salve Wishbone[5]

- WE_O: indica si el ciclo es de lectura o escritura.
- STB_O: indica que se esta llevando a cabo un ciclo valido de transferencia de datos.
- ACK_I: recibe la confirmación de una transferencia.
- ERR_I: recibe la indicación de un error en la transferencia.
- RTY_I: recibe pedido de re-transmisión de datos.
- LOCK_O: indica que el ciclo que se esta llevando a cabo no puede ser interrumpido.
- SEL_O(): asociado con la granularidad, indica donde hay o donde espera datos válidos en el bus.

Señales slave

- ADR_I(N..n); N: límite superior dado por el ancho del bus de direcciones; n: limite inferior dado por la granularidad del bus de datos.

- CYC_I: indica el comienzo de un ciclo.

- WE_I: Indica si el ciclo es de lectura o escritura.

- STB_I: indica que se esta llevando a cabo un ciclo valido de transferencia de datos.

- ACK_O: indica que la transferencia se ha realizado en forma exitosa.

- ERR_O: utilizado para señalizar un error en la transacción.

- RTY_O: utilizado para pedir una reintento en la transacción.

- LOCK_I: indica que el ciclo que se esta llevando a cabo no puede ser interrumpido.

- SEL_I(): asociado con la granularidad, indica donde hay o donde espera datos válidos en el bus.

Funcionamiento General

EL reset se produce en forma síncrona, y llevando la señal RST_I a '1'. Todas las interfaces WISHBONE deben de inicializarse con el primer flanco de subida en el que RST_I este activo.

Se indica que hay un ciclo válido si CYC_O esta activo. Cuando CYC_O es '0' ninguna de las otras señales del MASTER tienen sentido.

El handshake para la transferencia de datos es sumamente sencillo, el MASTER activa STB_O y lo mantiene así hasta que el esclavo activa alguna de las señales ACK_I, ERR_I, RTY_I. Al ocurrir esto el Maestro desactiva la señal. Las señales de respuesta ACK_O, ERR_O, RTY_O deben generarse solo si están activas CYC_I y STB_I. Las interfaces del esclavo deben de ser diseñadas para que ACK_O, ERR_O, RTY_O se activen y desactiven respondiendo a STB_O.

Durante las transferencias (escrituras y lecturas) en bloque (ver figuras 9 y 8), la interfaz básicamente realiza transferencias simples (lectura y escritura) y puede realizarse una en cada ciclo de reloj.

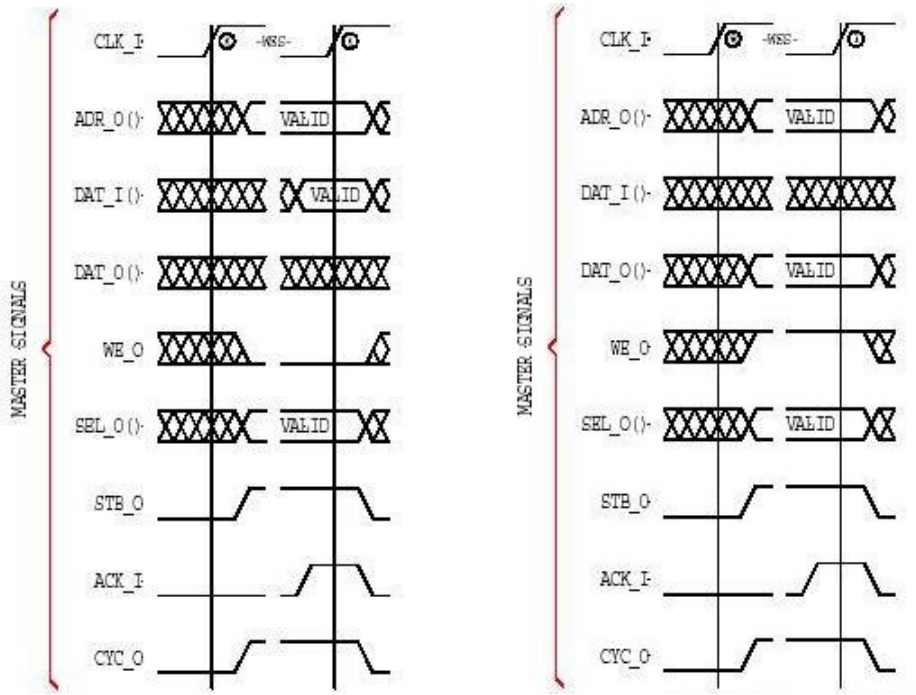


Figura 7: Ciclo simple lectura/escritura Wishbone[5]

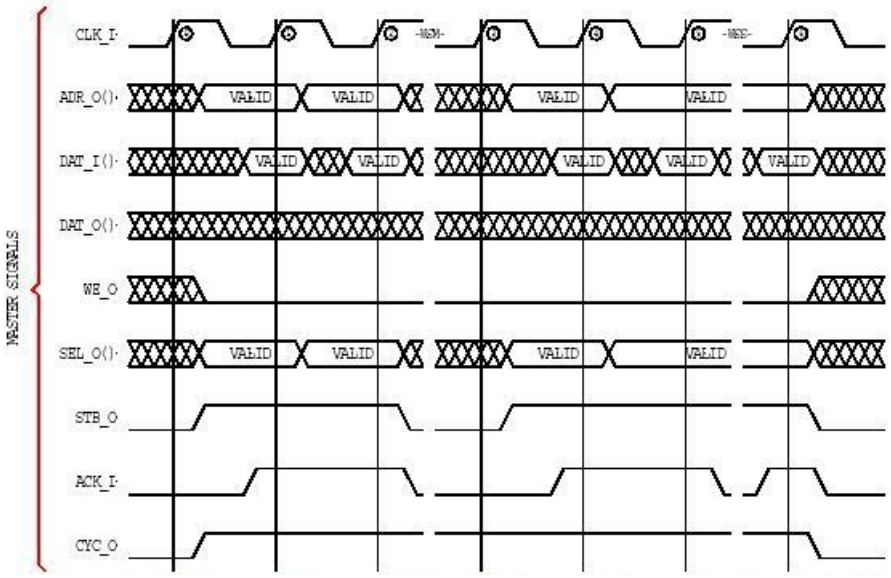


Figura 8: Lectura Wishbone en bloque[5]

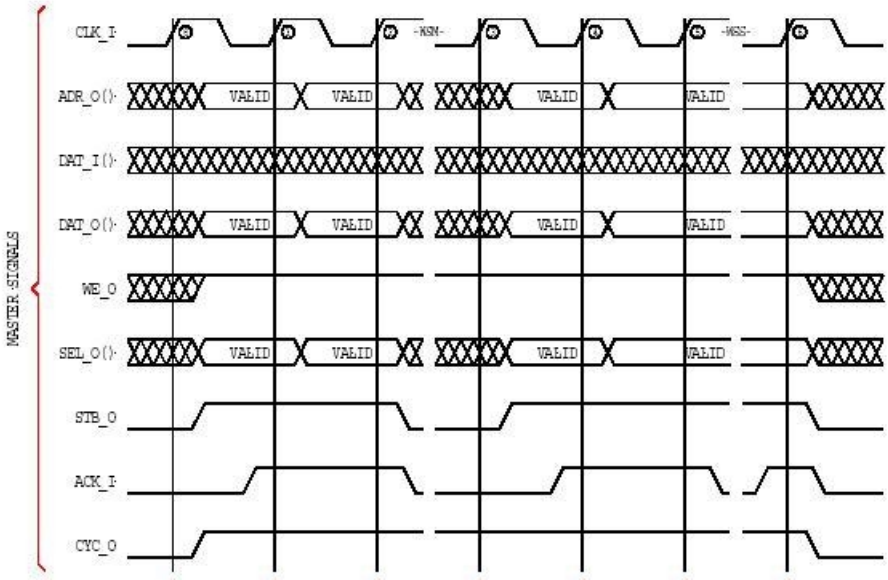


Figura 9: Escritura Wishbone en bloque[5]

7.4. Puertos de Entrada/Salida del Ethernet IP Core de Opencores

Como vimos en las secciones anteriores, el Ethernet MAC IP Core implementa dos tipos de señales además de las señales utilizadas para resetear el core MAC.

Por un lado tenemos las señales de Wishbone, que utilizaremos para conectar a un core diseñado por nosotros (que sea capaz de manejar el core MAC) mediante una interfaz slave y para conectar a la RAM donde escribiremos los paquetes recibidos y a ser enviados, mediante una interfaz master. Por otro lado tenemos las señales que utiliza el core MAC para conectarse a un chip PHY.

En la tabla 3 se muestran las señales Wishbone que dispone el core MAC para ambas interfaces (master y slave), y en la tabla 4 vemos las señales que se tienen en el core MAC para conectarse con el PHY. Además de las de las interfaces Wishbone, todas las señales son activas por nivel alto a menos que se explicite lo contrario.

Tabla 3: Señales Wishbone del core MAC[10]

SEÑAL	**ANCHO**	**TIPO (I/O)**	**DESCRIPCION**
CLK_I	1	I	Entrada de Reloj
RST_I	1	I	Entrada de Reset
ADDR_I	32	I	Bus de entrada de direcciones
DATA_I	32	I	Bus de entrada de datos
DATA_O	32	O	Bus de salida de datos
SEL_I	4	I	Array de selección de entradas. Indica que bytes son válidos en el bus de datos. Siempre que esta señal no sea "1111" durante una validación de acceso, la señal ERR_O se lleva a '1'.
WE_I	1	I	Write Enable. Indica un ciclo de escritura cuando está en '1' o uno de lectura si está en '0'.
STB_I	1	I	Entrada de Strobe. Indica el comienzo de un ciclo válido de transferencia de datos.
CYC_I	1	I	Entrada de Ciclo. Indica que un ciclo de bus válido está en progreso.
ACK_O	1	O	Salida de acknowledge. Indica el final normal de un ciclo.
ERR_O	1	O	Salida de acknowledge por error. Indica el final de un ciclo con error.
INTA_O	1	O	Salida de interrupción.
M_ADDR_O	32	O	Salida de direcciones.
M_DATA_I	32	I	Bus de entrada de datos.
M_DATA_O	32	O	Bus de salida de datos.
M_SEL_O	4	I	Array de selección de salidas. Indica que bytes son válidos en el bus de datos. Siempre que esta señal no sea "1111" durante una validación de acceso, la señal ERR_I se lleva a '1'.
M_WE_O	1	O	Write Enable. Indica un ciclo de escritura cuando está en '1' o uno de lectura si está en '0'.
M_STB_O	1	O	Salida de Strobe. Indica el comienzo de un ciclo válido de transferencia de datos.
M_CYC_O	1	O	Salida de Ciclo. Indica que un ciclo de bus válido está en progreso.
M_ACK_I	1	I	Entrada de acknowledge. Indica el final normal de un ciclo.
ERR_I	1	I	Entrada de acknowledge por error. Indica el final de un ciclo con error.

Tabla 4: Señales del core MAC hacia el PHY[10]

SEÑAL	ANCHO	TIPO (I/O)	DESCRIPCION
MTxClk	1	I	Transmite nibbles o símbolos de reloj. El PHY provee la señal MTxClk. Opera a una frecuencia de 25 MHz (100 Mbps) ó 2.5 MHz (10 Mbps). Es el reloj con el que se sincronizan las señales MTxD[3:0], MtxEn y MTxErr.
MTxD	4	O	Transmite nibbles de datos. Se sincronizan con el flanco de subida de MTxClk. Si se setea la señal MTxEn, el PHY acepta el dato de MTxD.
MTxEn	1	O	Transmit Enable. Cuando se setea esta señal indica al PHY que el dato en MTxD[3:0] es válido y la transmisión puede comenzar. La transmisión comienza con el primer nibble del preámbulo. La señal permanece en '1' hasta que todos los nibbles a ser transmitidos son entregados al PHY. Se vuelve a '0' con el primer flanco de MTxClk, luego del último nibble del paquete.
MTxErr	1	O	Transmit Error. Cuando se setea durante un período de MTxClk, estando además seteada la señal MTxEn, esta señal obliga al PHY a transmitir uno o más símbolos no válidos para el paquete para indicar que hubo un error en la transmisión del código.
MRxClk	1	I	Recibe nibbles o símbolos de reloj. El PHY provee la señal MRxClk. Opera a una frecuencia de 25 MHz (100 Mbps) ó 2.5 MHz (10 Mbps). Es el reloj con el que se sincronizan las señales MRxD[3:0], MRxDV y MRxErr.
MRxDV	1	I	Recepción de datos válidos. El PHY setea esta señal para indicar al Rx MAC que le está entregando datos válidos en el bus MRxD[3:0]. La señal se setea en forma síncrona con MRxClk a partir del primer nibble de cada paquete y permanece en '1' hasta el último nibble. Se lleva nuevamente a '0' con del primer flanco de MRxClk después del último nibble.

SEÑAL	ANCHO	TIPO (I/O)	DESCRIPCION
MRxD	4	I	Recepción de nibbles de datos. Este bus recibe los nibbles de datos en forma síncrona con MRxClk. Cuando se setea MRxDV, el PHY envía nibbles de datos al Rx MAC. Para un paquete válido se deben pasar a través de la interfaz los 8 bytes de preámbulo (7 bytes y el byte de SOF).
MRxErr	1	I	Error en recepción. El PHY setea esta señal para indicar al Rx MAC que se detectó un error en el medio durante la transmisión del paquete. MRxErr está sincronizada con MRxClk y permanece seteada por uno o más períodos de reloj antes de ser llevada a '0' nuevamente.
MColl	1	I	Detección de colisión. El PHY setea esta señal en forma asíncrona cuando se detecta una colisión en el medio. Mientras esté en '0' significa que no se ha detectado ninguna colisión.
MCrS	1	I	Detección de portadora. El PHY setea esta señal en forma asíncrona luego que se verifica que el medio no está en un estado "idle". Cuando está en '0' significa que el medio está en estado "idle" y que se puede transmitir.
MDC	1	O	Reloj de control de datos (Management Data Clock). Es el reloj utilizado por el canal serial de datos MDIO.
MDIO	1	I/O	Control de datos de entrada/salida (Management Data Input/Output). Canal bidireccional serial de datos para la comunicación PHY/MIIM.

7.5. Registros del core MAC

En esta sección se describen algunos de los registros de configuración del Ethernet IP Core. Básicamente aquellos que utilizamos para el funcionamiento de nuestro diseño.

7.5.1. Buffer descriptors

Dentro del core MAC tanto los procesos de escritura de paquetes como de lectura se realizan mediante los registros llamados buffers descriptors. Existen buffers descriptors de escritura y de lectura. Estos registros son de 64 bits de largo y se almacenan en una RAM interna del core, a partir de la dirección 0x400 y permitiendo almacenar hasta 128 buffer descriptors, o sea hasta la dirección 0x7FF (hay que tener en cuenta que la RAM tiene un ancho de 8 bits).

Existe también un registro llamado TX-BD-NUM que nos indica la cantidad de buffer descriptors de transmisión que han sido guardados. Por lo tanto los buffer descriptors de transmisión se encontrarán entre la dirección 0x400 y la dirección 0x400 mas el número guardado en TX-BD-NUM multiplicado por 8. En consecuencia los buffers de lectura se encontrarán entonces entre la dirección siguiente (adicionándole 8) a la del último buffer descriptor de transmisión y la dirección 0x7FF.

En los primeros 32 bits de cada buffer descriptor se gurda la información sobre el largo paquete y el estado del proceso, mientras que los últimos 32 bits contienen el puntero de la dirección a partir de la cual está grabado el paquete.

En nuestro caso, aquí es donde se indicará la dirección de nuestra RAM a partir de la cual debe comenzar a tomar o grabar los datos el core MAC a través de su interfaz master Wishbone, según sea un ciclo de escritura o lectura respectivamente.

En las tablas 5 y 6 se muestra en detalle como se conforman los buffer descriptors de transmisión y recepción respectivamente.

Tabla 5: Detalle del buffer descriptor de transmisión[10]

Nº DE BIT	**DESCRIPCION**
31-16	LEN – Número de bytes del paquete asociado al buffer descriptor.
15	RD – Ready 0 = El buffer asociado al buffer descriptor no está listo, y puede ser manipulado. Luego de una transmisión de datos o luego de un error, este bit es llevdo a '0'. 1 = El buffer de datos está listo para ser transmitido o está siendo transmitido. No se puede manipular este descriptor mientras el bit está en '1'.
14	IRQ – Interrupt Request Enable 0 = No se genera interrupción luego de la transmisión. 1 = Se genera una interrupción luego de la transmisión.
13	WR – Wrap 0 = No es el último buffer descriptor en la tabla. 1 = Este es el último buffer descriptor en la tabla. Luego de ser utilizado este buffer descriptor, se pasa nuevamente al primero en la tabla.
12	PAD – Pad Enable 0 = No se agregan datos de relleno al final de los paquetes pequeños. 1 = Se agregan datos de relleno al final de los paquetes pequeños.
11	CRC – CRC Enable 0 = No se agrega CRC al final del paquete. 1 = Se agrega CRC al final del paquete.
8	UR – Underrun
7:4	RTRY – Retry Count Indica el número de reenvíos necesarios antes de lograr enviar exitosamente el paquete.
3	RL – Retransmission Limit Este bit es seteado a '1' cuando el envío falla por existir el máximo de colisiones ("retry limit" + 1) al intentar enviar el paquete. Este valor se setea en el registro COLLCONF.
2	LC – Late Collision
1	DF – Defer Indication
0	CS – Carrier Sense Lost

Tabla 6: Detalle del buffer descriptor de recepción[10]

Nº DE BIT	**DESCRIPCION**
31-16	LEN – Número de bytes del paquete asociado al buffer descriptor.
15	E – Empty 0 = El buffer asociado al buffer descriptor fue llenado con datos o se detuvo a causa de un error. El core puede leer o escribir este buffer descriptor. Mientras este bit esté en '0', este buffer descriptor no será utilizado. 1 = El buffer de datos está vacío, listo para recibir datos ó está recibiendo datos.
14	IRQ – Interrupt Request Enable 0 = No se genera interrupción luego de la recepción. 1 = Se genera una interrupción luego de la recepción.
13	WRAP 0 = No es el último buffer descriptor en la tabla. 1 = Este es el último buffer descriptor en la tabla. Luego de ser utilizado este buffer descriptor, se pasa nuevamente al primero en la tabla.
8	CF – Control Frame 0 = Se recibió un paquete normal de datos. 1 = Se recibió un paquete de control.
7	M – Miss 0 = Se recibió el paquete debido a un reconocimiento de dirección. 1 = Se recibió el paquete debido a que se opera en modo promiscuo.
6	OR – Overrun
5	IS – Invalid Symbol Este bit es seteado a '1' cuando el PHY detecta un símbolo inválido.
4	DN – Dribble Nibble Este bit es seteado a '1' cuando no se puede dividir el paquete entre 8. Se recibió un nibble extra.
3	TL – Too Long Este bit This bit es seteado a '1' cuando se recibe un paquete de mayor tamaño al indicado en el registro PACKETLEN.
2	SF – Short Frame Este bit This bit es seteado a '1' cuando se recibe un paquete de menor tamaño al indicado en el registro PACKETLEN.
1	CRC – Rx CRC Error Se setea este bit a '1' cuando existe un error en el CRC del paquete recibido.
0	LC – Late Collision

7.5.2. Registro Moder

Podríamos decir que mediante este registro es que se maneja el envío y recepción de paquetes. Es un registro de 32 bits de escritura y lectura, del

cual se utilizan solo los primeros 16. Cada uno de estos 16 bits indican alguna característica del funcionamiento del core MAC según estén en '1' ó en '0'. Detallaremos la utilidad de alguno de estos bits de configuración.

Si el bit 0 (RXEN) del registro moder está seteado, esto indica que la recepción de paquetes está habilitada. De lo contrario, no está habilitada. Análogamente, el bit 1 (TXEN) del moder indica si está o no habilitada la transmisión de paquetes. Cabe destacar que el seteo o no de estos bits depende también del valor del registro TX-BD-NUM, ya que si este se encuentra en 0, el valor de TXEN automáticamente se pone en '0' deshabilitando el envío de paquetes. Por el contrario si el valor de TX-BD-NUM es 0x80 (valor máximo posible), entonces queda automáticamente deshabilitada la recepción de paquetes llevando a '0' el bit RXEN del moder.

El bit 2 del registro indica si los paquetes se envían o no con encabezado IP. En operación normal este encabezado debe ser agregado por el core MAC, lo cual se indica con un '0' en este bit. Sin embargo podría suceder que este encabezado no sea necesario agregarlo, posiblemente porque este sea agregado previo a ser enviado hacia el core MAC, en cuyo caso se debería setear un '1'.

El tercer bit del moder (BRO) indica con un '0' que se reciben los paquetes que tengan como destino la dirección de broadcast. Si BRO está en '1' se rechazarán los llamados paquetes de broadcast, a menos que el bit 5 (PRO) esté en '1'. En este caso, el core queda en modo promiscuo, por lo cual deja pasar cualquier paquete sin importar la dirección de destino. Para desactivar el modo promiscuo basta con poner en '0' el bit PRO.

El bit 7 determina si se opera en "modo loopback" ('1') o en modo normal ('0'). Si este modo está activado, todos los paquetes que sean transmitidos por la capa MAC, no saldrán a la capa física, sino que serán devueltos a la MAC como si hubieran sido recibidos desde la capa física. Existe también un bit que permite configurar si trabajamos en half-duplex ('0') ó full-duplex ('1'). Es el bit 10 del moder, llamado FULLD. El resto de los bits sirven para configurar detalles sobre el CRC, el tamaño mínimo y máximo de paquetes permitido, y algunas otras particularidades más finas las cuales preferimos dejar por defecto.

Es importante que una vez que se setea el bit TXEN y/o RXEN, no se escriba más el registro moder hasta que se realicen las transmisiones y/o

recepciones deseadas. El hecho de escribir el moder luego de setear alguno de estos bits no nos permite asegurar nada respecto a la configuración con la que se trabaja.

7.5.3. Registros de interrupción

El core MAC cuenta con 2 registros referidos a las interrupciones.

Uno de ellos es el que indica la fuente de las interrupciones (INT_SOURCE). A pesar de ser un registro de 32 bits, se utilizan solo los primeros 7. Cada uno de estos bits indica el status de la interrupción de cada una de las fuentes.

Las fuentes de interrupción por bit son:

- Bit 0: Se genera una interrupción si se transmite un buffer.
- Bit 1: Se genera una interrupción si ocurre un error durante la transmisión de un buffer.
- Bit 2: Se genera una interrupción si recibe un paquete.
- Bit 3: Se genera una interrupción si ocurre un error durante la recepción de un paquete.
- Bit 4: Se genera una interrupción si se descarta un paquete recibido por no tener buffer descriptors de recepción libres.
- Bit 5: Se genera una interrupción si se transmite un paquete de control.
- Bit 6: Se genera una interrupción si se recibe un paquete de control.

Para nuestro diseño nos interesa en particular el bit 2 del registro de interrupciones. Este bit de interrupción se habilita solamente si en el buffer descriptor de recepción está el bit IRQ (que habilita la generación de interrupciones) seteado6. Una vez que se genera la interrupción, el status se limpia escribiéndole un '1' en ese bit.

El otro registro utilizado en las interrupciones es el registro de la máscara de interrupciones (INT_MASK). Como lo indica su nombre, este registro sirve como máscara (como enable) del registro INT_SOURCE. Por lo tanto para que una recepción de un paquete genere una interrupcion, además es necesario setear en '1' el bit 2 del registro INT_MASK.

La máscara funicona de forma análoga para los bits de las demás fuentes de interrupción.

7.6. Testbench del IP Core de Opencores

El paquete IP core además del core MAC trae un módulo llamado testbench, el cual crea un ambiente de testing que permite probar todas las funcionalidades del core MAC.

Para el uso del testbench, existe también un módulo PHY que simula un chip LXT971A de Intel. Este chip es bastante similar al que tiene la placa, de hecho se puede decir que es el mismo chip pero con un solo puerto en vez de 4.

Durante las pruebas con el testbench, cuando se transmiten datos (PHY recibe datos del core MAC), el PHY controla el protocolo del envío. Verifica el preámbulo, el SOF, y escribe los datos en su memoria.

Cuando el PHY envía datos hacia el MAC, puede generar varios paquetes, de diferentes largos, preámbulos erróneos, para así probar las distintas respuestas del core MAC. Los datos los toma de su memoria, por lo que es necesario que en la prueba el módulo de testbench se encargue de escribirla antes de comenzar con la transmisión.

También existe un submódulo Wishbone que monitorea el core (wb_bus_monitor). Este submódulo contiene a su vez dos submódulos cada uno con su interfaz, uno Wishbone master y uno Wishbone slave, que se conectan a las interfaces slave y master del core MAC respectivamente. La interfaz del submódulo master permite escribir los registros del core MAC durante las pruebas y generar ciclos Wishbone desde la interfaz master del core MAC hacia el submódulo slave.

El submódulo slave responde a los ciclos Wishbone iniciados por el core MAC a través de su interfaz Wishbone master. Este submódulo incluye además una RAM estática donde se almacenan los datos a ser enviados y recibidos durante las pruebas.

Las pruebas realizadas por este testbench incluyen entre otras, escrituras y lecturas a los registros del core MAC, escrituras y lecturas a los

buffer descriptors, envío y recepción de distintos paquetes, verificando la respuesta adecuada por parte del módulo PHY.

Durante la etapa de diseño, este core nos fue muy útil, sobre todo para comprender el funcionamiento del core MAC. Nos ayudó mucho a la hora de resolver problemas en nuestro diseño, comparando los resultados de nuestras simulaciones y los de las pruebas con el testbench. Además nos dio la idea de construir nuestro propio testbench, usando el módulo PHY propuesto por el Ethernet IP core, para poder realizar pruebas y simulando el chip real, mientras esperábamos por las placas. Si bien el chip real que tiene la placa no es el mismo que el del módulo, si lo es desde el punto de vista funcional, y nos reflejaría una simulación bastante acercada a la realidad.

7.7. RAM utilizada

Como se explica en las secciones anteriores, era necesario implementar una RAM donde se puedan almacenar los datos que se envíen y reciban.

Además vimos que el core MAC tiene una interfaz Wishbone master, a la cual poder conectar la RAM. Como además iba a ser necesario manejar este core a través de su interfaz Wishbone slave (recordemos que el core MAC utiliza DMA), era necesario que la RAM tuviese una interfaz Wishbone slave para su manejo.

De la web de Opencores se consiguió una "single port" RAM parametrizable, a la cual se le podía anexar un módulo con una interfaz Wishbone slave. Este bloque en conjunto nos queda entonces como una RAM con interfaz Wishbone slave.

Utilizamos una instancia de esta RAM con interfaz Wishbone, a la cual configuramos con un ancho de palabra de 32 bits.

7.8. Diseño del bloque MAC_eth_controller

El bloque MAC_eth_controller fue diseñado para manejar el core MAC y la RAM. Tiene dos interfaces Wishbone master, una para el manejo de la RAM y la otra conectada a la interfaz Wishbone slave del core MAC. En

la tabla 7 se muestran un detalle de las señales de este bloque.

Además en este bloque se diseñaron los procesos necesarios para la escritura y lectura de datos tanto en la RAM, como en los registros del core MAC.

Tabla 7: Señales de MAC_eth_controller

SEÑAL	**TIPO (I/O)**	**DESCRIPCION**
MAC_DAT_O	O	Salida de datos hacia el bloque MAC. Bus de 32 bits.
MAC_DAT_I	I	Entrada de datos desde la MAC. Bus de 32 bits.
MAC_ADR_O	O	Bus de direcciones hacia el bloque MAC. Bus de 32 bits
MAC_SEL_O	O	Select Output. Indica que bytes son válidos en el bus de datos. Debe permanecer en 1111b.
MAC_STB_O	O	Strobe output. Se utiliza para indicarle al bloque MAC que un ciclo válido da comienzo.
MAC_WE_O	O	Write output. Se le indica al bloque MAC si es un ciclo de escritura (‘1’) o de lectura(‘0’).
MAC_ACK_I	I	Acknowledge input. Entrada que nos permite saber cuando un ciclo normal termina en el bloque MAC.
MAC_CYC_O	O	Cycle output. Se utiliza para indicarle al bloque MAC que un ciclo válido está en progreso.
MAC_ERR_I	I	Error input. Indica que una finalización errónea de un ciclo.
MAC_RST_O	O	Reset output. Sirve para resetear el bloque MAC.
RAM_DAT_O	O	Salida de datos hacia la RAM. Bus de 32 bits.
RAM_DAT_I	I	Entrada de datos desde la RAM. Bus de 32 bits.
RAM_ADR_O	O	Bus de direcciones hacia la RAM. Bus de 32 bits.
RAM_STB_O	O	Strobe output. Señal para indicarle a la RAM que hay un ciclo en progreso.
RAM_WE_O	O	Write enable. Indica si es un ciclo de escritura (‘1’) o de lectura(‘0’).
RAM_ACK_I	I	Acknowledge input. Señal que nos indica que un ciclo de escritura o lectura a la RAM ha finalizado.
RELOJ	I	Reloj del bloque.

7.8.1. Ciclos de escritura/lectura a la RAM

Los procesos de escritura y lectura a la RAM son los necesarios para escribir los paquetes a ser enviados y para leer la información de los paquetes que se reciban desde el PHY.

Cada ciclo de lectura o escritura simple en la RAM dura dos ciclos de reloj. En el primer ciclo se debe tener la dirección de RAM fija en el bus de direcciones, en caso de que sea una escritura, también debe estar fijo el dato a escribir en el bus de datos. Si además levantamos la señal RAM_STB_O e indicamos en la señal RAM_WE_O si es escritura o lectura (‘1’ ó ‘0’), al llegar el flanco de reloj se escribirá o leerá un dato desde la RAM.

En el segundo ciclo se espera recibir la confirmación de la RAM a través de un ‘1’ en la señal RAM_ACK_I. De ser así se baja nuevamente la señal RAM_STB_O, y finaliza el ciclo. Ambos ciclos cumplen con la especificación Wishbone.

Los ciclos de escritura en ráfaga, como el que se ve en la figura 10, comienzan con una escritura simple, pero en el momento de recibir un ’1’ en la señal RAM_ACK_I (segundo ciclo de reloj), ya se tiene un nuevo dato y una nueva dirección en los buses respectivos, por lo que se mantiene la señal RAM_STB_O en ’1’, y comienza a escribirse un dato por ciclo de reloj.

Esto se mantiene hasta la última escritura de la transferencia, la cuál finaliza idéntica a una escritura simple tal como está estipulado por la especificación Wishbone.

Figura 10: Escritura en RAM

7.8.2. Ciclo de escritura/lectura al core MAC

Los procesos de escritura y lectura al core MAC son necesarios para la configuración de los buffer descriptors, los registros internos del core MAC en especial el registro moder, que como ya vimos es mediante su configuración que se da inicio a la transmisión y recepción de paquetes.

La duración de los ciclos de lectura o escritura al bloque MAC son variables según el registro al que se esté accediendo. Como ejemplo en la figura 11 se puede observar que la escritura de cada palabra de 32 bits del buffer descriptor dura 4 ciclos de reloj, y en cambio una escritura al registro moder (32 bits) dura solo 2 ciclos de reloj.

En el primer ciclo se debe tener la dirección de la MAC fija en el bus de direcciones, en caso de que sea una escritura, también debe estar fijo el dato a escribir en el bus de datos. Si además levantamos las señales MAC_STB_O y MAC_CYC_O e indicamos en la señal MAC_WE_O si es escritura o lectura ('1' ó '0'), al llegar el flanco de reloj se escribirá o leerá un dato desde el bloque MAC.

En los siguientes ciclos se espera a recibir la confirmación del bloque MAC a través de un '1' en la señal MAC_ACK_I. De ser así se bajan nuevamente las señales MAC_STB_O y MAC_CYC_O, y finaliza el ciclo.

Todos los ciclos de transferencia al core MAC cumplen con el protocolo Wishbone.

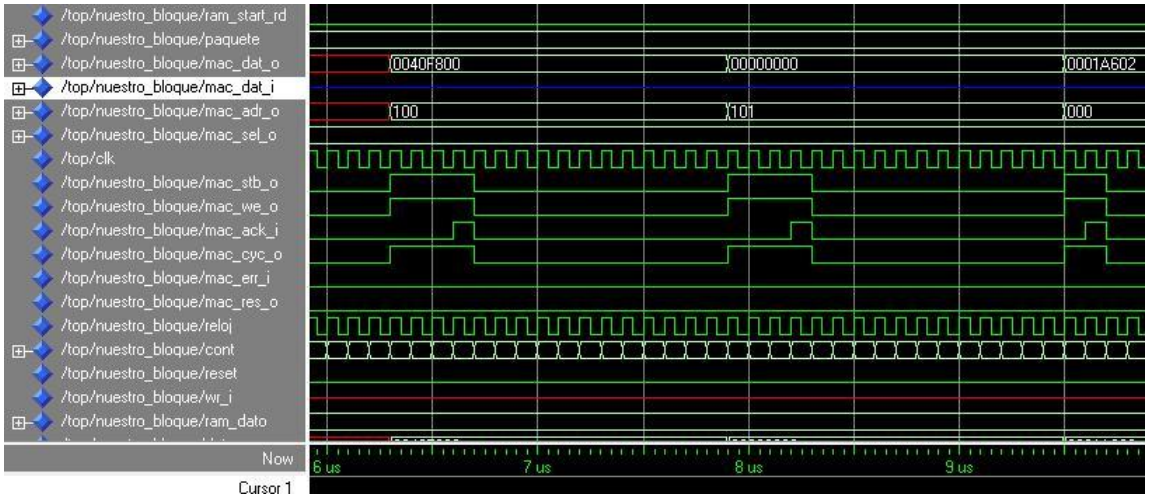


Figura 11: Escritura de un Buffer Descriptor

7.9. Bloque principal

Este bloque es el que agrupa todos los módulos vistos anteriormente, las conexiones entre ellos y por ende el que luego se sintetiza y se graba en el FPGA (ver figura 12).

Para la conexión de los bloques no surgieron mayores problemas salvo por la conexión de la RAM. La RAM tiene solamente una interfaz slave para que se la pueda manejar. Sin embargo, para el funcionamiento de nuestro diseño es necesario que la RAM sea manejada por el bloque MAC_eth_controller, y además por el core MAC, para la lectura y escritura de datos.

El bloque MAC_eth_controller es el encargado de manejar tanto la RAM como de generar los ciclos necesarios para la escritura o lectura de la RAM por parte del core MAC. Por lo tanto es quien debe estar encargado de determinar a quien corresponde el manejo de la RAM en cada momento.

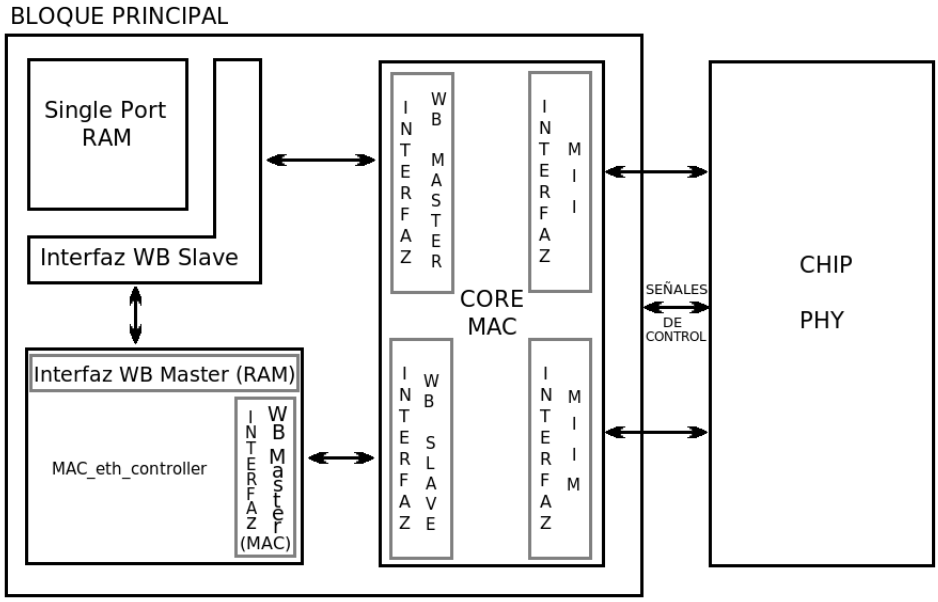


Figura 12: Arquitectura del Bloque Principal

Como solución a este problema se dispuso de una señal de salida en el bloque MAC_eth_controller, mediante la cual se indica si se quiere utilizar o no el bus de la RAM. En el bloque principal lo que se hizo fue multiplexar el bus de la RAM, utilizando como selector esta señal llamada SEL. Si SEL está en '1' se conecta la RAM a la interfaz Wishbone master del bloque MAC_eth_controller diseñada para ese propósito. De los contrario, las señales de la RAM son conectadas a la interfaz Wishbone master del core MAC.

Las únicas señales de entrada y salida que tiene este bloque son las necesarias para la conexión con el chip PHY, en este caso el chip LXT974. La mayoría de las señales de este chip son manejadas por el core MAC. Existen algunas señales de entrada que no están contempladas por este core, pero que sin embargo son parámetros que no nos interesa variar una vez que se fijan. Estos parámetros son fijados en este bloque.

7.9.1. Conexionado del Bloque Principal

En la tabla 8 se pueden observar las señales de entrada y salida del bloque principal, las cuales son conectadas al chip PHY, salvo por la entrada de

reloj global (CLK).

Tabla 8: Señales de entrada/salida del bloque principal

SEÑAL	**TIPO**	**DESCRIPCION**
CLK	I	Señal de reloj global.
MTX_CLK_PAD_I	I	Señal de reloj de transmisión de datos entregado por el chip PHY.
MTXD_PAD_O	O	Bus de ancho 4 bits, de transmisión de nibbles hacia el PHY.
MTXEN_PAD_O	O	Enable de transmisión de datos.
MTXERR_PAD_O	O	Señal para indicarle al chip PHY si hubo un error (en la subcapa MAC) en la transmisión de datos.
MRX_CLK_PAD_I	I	Señal de reloj de recepción de datos entregado por el chip PHY.
MRXD_PAD_I	I	Bus de ancho 4 bits, de recepción de nibbles desde el PHY.
MRXDV_PAD_I	I	Señal que indica la recepción de un dato válido desde el PHY.
MRXERR_PAD_I	I	Señal que indica si hubo un error la recepción de un dato desde el PHY.
MCOLL_PAD_I	I	Señal que indica si hubo una colisión.
M_RST_N_O	O	Señal activa por nivel bajo, para resetear la configuración del chip PHY.
MD_PAD_IO	I/O	Señal bidireccional para la comunicación serial entre el chip PHY y el bloque principal. Es la señal de datos para el manejo de la interfaz MII.
MDC_PAD_O	O	Señal de reloj entregada por el bloque principal al chip PHY para el manejo de la interfaz MII. Con esta señal se sincroniza MD_PAD_IO.
MDDIS	O	Señal que activa o desactiva el control de la interfaz MII. Cuando se activa (por nivel alto) la señal MD_PAD_IO queda solo como entrada, o sea un canal de solo lectura.
AUTOENA	O	Señal que habilita o deshabilita la autonegociación en los puertos Ethernet.
FDE	O	Cuando se activa por nivel alto, habilita el modo full-duplex en todos los puertos Ethernet.
CFG	O	Bus de tres bits que permiten una configuración global del chip PHY.
PHY_ADD	O	Bus de direcciones de 3 bits ADD(4..2). Los bits 0 y 1 se setean internamente en el chip PHY. Cada combinación indica uno de los 4 puertos
PWRDN	O	Power Down, activa por nivel alto.
TRSTE	O	Bus de 4 bits (uno por puerto). Cuando se activa alguno de los bits, el puerto correspondiente se aisla de la interfaz MII de datos.

7.9.2. Señales de reset y reloj

El core MAC cuenta con una entrada de reset (RST_I) en su interfaz Wishbone slave. En la implementación del bloque principal, esta señal es conectada a una señal de salida del bloque MAC_eth_controller llamada MAC_RST_O, ya que el reset del core MAC se maneja desde este bloque. Existe además una señal de reset de salida en el bloque principal, que se utiliza para resetear la configuración del chip PHY.

El bloque principal tiene como entrada tres señales de reloj. Dos de ellas son específicas y son entregadas por el chip PHY (mtx_clk_i & mrx_clk_i). Estas señales son las que sincronizan la transmisión y recepción de datos entre el bloque principal y el chip. Ambas tienen una frecuencia de 25 MHz. El funcionamiento de estos dos submódulos de transferencia de datos entre el bloque principal y el chip PHY es por lo tanto independiente del resto de los procesos del bloque.

La restante es una señal de reloj global (CLK). Esta señal dentro del bloque principal está conectada a la señal RELOJ del bloque MAC_eth_controller, a la señal WB_CLK_I del core MAC y a la señal CLK_I de la RAM. Mediante esta señal de reloj se sincronizan todas las señales y procesos síncronos dentro del bloque principal, salvo por las transferencias desde y hacia el chip PHY mencionadas en el párrafo anterior.

Al momento de programar el FPGA con este diseño, las señales mtx_clk_i y mrx_clk_i deben ser conectadas a sus pares en el chip PHY. El reloj global puede ser conectado a cualquiera de los relojes disponibles en la placa sin problemas. Durante las pruebas se intentó conectarlo a relojes de distinta frecuencia con resultados exitosos, llegando a una frecuencia máxima de 50 MHz (según los resultados de la síntesis se puede utilizar una frecuencia aún mayor).

8. Simulación

El diseño de nuestra primera versión de este core fue culminada por lo menos un mes antes de poder tener acceso a las placas PgVirtex. Durante la etapa de diseño habíamos utilizado bastante como herramienta de ayuda el testbench hecho por quienes diseñaron el core MAC que utilizamos en nuestro diseño. Como se vio anteriormente, este testbench proponía la

simulación de un chip PHY similar al que tienen las placas PgVirtex.

Por lo tanto, y puesto que nos vimos estancados hasta no tener las placas, decidimos realizar nuestro propio testbench utilizando esta implementación del chip PHY. De esta forma, si bien no simularíamos exactamente la prueba que íbamos a realizar, si nos serviría para corregir algunos errores de diseño, y por lo menos nos permitiría ver el comportamiento del chip implementado frente a los procesos de transferencia de datos.

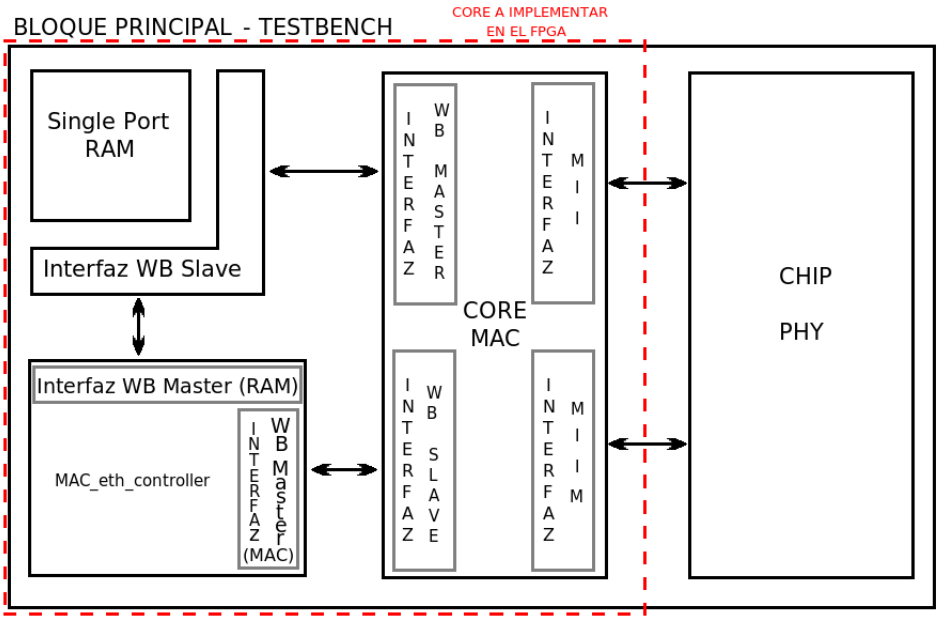


Figura 13: Arquitectura del testbench

Para el diseño de nuestro testbench, se tomó el bloque principal y se le agregó el módulo PHY (implementa el chip LXT971A) mencionado anteriormente. Nos quedó entonces una versión del bloque principal pero con las entradas y salidas correspondientes al chip LXT974 como señales internas conectadas al módulo PHY, como se indica en la figura 13.

Para la transferencia de paquetes, tanto para la simulación como para la prueba del core diseñado, decidimos realizar los siguientes pasos:

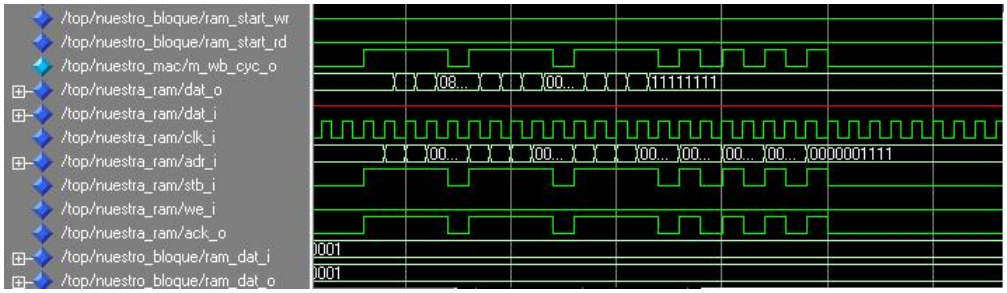


Figura 14: Lectura de la RAM desde el core MAC (acceso DMA)

1. Grabar un paquete UDP en la RAM. Hay que tener en cuenta que la utilidad de la prueba no es manejar protocolos complicados sino verificar el correcto funcionamiento del diseño realizado. Da lo mismo el tipo de paquete que se envíe, ya que es solo para realizar pruebas. Este diseño por si solo no es de mayor utilidad, y el alcance del mismo es que pueda tomar cualquier paquete, agregarle el encabezado Ethernet y enviarlo a través cualquiera de los puertos.

2. Configurar un buffer descriptor, y el registro TX-BD-NUM del core MAC.

3. Configurar el registro moder, de manera que se habilite el envío del paquete almacenado en la RAM, y determinado por el buffer decriptor previamente configurado.

El paquete UDP grabado en la RAM fue creado con un software de licencia freeware llamado Colasoft Packet Builder. Esta herramienta permite escoger un protocolo, en este caso UDP, y convierte los datos ingresados por el usuario en cada campo, a hexadecimal, calculando además el CRC de los datos del protocolo (el CRC propio de los datos del protocolo, que es distinto al CRC de Ethernet generado por el core MAC).

Durante las simulaciones realizadas con ModelSim, corregimos muchos errores de diseño, y errores en el funcionamiento del core por no haber comprendido correctamente como configurar alguno de los registros del core MAC.

Luego de muchos intentos logramos obtener una versión en la cual podíamos apreciar como en el bus de datos de entrada del módulo PHY aparecía el paquete escrito por el bloque MAC_eth_controller en la RAM. Esta versión del bloque principal (aunque sin el módulo PHY), sería la que utilizaríamos para realizar las pruebas una vez que pudieramos disponer de las placas.

Después de lograr las primeras pruebas exitosas con la transmisión de paquetes, nos propusimos modificar los procesos de prueba para lograr recibir y enviar el mismo paquete, o sea realizar un echo request[17]. Para ello los pasos a seguir eran los siguientes:

1. Cofigurar un buffer descriptor de recepción (es necesario escribir también el registro TX_BD_NUM) adecuadamente para la recepción de un paquete. Además se deben configurar los registros de interrupción para que luego de recibir un paquete el core MAC genere una interrupción.

2. Configurar el moder de manera que quede habilitada la recepción de paquetes. A partir de este momento queda esperando por la recepción de un paquete.

3. Utilizando el software Packet Builder de Colasoft, crear un paquete UDP, indicando el puerto 7 como destino (Echo Request).

4. Luego de tener conectados el puerto Ethernet de la PC donde se está corriendo el software con un puerto Ethernet de la placa PgVirtex, enviar el paquete creado (Packet Builder dispone de esta opción).

5. Luego que el paquete de echo request es recibido y guardado en RAM (de acuerdo a la configuración del buffer descriptor), se genera una interrupción.

6. Debido a la interrupción, se debe mirar la dirección a donde apunta y el largo del buffer descriptor de recepción, y crear uno de transmisión que permita enviar el paquete recibido.

7. Configurar nuevamente el registro moder para habilitar la transmisión del paquete indicado por el buffer descriptor de transmisión.

[17] El protocolo de un paquete de echo request es UDP, pero se envía a un puerto específico.

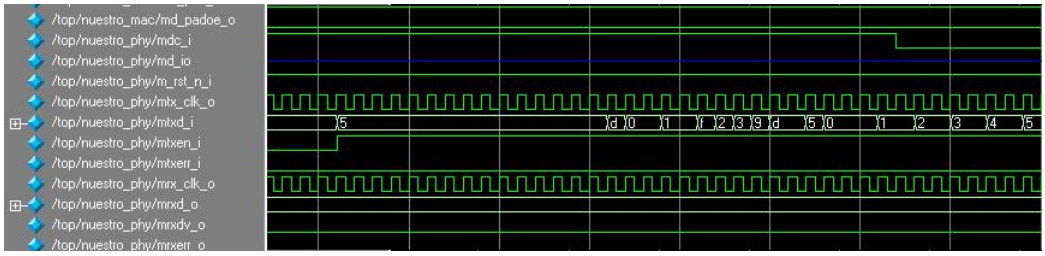


Figura 15: Envío de paquete desde el core MAC al PHY (inicio)

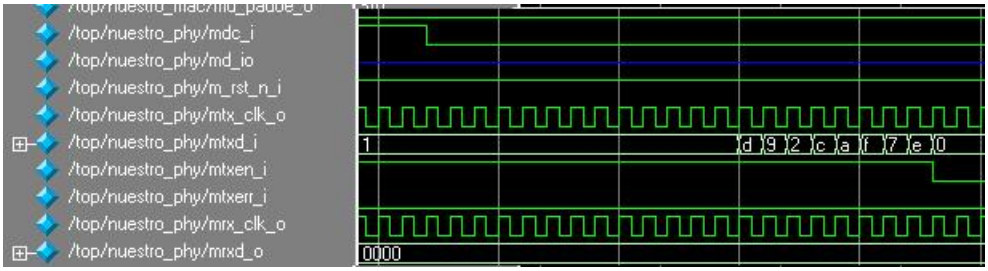


Figura 16: Envío de paquete desde el core MAC al PHY (fin)

8. Observar la recepción del mismo paquete enviado por el Packet Builder usando un software de análisis de redes.

9. Síntesis y pruebas

Los problemas mayores comenzaron durante la síntesis del core que íbamos a programar en la placa. Nuestro core tiene 3 entradas de reloj. Una de ellas es el reloj general del core, que es el utilizado en el core MAC, el bloque MAC_eth_controller y en la RAM.
Las otras dos entradas de reloj se deben conectar al chip PHY. El chip proporciona las dos señales de reloj utilizadas para sincronizar la transmisión y recepción de datos entre el chip PHY y el core MAC.

A la hora de realizar la síntesis, ésta nos devolvía un error, ya que estas dos señales son reconocidas como señales de reloj pero no están asignadas a ninguno de los pines de reloj global (Global Clock Buffer) del FPGA. El problema era que los pines de reloj global disponibles ya están físicamente conectados en la placa a distintos relojes que tiene disponibles, lo

que hacía imposible que pudieramos a la vez asignar estas señales a relojes globales y conectarlas a las salidas de reloj del chip PHY.

Como primer intento por solucionar este problema, intentamos latchear las señales de reloj a la entrada del bloque principal de manera de "engañar" al software de síntesis, y que éste no tomara estas señales como si fueran de reloj. De lograr esto nos hubiera permitido asignarle cualquier pin de uso general sin problemas. Pero no fue así, siguió exigiendo que conectáramos esas señales a alguno de los pines de reloj global.

Leyendo la hoja de datos del FPGA, encontramos que este permite la posibilidad de que cierta cantidad de señales sean mapeadas como líneas con poco retardo (low skew lines). Esta propiedad está especialmente especificada en los FPGAs para algunos tipos de señales en particular, como relojes ó señales con alto fan-out[18]. Sin embargo, no encontramos información acerca de como configurar o asignarle esta propiedad a las señales.

Como segundo intento, y al no tener otra solución a nuestro alcance, quisimos asignarle uno de los relojes ya conectados en la placa a estas señales, cuya frecuencia era igual a la de salida de los relojes del chip PHY. Si bien la frecuencia era la misma éramos concientes de que no podíamos asegurar nada acerca del defasaje que podía existir entre un reloj y otro. Pudimos sintetizar el diseño correctamente y programarlo en el FPGA, pero al realizar las pruebas vimos que en realidad era utópico pensar que esta solución podía llegar a funcionar.

Recorriendo varios foros en Internet, pudimos encontrar que la forma de asignar las señales de reloj para que sean mapeadas como relojes locales es mediante una directiva VHDL en el código. Lo que se hace mediante esta directiva (llamada "ibuf") es indicarle a la herramienta de síntesis que queremos que mapée estas señales utilizando líneas de poco retardo, y como ya interpretaba que eran señales de reloj, es por ello que las define como relojes locales. Con esto solucionamos el problema de los relojes, y pudimos finalmente comenzar a probar realmente.

Para la prueba, se conectó un patchcord (cable de red) desde una de las bocas de red al puerto Ethernet de una PC. La prueba que se hizo fue la misma que la descrita en la sección anterior, solo que se realizó iterativamente, enviando el paquete UDP a una razón de una vez por segundo. En

[18] señales con mucha carga de salida

la PC se puso en funcionamiento un software de análisis de redes (Wireshark), el cuál permanece escuchando el canal y desplegando en pantalla la información de los paquetes que llegan. En estas circunstancias daríamos la prueba como efectiva si vemos desplegarse en pantalla una vez por segundo el mismo paquete que el escrito en RAM para ser enviado.

Durante las primeras pruebas luego de observar en pantalla los paquetes enviados al establecerse el enlace físico entre los puertos Ethernet, no observábamos más paquetes. Sin embargo, en la placa veíamos que el led correspondiente a la transmisión de datos a través del puerto si se encendía una vez por segundo, lo cual nos daba pruebas de que algo se estaba transmitiendo, aunque evidentemente no era lo que esperábamos.

Posiblemente los datos que se estaban transmitiendo eran filtrados antes de llegar a ser "vistos" por el software de análisis de redes debido a la malformación de los mismos.

Después de varias hipótesis que nos planteamos, encontramos que en la simulación, el envío de datos que se hace de a nibbles hacia el chip PHY se realizaba tomando los datos de la RAM como si estuvieran guardados usando "big endian" (nibble bajo – dirección baja, nibble alto – dirección alta), y nosotros lo estábamos guardando suponiendo que trabajaba como "little endian".

Este error ocurrió debido a que al observar en el bus los datos que se transfieren desde la subcapa MAC al chip PHY durante las simulaciones, notábamos que el orden de los datos no era el mismo con el que se guardaban en la RAM. Sin embargo, no tuvimos en cuenta que esto era simplemente durante esta etapa de la transmisión y que luego los nibbles eran reordenados en forma correcta antes de ser enviados al medio físico.

Una vez corregido esto, logramos una prueba satisfactoria observando la llegada de los paquetes UDP una vez por segundo.

Parte II

Ethernet con Leon y Linux

En esta etapa se implementó un sistema SOC basado en el microprocesador Leon de Gaisler Research. A este sistema se le configuró para ser capaz de manejar un puerto Ethernet y se compiló un kernel de linux compatible con los requerimientos de hardware del sistema. Al sistema operativo se le dio la capacidad de manejar el protocolo TCP/IP para poder resolver el pasaje de datos entre nuestro SOC y un PC a través de un puerto Ethernet.

10. Processador LEON2

El modelo VHDL de LEON2 implementa un procesador de 32 bits de acuerdo a la arquitectura SPARC V8[7]. SPARC[19] es una arquitectura de set de instrucciones 1 para un microprocesador RISC2. La arquitectura fue diseñada por Sun Microsystems en 1985; más tarde, y con el objetivo de promover la arquitectura, se creó la organización SPARC International. SPARC International liberó el estándar, que al día de hoy es totalmente abierto y no propietario.

10.1. Arquitectura

LEON2 cuenta con las siguientes características on-chip:

- Cache separado para instrucciones y datos.
- Multiplicadores y divisores hardware.
- Controlador de interrupciones.
- Unidad de debugging con buffer de seguimiento de instrucciones
- Dos timers 24-bit.
- Dos UARTs.
- Función power-down.
- Watchdog
- Puerto 16-bit I/O
- Controlador de memoria
- Interfaces Ethernet MAC y PCI.

[19] Scalable Processor ARChitecture

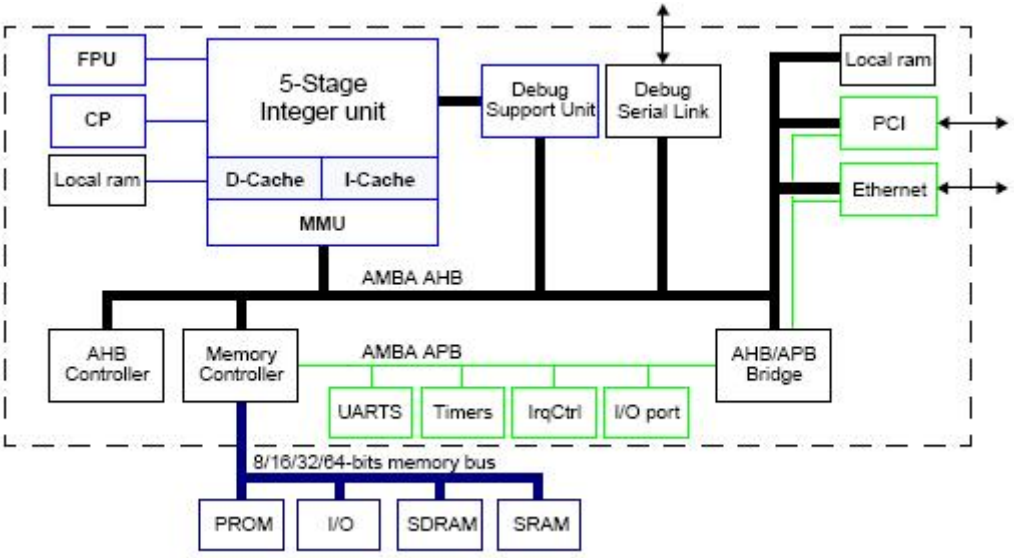


Figura 17: Arquitectura del microprocesador LEON2

Un diagrama de bloques de la arquitectura Leon se puede ver en la figura 17. A continuación se enumerarán y describirán brevemente los distintos bloques.

Integer Unit

"Esta unidad implementa el estándar SPARC V8 completo, incluyendo todas las instrucciones de multiplicación y división. El número de ventanas de registros es configurable dentro del límite de 2-32 definido por el estándar. Para ayudar al debugging de software hasta 4 watchpoints pueden ser configurados. Cada registro puede interrumpir en una instrucción o dirección arbitraria. Si la unidad de debugging esta habilitada, los watchpoints pueden ser usados para entrar en modo debug.

Unidad de FP y Co-procesador

El modelo de LEON provee una interfaz para las unidades de FP, GRFPU (Gaisler), Meiko FPU core (Sun Microsystems) y LTH FPU. Además provee una interfaz genérica de co-procesador para interfacear un co-procesador implementado por el usuario.

Caché

Provee cache multi-set de instrucciones y datos por separado, configurables de acuerdo a:

- 1-4 sets
- 1 - 64 kbyte/set
- 16 - 32 bytes por línea

Una RAM local puede ser adherida al caché de instrucciones, permitiendo acceso 0-waitstates sin necesidad de reescribir los datos a memoria externa.

MMU

La MMU[20] es opcional, y esta implementada de acuerdo a la referencia SPARC V8. Su habilitación permite la posibilidad del uso de algún sistema operativo como Linux o Solaris.

DSU

La DSU[21] es opcional y permite debuggear el hardware de destino. Esta unidad de debugging permite la inserción de breakpoints y watchpoints, además de acceso a todos los registros desde un debugger remoto. Un "trace buffer"permite el seguimiento de las instrucciones ejecutadas así como del tráfico a través del bus AHB[22].

Interfaz de memoria

La interfaz de memoria provee una interfaz directa de PROM, mapeo de dispositivos de I/O, static RAM y SDRAM. El ancho de palabra puede programarse a 8,16,32,64 bits.

[20] Memory Management Unit
[21] Debug Support Unit
[22] Advanced High-performand Bus

UART

Dos UART's de 8 bits son provistas on-chip. El baud-rate es programable y los datos son enviados en frames de 8 bits con un bit de parada. Opcionalmente un bit de paridad puede ser generado.

Controlador de interrupciones

Este controlador maneja un total de 15 interrupciones. Cada una puede ser programada con uno de dos niveles de prioridad. También esta disponible un controlador secundario para hasta 32 interrupciones adicionales.

Puerto I/O paralelo

Provee un puerto paralelo de 32 bits para entrada salida. 16 bits están siempre disponibles para ser usados como entrada o salida, los restantes 16 bits solo están disponibles si la memoria se configura para operar con 8 o 16 bits. Algunos de los bits se pueden alternar entre entrada/salida e interrupciones externas.

Bus AMBA on-chip

LEON2 cuenta con una implementación completa de los buses AMBA AHB y APB. Un esquema flexible de configuración permite agregar nuevos IP cores. LEON usa el bus AHB para conectar los controladores de caché al controlador de memoria y opcionalmente a alguna otra unidad de alta velocidad. Por defecto el único master en el bus es el procesador, mientras que se proporcionan dos slaves, el controlador de memoria y el APB bridge. En la figura 18 se muestra la distribución del espacio de direcciones del bus AHB.

Controlador de Memoria

El bus de memoria externa es controlado por un controlador de memoria programable. El controlador actúa como slave en el bus AHB. El funcionamiento del controlador es programado a través de 3 registros de

Address range	Size	Mapping	Module
0x00000000 - 0x1FFFFFFF 0x20000000 - 0x3FFFFFFF 0x40000000 - 0x7FFFFFFF	512 M 512 M 1 G	Prom Memory bus I/O SRAM and/or SDRAM	Memory controller
0x80000000 - 0x8FFFFFFF	256 M	On-chip registers	APB bridge
0x90000000 - 0x9FFFFFFF	256 M	Debug support unit	DSU
0xB0000000 - 0xB001FFFF	128 K	Ethernet MAC registers	Ethernet

Figura 18: Espacio de direcciones AHB[7]

configuración (MCR1, MRC2, MRC3), en el primero se programa el timing de la ROM y accesos I/O, el segundo controla el funcionamiento de la SRAM y SDRAM donde se especifica que chips de memoria están habilitados (SRAM, SDRAM), y parámetros como el tamaño de las filas, columnas y bancos de la SDRAM; en el último se programan los tiempos de refresco de la SDRAM. El bus de memoria también puede ser configurado en modo 8 ó 16 bits para aplicaciones que requieran escasa memoria y requerimientos de alta performance. El controlador decodifica un espacio de memoria de 2 GB, dividido de acuerdo a la figura 19.

Address range	Size	Mapping
0x00000000 - 0x1FFFFFFF	512 M	Prom
0x20000000 - 0x3FFFFFFF	512M	I/O
0x40000000 -0x7FFFFFFF	1 G	SRAM/SDRAM

Figura 19: Mapeo de direcciones del controlador de memoria[7]

SDRAM Access

El acceso a SDRAM es soportado para 2 bancos de dispositivos compatibles con PC100/PC133. El controlador soporta dispositivos de 64M, 256M y 512M, de 8-12 column-address bits, hasta 13 row-address bits y 4 bancos. El tamaño de los bancos es programable. La operación del controlador SDRAM es controlada vía los registros de configuración de memoria. La SDRAM tiene buses de direcciones y control separados. Las señales de datos se pueden usar a través de un bus común o vía un bus

de datos dedicado. En este último caso el ancho de datos de la memoria puede ser de 32 o 64 bits. Los 2 bancos de memoria se pueden mapear a partir de las direcciones 0x40000000 ó 0x60000000 dependiendo de si la SRAM está habilitada o no."[4][7]

10.2. Configuración

La configuración se puede realizar a través de una interfaz gráfica realizada en TCL/TK ó simplemente utilizando el terminal de Linux. Para configurar el procesador utilizando la interfaz gráfica se debe de teclear "*make xconfig*" estando ubicados en el directorio Leon. Luego de elegir las opciones según nos conviene y de salvar los cambios se genera el archivo de configuración *device.vhd*.

11. Antecedentes

El proyecto de fin de carrera CicloP[4] realizado por Leonardo Etcheverry, Juan Manuel Horta y Sergio Nan consistió entre otras cosas en desarrollar un co-procesador genérico para LEON2 donde uno podía integrar instrucciones propias realizadas en vhdl. Como resultado central se destaca el logro de una plataforma genérica para codesarrollo software/hardware, constituida por una configuración totalmente sintetizable y de funcionamiento probado sobre la placa PgVirtex del microprocesador LEON2, junto a una interconexión física y lógica con una unidad de coprocesamiento genérica. Esto permite a un usuario del sistema, diseñar sus propios bloques de hardware (instrucciones) e integrarlos de manera sencilla al coprocesador. Es decir, el proyecto Cicplop permite agregar instrucciones en el coprocesador que sean soluciones de un problema en particular, por lo que este es un coprocesador "full custom". Como tarea extra, los integrantes de CicloP integraron Snapgear Linux a su sistema y lo mostraron en la defensa del proyecto. Sin embargo al no formar parte de los objetivos iniciales del proyecto no realizaron documentación al respecto, por lo tanto para la implementación de este sistema operativo sobre el LEON2 en este proyecto se comenzó de cero.

El segundo antecedente al respecto es el del proyecto Algoritmo Morfológico de Procesamiento de Imágenes con el procesador LEON2 de la

materia Diseño Lógico 2[23]. Este proyecto se basó en el sistema CicloP para crear instrucciones especiales que permitan hacer operaciones de algoritmos morfológicos por Hardware.

12. Snapgear Linux para LEON

Los creadores de Leon dan soporte de una versión especial de Snapgear para Leon. Snapgear es una distribución de Linux especialmente diseñada para microprocesadores embebidos que contiene kernel, librerías y códigos de aplicación para un rápido desarrollo de sistemas con Linux embebido.

Existen dos versiones de Snapgear para Leon: uClinux (linux-2.0.x) para sistemas sin MMU y linux-2.6.21.1 para sistemas con MMU. Ambas versiones se pueden utilizar tanto con LEON2 como con LEON3. El kernel de Linux se puede configurar utilizando una interfaz gráfica, o simplemente mediante el terminal de windows.

Soporte de hardware de linux-2.0.x:

- non-MMU
- GRETH 10/100/1000 Ethernet
- SMC91x 10/100 Ethernet
- OpenCores 10/100 Ethernet
- PS/2 (GRPS2)
- Text VGA (APBVGA)
- APBUART y GRTIMER

Soporte de harware de linux-2.6.21.1:

- MMU
- LEON3 SMP
- GRETH 10/100/1000 Ethernet
- SMC91x 10/100 Ethernet
- OpenCores 10/100 Ethernet
- PCI (GRPCI)
- GRETH sobre PCI

[23]Relizado por dos de los integrantes de este proyecto, Alejandro Romio Ravazzani y Rodrigo Taborda, y por Antonio Sena

- ATA DMA y non-DMA (ATACTRL)
- Host USB 1.1 y/o 2.0 (GRUSBHC)
- PS/2 (GRPS2)
- SVGA Framebuffer (GRVGA)
- Text VGA (APBVGA)
- I2C (I2CMST)
- SPI (SPICTRL)
- APBUART y GRTIMER

13. Elección del tipo de procesador

Antes de compilar el kernel de Linux se debe conocer exactamente sobre que hardware se trabajará. Además, ambas configuraciones, la del Procesador y la del sistema operativo deben ser compatibles para que el sistema funcione correctamente.

Una de las decisiones que se tuvo que tomar fue la de elegir el procesador, es decir si utilizar el LEON2 ó el LEON3.

LEON3 es la versión más nueva de estos procesadores. Forma parte de GRLIB que es un IP Library que contiene diversos IP cores diseñados para desarrollo de *System On Chip*. Entre ellos se destacan controladores de PS/2, VGA, USB 2.0, CAN 2.0 y Ethernet. Al igual que el LEON2, estos cores se interconectan entre si mediante el bus AMBA.

GRLIB permite implementar sistemas de multiprocesador. Se pueden colocar hasta cuatro procesadores en paralelo, lo que deja en evidencia por qué se ha introducido el concepto GRLIB y no simplemente se lo llama LEON3 como en el anterior procesador.

El procesador que se eligió para el sistema a implementar es el LEON2. Este es el que se adapta mejor a los requerimientos de hardware del diseño, que son los de contar con un controlador Ethernet, puerto serie y controladora de SDRAM.

El controlador Ethernet que está integrado en el LEON2 es el 10/100 Mbit Opencores MAC mientras que el utilizado por LEON3 es el GRETH Ethernet. Esta fue también una de las razones que nos hizo elegir el LEON2 ya que a esa altura del proyecto conocíamos a fondo el funcionamiento

del core MAC utilizado por el LEON2.

La placa PG-VIRTEX cuenta con un puerto USB. La posible integración de éste a nuestro sistema nos obligaría a elegir el LEON3 ya que este es el único que puede manejar uno. Sin embargo finalmente se decidió no integrar USB al sistema por lo que se decidió utilizar LEON2.

14. Configuración del LEON

La configuración del LEON2 se realiza a través de una interfaz gráfica hecha en TCL/TK 8.4. A continuación se describirán las opciones de configuración más importantes.

Opciones de Sintesis:

Lo más importante que se debe elegir en esta parte es el tipo de FPGA con el cual trabajaremos, en nuestro caso es VIRTEX_E. Además se debe especificar si queremos que nuestro sistema cuente con una ROM, dentro del FPGA.

Opciones de reloj:

Aquí debemos especificar si queremos utilizar el reloj de PCI como reloj del sistema y además si queremos utilizar el macro virtex clkdll para manejar el reloj.

Opciones de Procesador:

Las opciones mas importantes que hay que especificar son:

- Capacidad de manejar instrucciones de multiplicación y división.
- Habilitar o no la unidad de punto flotante.
- Habilitar o no el coprocesador.
- Opciones de caché, tamaño, etc.
- Si queremos un sistema con MMU o no. Esta opción es muy importante ya que nos define el tipo de Linux que utilizaremos, es decir si la versión 2.0 que es para sistemas sin MMU ó la versión 2.6.

- Unidad de Debugging. Debemos tener la unidad de Debugging habilitada para poder manejar el procesador con grmon.
- Elección de soporte para memoria RAM estática de diferentes anchos de palabra, y lo mas importante habilitar el controlador de memoria dinámica, es decir que contamos con un controlador de SDRAM intefgrado en FPGA.
- Periféricos. Habilitar o no RAM interna, Watchdog, registro de configuración del LEON, registro de Status del bus AMBA, PCI y controladora de Ethernet.

Opciones de booteo:

Aquí debemos especificar el reloj del sistema en MHz; desde donde se booteará, si desde la ROM interna o desde memoria RAM, y el baud rate del puerto serie Uart 0. Esta última opción es importante ya que la comunicación de la unidad de debugging con Grmon es a través de esta interfaz.

El archivo de configuración *device.vhd* se puede ver en la figura 20. A cada constante se le asignan los valores elegidos por el usuario, por ejemplo vemos que en la constante "*constant peri_config*"la opción *ethen* esta seteada a TRUE lo que quiere decir que esta habilitado el controlador de Ethernet.

15. Compilación y simulación de Linux

Al igual que el LEON2, la configuración de Snapgear Linux se realiza a través de una interfaz gráfica hecha en TCL/TK. La configuración del Linux es mas extensa y compleja que la del LEON, se deben de setear una gran cantidad de opciones de configuración que van desde aspectos generales del sistema, opciones del kernel como lo son drivers para los diferentes hardware y opciones de usuario como librerías, comandos, etc.

En esta documentación no se entrará en detalle sobre todas las opciones elegidas en la configuración, solo se especificarán las opciones de aspectos generales del sistema y alguna más.

De aquí en mas cada vez que hagamos referencia a linux se estará mencionando la versión *uclinux 2.0* para sistemas sin MMU.

device.vhd

```
  dsnoop => none, drfast => false, dwfast => false, dlram => false,
  dlramsize => 1, dlramaddr => 16#8F#);

constant mmu_config : mmu_config_type := (
  enable => 0, itlbnum => 8, dtlbnum => 8, tlb_type => combinedtlb,
  tlb_rep => replruarray, tlb_diag => false );

constant ahbrange_config  : ahbslv_addr_type :=
      (0,0,0,0,0,0,4,0,1,2,7,5,7,7,7,7);

constant ahb_config : ahb_config_type := ( masters => 4, defmst => 0,
  split => false, testmod => false);

constant mctrl_config : mctrl_config_type := (
  bus8en => false, bus16en => false, wendfb => false, ramsel5 => false,
  sdramen => true, sdinvclk => true, sdsepbus => false,
  sd64 => false);

constant peri_config : peri_config_type := (
  cfgreg => true, ahbstat => false, wprot => false, wdog => false,
  irq2en => false, ahbram => true, ahbrambits => 11, ethen => true );

constant debug_config : debug_config_type := ( enable => true, uart => false,
  iureg => false, fpureg => false, nohalt => false, pclow => 2,
  dsuenable => true, dsutrace => true, dsumixed => true,
  dsudpram => false, tracelines => 256);

constant boot_config : boot_config_type := (boot => prom, ramrws => 0,
  ramwws => 0, sysclk => 25000000, baud => 9600, extbaud => false,
  pabits => 8);
```

Figura 20: Archivo de configuración de LEON2

15.1. Aspectos generales del sistema

Como vemos en las figuras 21, 22 y 23 se debe especificar:

- el tipo de procesador, si LEON2 ó LEON3.
- si el procesador tiene unidad de punto flotante y soporta instrucciónes de multiplicación y división.
- el reloj del sistema.
- describir la memoria que tiene el sistema (ROM, SRAM y SDRAM).
- aspectos del puerto serie *ttyS0* que es el puerto por donde veremos el terminal de Linux a través de un terminal 232.

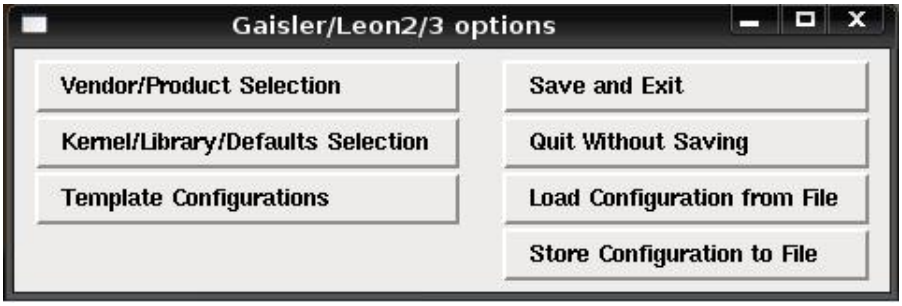


Figura 21: Configuración de Linux

Figura 22: Elección del procesador

Opciones de compilación del Kernel

No entraremos en detalle acerca de las opciones de configuración del Kernel. Solo mencionaremos lo que consideramos mas importante para nuestro proyecto que es la habilitación de simulación del linux a través del simulador Tsim; dar soporte para Ethernet; en particular para el Opencores MAC y habilitar TCP/IP networking.

Opciones de Usuario

Como se puede ver en las figuras 25 y 26, las opciones de usuario van desde elección de librerías, aplicaciones, comandos de red y hasta juegos. Por ejemplo se puede habilitar el comando *ifconfig* para poder manejar la interfaz Ethernet eth0.

Figura 23: Aspectos generales del sistema.

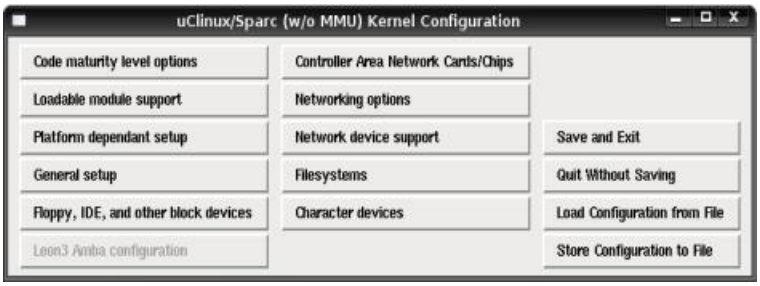


Figura 24: Configuración del Kernel

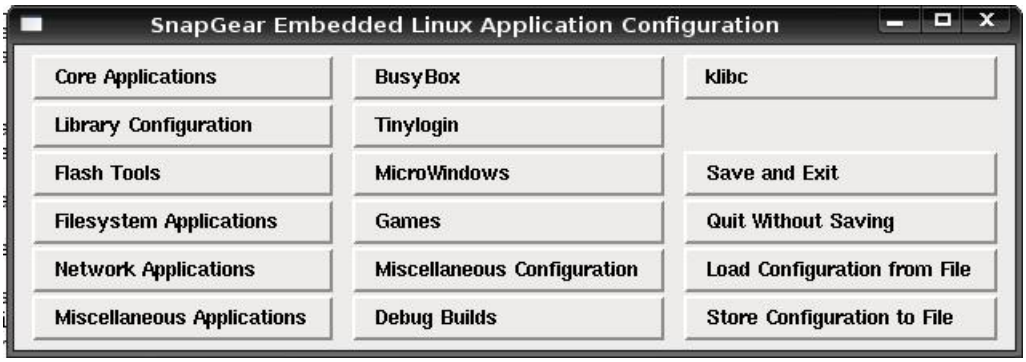


Figura 25: Configuraciones de usuario

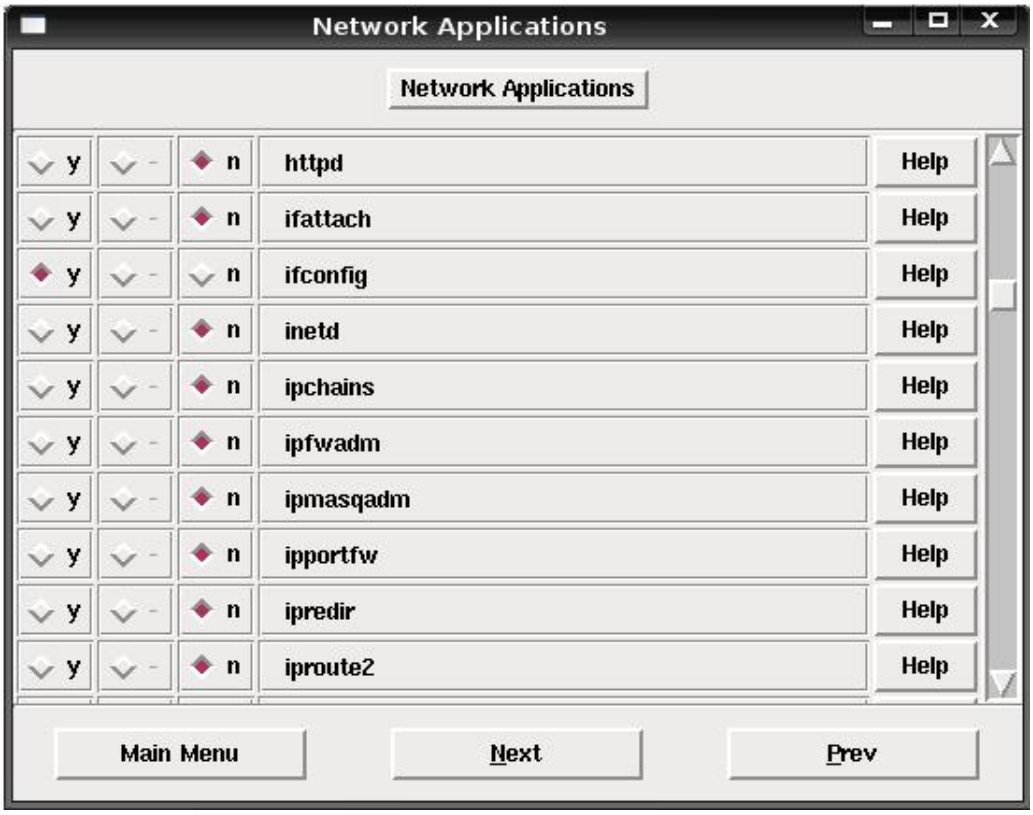


Figura 26: Elección del comando ifconfig

16. Implementación del sistema

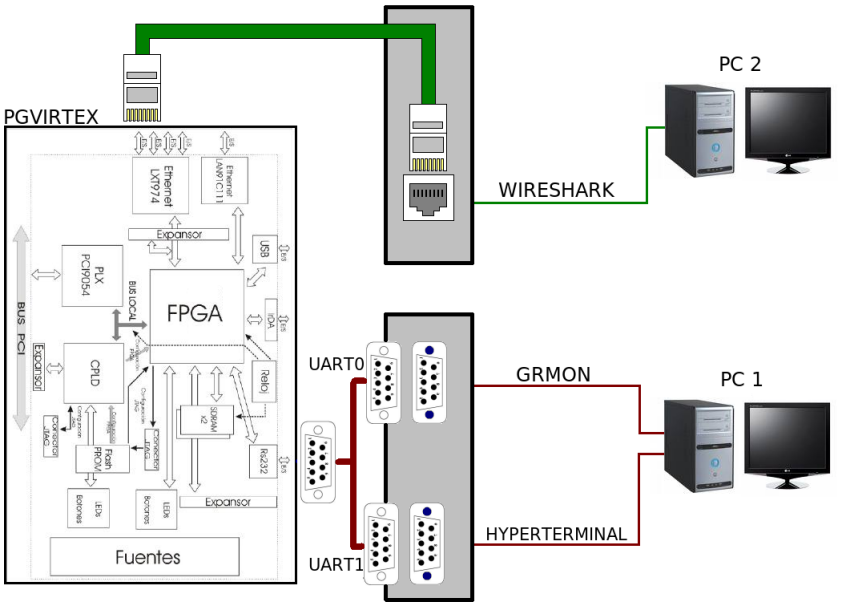


Figura 27: Prueba de Sistema LEON - Linux con Ethernet

En la figura 27 se presenta el sistema completo armado para realizar las pruebas. El de la figura corresponde a las últimas pruebas, cuando se tenía el sistema armado con todas las características requeridas. Sin embargo, se realizaron pruebas previas con más restricciones en el sistema. Igualmente la figura sirve para ilustrar alguna de estas pruebas previas.

Para el proceso de implementación del sistema nos propusimos trabajar de la siguiente manera:

1. Configuramos un LEON lo más simple posible, sin habilitar el controlador Ethernet. Lo sintetizamos, lo programamos en el FPGA y realizamos las pruebas de comunicación con Grmon.
2. Compilamos una imagen de Linux sin soporte para redes, lo simulamos con Tsim, lo cargamos en memoria con Grmon y hacemos correr el sistema operativo sobre el procesador.
3. Configuramos el LEON completo, es decir con el controlador Ethernet habilitado, lo probamos con Grmon y le corremos el Linux anteriormente configurado.

4. Compilamos una nueva imagen de Linux con soporte para redes, en especial con el comando *ping* activado, lo simulamos con Tsim, lo cargamos en memoria, lo corremos y luego verificamos que los paquetes ICMP lleguen a una PC correctamente.

16.1. Síntesis del LEON sin Ethernet, prueba con grmon.

El sistema implementado para esta parte tiene entre otras las siguientes características:

- Controladora de Ethernet deshabilitada.
- Controladora de SDRAM habilitada.
- Coprocesador y FPU deshabilitados.
- Reloj del sistema en 25 MHz.
- Unidad de Debugging habilitada (DSU).
- Dos UARTS habilitadas.

La síntesis del sistema se realizó con el programa Simplify y la asignacion de pines, "mapping" y "place and route" se realizaron con el software de Xilinx ISE 6.0.

Luego de creado el archivo de programación se procedió a programar el FPGA. A este sistema que cuenta con un procesador LEON2 configurado como se muestra más arriba, se le carga una imagen de linux. Para realizar pruebas de funcionamiento de este sistema se conectó un cable serie bifurcado con conectores DB9, entre dos puertos serie de una PC y el puerto serie RS232 de la placa PgVirtex. Cada terminal de la bifurcación del cable corresponde a cada una de las UARTS habilitadas en el procesador. Una de las UARTS se utiliza para la comunicación con la placa a través de Grmon, para cargar en memoria el sistema operativo. La otra UART es la que se utiliza para la comunicación del sistema operativo con la interfaz al usuario (línea de comandos en una consola).

El sistema descripto es igual al que se puede ver en la figura 27, salvo porque este no tiene la conexión Ethernet entre la placa y una PC.

A continuación se muestra en la figura 28 el sistema conectado a través de Grmon. El comando *infosys* muestra el hardware del sistema y sus correspondientes direcciones del bus AMBA. Además se pueden ver los

registros internos del procesador, desplegar el contenido de la memoria ó cargar la memoria entre otras cosas.

```
initialising .............
detected frequency:  25 MHz

Component                            Vendor
LEON2 Memory Controller              European Space Agency
LEON2 AHB Status & Failing Addr      European Space Agency
LEON2 SPARC V8 processor             European Space Agency
LEON2 Write Protection               European Space Agency
LEON2 Configuration register         European Space Agency
LEON2 Timer Unit                     European Space Agency
LEON2 UART                           European Space Agency
LEON2 UART                           European Space Agency
LEON2 Interrupt Ctrl                 European Space Agency
LEON2 I/O port                       European Space Agency
AHB Debug UART                       Gaisler Research
LEON2 Debug Support Unit             Gaisler Research

Use command 'info sys' to print a detailed report of attached cores

grlib> info sys
00.04:00f   European Space Agency  LEON2 Memory Controller (ver 0)
             ahb: 00000000 - 20000000
             ahb: 20000000 - 40000000
             ahb: 40000000 - 80000000
             apb: 80000000 - 80000010
             64-bit prom @ 0x00000000
             32-bit sdram: 1 * 16 Mbyte @ 0x40000000, col 8, cas 2, ref 7.7 us
01.04:017   European Space Agency  LEON2 AHB Status & Failing Addr (ver 0)
             apb: 8000000c - 80000014
02.04:002   European Space Agency  LEON2 SPARC V8 processor (ver 0)
             apb: 80000014 - 80000018
03.04:018   European Space Agency  LEON2 Write Protection (ver 0)
             apb: 8000001c - 80000020
04.04:008   European Space Agency  LEON2 Configuration register (ver 0)
             apb: 80000024 - 80000028
             val: 6077bf00
05.04:006   European Space Agency  LEON2 Timer Unit (ver 0)
             apb: 80000040 - 80000070
06.04:007   European Space Agency  LEON2 UART (ver 0)
             apb: 80000070 - 80000080
             baud rate 38400
07.04:007   European Space Agency  LEON2 UART (ver 0)
             apb: 80000080 - 80000090
```

Figura 28: Comunicación con el sistema a través de Grmon

Luego de configurado y compilado el Linux, cargamos la imagen en memoria con Grmon y corrimos el sistema operativo (ver figura 28). Si bien este Linux no dispone de muchos comandos, por lo menos se pueden recorrer los directorios del sistema de archivos con el comando *cd* y ver los contenidos de las carpetas con el comando *ls*.

16.2. Síntesis del LEON con Ethernet, verificación con wireshark

Para esta parte sí habilitamos el controlador de Ethernet del sistema y le dimos al Linux el soporte para redes, en particular habilitamos TCP/IP y algunos comandos como *ifconfig* para manejar la interfaz eth0 y *ping* para poder enviar paquetes ICMP hacia un PC.

Para esta parte el sistema montado es igual al que se muestra en la fig 27. Al igual que en la prueba anterior, se utiliza un cable serie bifurcado para la comunicación con las dos UARTs habilitadas en el procesador. Nuevamente se utiliza una para la carga de la imagen de linux con Grmon, y la otra para la visualizar la línea de comandos del sistema operativo corriendo en el procesador.

Lo que se agrega en este sistema es la comunicación entre otra PC (se podría utilizar la misma que para la comunicación serie) y la placa PgVirtex mediante un patchcord entre dos puertos Ethernet.

Corrimos el sistema operativo sobre el procesador y se verificó el correcto funcionamiento del mismo. "Pingueamos" la interfaz de la PC y "snifeamos" los paquetes con WireShark en la misma. En la figura 29 se puede ver la ejecución desde la línea de comandos del Snapgear.

```
sh 26: Child 27 diedaß
Please press Enter to activate this console.

Sash command shell (version 1.1.1)
/> /bin/ping 192.168.0.15
PING 192.168.0.15 (192.168.0.15): 56 data bytes
64 bytes from 192.168.0.15: icmp_seq=0 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=1 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=2 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=3 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=4 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=5 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=6 ttl=128 time=10.0 ms
64 bytes from 192.168.0.15: icmp_seq=7 ttl=128 time=10.0 ms
-
```

Figura 29: Ejecución del comando ping desde Snapgear

Parte III

PCI

17. Estudio de alternativas

Llegado a este punto del proyecto se evaluaron algunas posibilidades dentro de lo que era la motivación y el objetivo de nuestro proyecto.

17.1. Manejo del puerto USB

Una de las alternativas era intentar manejar otro periférico como el puerto USB existente en la placa, con el LEON. Esta alternativa no era la más atractiva desde el punto de vista de la utilidad que podía llegar a tener por varias razones. En primer lugar el chip USB existente en la placa, el CP2417 de Philips no es de los más conocidos en el mercado, y es un poco viejo. Además, si bien nos interesaba esta opción, se decidió tomar otro camino, ya que este nos hubiera apartado un poco de la línea de nuestro proyecto.

17.2. PCI con LEON2

Otra opción planteada era intentar manejar el puerto PCI existente en la placa utilizando el LEON2. El LEON2 tiene entre sus características la posibilidad de configurarle un core diseñado para el manejo de un puerto PCI. El mayor problema existente respecto a esto es que ya existe en la placa un adaptador del puerto PCI que entrega del lado del FPGA una interfaz particular. El procesador Leon2 tiene integrado un controlador de bus PCI pero este se conecta directamente al bus PCI y es utilizado únicamente para debugging o sea para la comunicación entre grmon y la DSU.

Debido a esto no sabíamos de antemano si sería sencillo e incluso viable la implementación de un core para manejar esta interfaz con el procesador. Para esto no solo tendríamos que hacer un core que maneje el bus PCI y compatible con AMBA, sino que aparentemente (si no encontramos un driver de Linux para PLX) también tendríamos que diseñar un driver e integrarlo al kernel de linux

17.3. PCI - Ethernet

La otra alternativa planteada fue lograr manejar el puerto PCI pero desde un core sobre el FPGA. Recordemos que en este caso no se estaría manejando el puerto PCI directamente, sino que la interfaz brindada por el controlador de bus PCI de PLX implementado en la placa. Como antecedente teníamos que el grupo de proyecto PgVirtex ya había diseñado un core para poder programar el FPGA a través del puerto PCI.

Además también surgió la posibilidad de continuar con la idea del manejo de los puertos Ethernet de la primera etapa. Recordemos que el core diseñado, no tenía mucha utilidad por si solo ya que mas allá de que se manejaran los puertos Ethernet, no se realizaba ningun procesamiento de datos en el FPGA, o algoinput que justificara la transferencia de datos a través de estos puertos.

La idea entonces, era poder enviar datos a la placa a través del puerto PCI, más precisamente a la RAM implementada en el FPGA por el core diseñado anteriormente, para que fueran luego enviados a través de los puertos Ethernet. Finalmente se optó por esta última alternativa.

18. Estudio del Chip PCI

Para el manejo del puerto PCI existe en la placa un chip que fue agregado a la placa por el grupo del proyecto PgVirtex con el propósito de brindar al usuario de la placa una interfáz mas amigable que la interfáz PCI para programar el FPGA y el pasaje de datos.[8]. El chip es un adaptador de bus PCI, PCI 9054 de la compañía PLX technologies, compatible con la versión del bus PCI 2.2, de 32 bits y 33 MHz[11].

Tiene una interfaz de bus local diseñada para soportar la conexión directa de algunos dispositivos de Motorola, Intel e IBM. Además tiene un bus local programable, que funciona a una frecuencia máxima de 50 MHz y soporta buses de direcciones y datos de 32 bits, no multiplexados, de 32 bits multiplexados, y accesos como esclavo de dispositivos con buses de 8, 16 ó 32 bits.

Cuenta además con la posibilidad de agregar una EEPROM[24] serial op-

[24] Electrically-Erasable Programmable Read-Only Memory

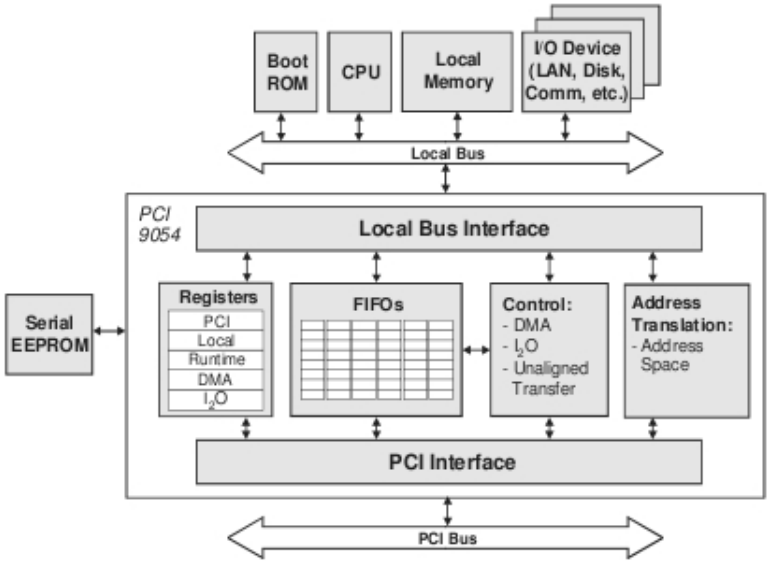


Figura 30: Arquitectura del chip PCI 9054[11]

cional, que puede ser utilizada para cargar información. Esto es muy útil para cargar información única para un adaptador en particular, como ID de proveedor, o ID de red. En la placa también se incluyó por parte del grupo del proyecto PgVirtex esta EEPROM opcional, que permite guardar la configuración de los registros del chip. De esta manera cada vez que se conecta la placa no es necesario configurarla, ya que se carga la información desde la EEPROM, por más que el chip sea reseteado.

El chip puede ser utilizado con el bus PCI como iniciador (master) ó target (slave), disponiéndose en ambos casos de 3 espacios de memoria (espacio 0, espacio 1 y espacio de expansión de ROM) a los cuales se puede acceder con cualquier bus PCI master.

Para la utilización del bus local el chip opera en tres modos diferentes:

- Modo M: utiliza bus de direcciones y datos de 32 bits no multiplexados. Este modo se utiliza para conexión directa con disositivos MPC680 y MPC850 de Motorola.
- Modo C: utiliza bus de direcciones y datos de 32 bits no multiplexados.
- Modo J: utiliza bus de direcciones y datos de 32 bits multiplexados.

Los modos C y J utilizan cuatro tipos de transferencia de datos posibles:

- Configuración de registros de acceso
- Operación con el bus PCI como master
- Operación con el bus PCI como salve
- Operación DMA

A partir de la familiarización con el protocolo de salida hacia el bus local, debimos evaluar la posibilidad de crear un core que fuese compatible con nuestro diseño anterior. Nuestro diseño anterior tenía interfaces Wishbone tanto para los accesos a la memoria RAM como para los accesos al core MAC. Por lo tanto necesitábamos la manera de lograr que el protocolo de bus local fuese compatible con Wishbone para poder grabar los datos en la RAM.

19. Utilización del CPLD

El CPLD (ver figura 31) fue colocado en la placa por el grupo de proyecto PgVirtex, de manera de tener un core grabado permanente en la placa para la configuración del FPGA (el core grabado en el CPLD no se pierde por más que la placa no esté alimentada).

El core existente para el CPLD toma las señales desde el bus local del chip PCI 9054 y es capaz de programar el FPGA. Además debía actuar como arbitro del bus local, aunque hubo que realizar algunas modificaciones para su correcto funcionamiento.

20. Software necesario

20.1. PLXmon

Este software es utilizado para acceder a los registros de configuración del PCI 9054 . Mediante este programa se configura la EEPROM existente junto al chip. La información incluida en la EEPROM es la que luego se carga a los registros que configuran los espacios de memoria y las direcciones de los buses PCI y local para luego poder manejar adecuadamente las señales con los cores, tanto del CPLD como del FPGA.

Este software es brindado por el fabricante del PCI 9054, y permite además observar el estado, tanto de la EEPROM, como de los registros internos del chip necesarios para la configuración de los buses PCI y local.

Mediante el PLXmon, configuramos la EEPROM, y por ende el chip PCI 9054, de acuerdo con las recomendaciones hechas en la documentación del proyecto PgVirtex. Esta configuración nos permitiría, en primera instancia programar nuestro core al FPGA, y luego que el core programado tomara control sobre el bus local, nos permitiría enviar datos (paquetes) hacia la placa, para luego ser grabados en la RAM implementada en nuestro diseño.

20.2. Windriver

Este software funciona como herramienta para la creación del driver capaz de manejar el chip PCI. Permite además escanear los buses PCI para ver qué dispositivos se encuentran conectados a la PC y que espacios de memoria reservan.

Ya existía un ejecutable que permite grabar archivos al bus PCI, diseñado por los creadores de Windriver y modificado por el grupo de proyecto PgVirtex para enviar el archivo de programación del FPGA. Estudiamos el archivo fuente de este ejecutable hecho en lenguaje C, para ver que modificaciones se podían efectuar para lograr que nos sirviera para nuestro cometido.

Nuestro propósito era enviar paquetes ya armados dentro de un archivo, por lo que en escencia la función que iba a cumplir el software en nuestra aplicación sería muy similar. No fue necesario modificar este ejecutable, ya que por como está implementado permite variantes a la hora de realizar una transferencia. Permite distintos anchos de palabra (8, 16 ó 32 bits), escritura y lectura hacia cualquiera de los espacios de memoria del bus local.

21. Problemas Encontrados

21.1. Problemas de Software

A la hora de intentar probar lo que ya estaba hecho (programar la placa), nos encontramos con diversos problemas de software, principalmente porque en la documentación existente no se especificaba nada acerca de los sistemas operativos utilizados para la utilización de los softwares mencionados.

21.1.1. Problemas con PLXmon

Debido a nuestro desconocimiento y a la falta de documentación al respecto perdimos bastante tiempo intentando realizar pruebas sobre un sistema operativo Windows XP, cuando en realidad la versión que teníamos del software fue diseñado antes de la aparición de este sistema operativo. Buscamos una versión más reciente de este software, pero la que encontramos no tenía todas las funcionalidades que necesitábamos para nuestro propósito.

Debimos recurrir entonces a un sistema operativo más antiguo (Windows 98), sobre el cual el software si funcionó correctamente.

21.1.2. Problemas con el Windriver

Al utilizar las opciones indicadas en el manual de la placa PgVirtex para poder grabar un archivo para la programación del FPGA no obtuvimos el resultado esperado. Si bien seguíamos los pasos tal cual se indicaba, intentando acceder a la dirección 0x00 del bus PCI para grabar allí el archivo, el software en cuestión en algunos casos nos devolvía un mensaje de error indicando que estábamos intentando acceder a una dirección no permitida y en otros casos simplemente colgaba el sistema operativo.

Según nuestro criterio los pasos indicados eran los adecuados para la correcta programación del FPGA a través del PCI, y como obteníamos errores de acceso a memoria, nuestras sospechas apuntaron a algún posible problema de permisos de paginado del sistema operativo. Realizamos muchas pruebas en distintas máquinas y con distintos sistemas operativos

sin obtener ninguna prueba satisfactoria.

Al no encontrar la verdadera naturaleza de nuestro problema, decidimos buscar algún camino alternativo al indicado en la documentación existente. Luego de varias pruebas más logramos transferir datos hacia la placa a través del bus PCI pero indicando al software de programación directamente una dirección del bus local en vez de una dirección del bus PCI.

Cabe recalcar que estos problemas inesperados de software, así como los de hardware explicados más adelante nos insumieron mucho tiempo en su resolución, lo cual repercutió bastante en la planificación de acuerdo al calendario.

21.2. Problemas de hardware

Nos vimos ante algunos problemas de hardware por lo que tuvimos que probar en varias PCs hasta encontrar una en la cual el chip PCI de la placa funcionara correctamente.

La placa tiene dos problemas a la hora de conectarla a un slot PCI:

- Los contactos de la placa no se encuentran dorados por lo que la conductividad de sus contactos no es la mejor.
- la placa no cuenta con un soporte mecánico que permita mantenerla firme al estar conectada lo cual impide una buena conexión en algunos casos.

Además, existieron problemas de compatibilidad entre la placa como dispositivo PCI y otros periféricos como tarjetas de video, lo cual creaba conflictos que hacía imposible el funcionamiento de la PC.

Finalmente conseguimos una PC donde la placa funciona correctamente y en armonía con el resto del hardware. Igualmente nos pareció oprtuno dejar claros estos detalles, ya que a la hora de trabajar con la placa conectada a un puerto PCI hay que tener en cuenta que no funciona correctamente en cualquier máquina.

22. Modificación del diseño inicial

22.1. Introducción

Una vez que logramos programar el FPGA por medio del puerto PCI, nos vimos en condiciones de probar las modificaciones de nuestro core que una vez grabado en el FPGA debía tomar el control del bus local del chip PCI 9054, tomar los paquetes ingresados a través del PCI y almacenarlos en RAM. Una vez en la RAM, el core ya diseñado se encargaría de enviarlos a través de uno de los puertos Ethernet.

Durante las pruebas nos dimos cuenta de que el core del CPLD no funcionaba correctamente y que para poder lograr nuestro cometido tendríamos que modificarlo. Si bien el core lograba la programación del FPGA, al tratar de programar la placa por segunda vez éste colgaba el sistema operativo. Además este core no era capáz de distinguir entre los distintos espacios de memoria, o sea que no era compatible con el pasaje de datos para uso general, solo para programación. Todos los datos que recibía los interpretaba como datos de programación del FPGA.

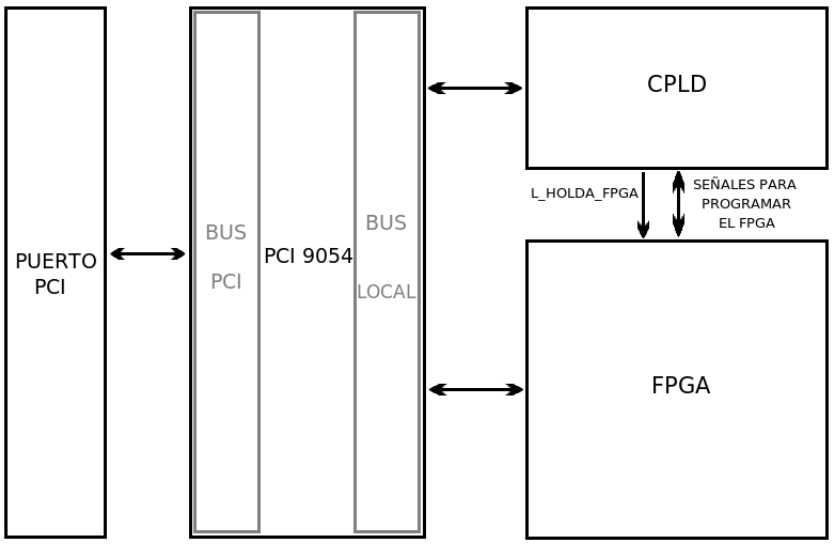


Figura 31: Conexión PCI 9054 - CPLD - FPGA

22.2. Compatibilidad de protocolos

El core que habíamos diseñado, incluía una RAM con interfaz Wishbone, por lo cual la lógica a diseñar que conectara el bus local con nuestro bloque debía ser capaz de recibir datos del bus local y escribirlos en RAM mediante la interfáz Wishbone.

En principio se realizó un conexionado muy sencillo (mientras todavía estábamos solucionando los inconvenientes de hardware y software encontrados), un adaptador entre el bus local y Wishbone, el cual suponíamos que funcionaría parcialmente y deberíamos ajustar detalles para respetar los tiempos del bus local. Sin embargo, al comenzar con las pruebas, no obtuvimos resultados muy auspiciosos que nos hicieran seguir por este camino.

Fue allí que se nos ocurrió tomar como ejemplo el ciclo de escritura del core diseñado por el grupo PgVirtex para grabar en el CPLD. En principio nuestro ciclo de escritura a la RAM desde el bus local debía ser similar a este.

22.3. Modificación del bloque principal

Las modificaciones realizadas al bloque principal incluyeron una nueva interfaz para poder comunicarse con el bus local del chip PCI 9054, una señal de control conectada al CPLD, que actúa como árbitro del bus local, además del proceso necesario para escribir datos desde el bus local a la RAM.

22.3.1. Interfaz hacia el bus local y CPLD

Tabla 9: Señales de interfaz bloque principal - bus local/CPLD

SEÑAL	ANCHO	TIPO (I/O)	DESCRIPCION
LA	32	I	Bus de direcciones
LD	32	I	Bus de datos
LHOLD_PLX	1	I	Señal de entrada que indica al core cuando el chip PLX necesita el bus local.
LHOLDA_PLX	1	O	Señal de salida mediante la cual se indica al chip PLX cuando puede tomar el control del bus local.
ADS_N	1	I	Address Strobe. Señal de entrada que indica cuando las direcciones en el bus son válidas.
BLAST_N	1	I	Señal de entrada que indica cuando culmina una transacción en ráfaga.
LBE_N	4	I	Señal de entrada que en el caso de un bus de 32 bits indica que byte está activo durante una transferencia.
LW_R_N	1	I	Señal de entrada que indica si un ciclo es de escritura ('1') o lectura ('0').
READY_N	1	O	Señal de salida para indicar el estado de una transferencia en proceso. Debe permanecer en '0' durante todo el ciclo de transferencia y ser llevada a '1' al finalizar la misma.
LHOLDA_FPGA	1	I	Señal de entrada mediante la cual el CPLD indica cuando se puede tomar o no posesión del bus local.

22.3.2. Proceso de escritura de datos

El proceso de escritura de datos desde el bus local hacia la RAM está basado en una máquina de estados.

El core del FPGA permanece en un estado esperando que se solicite el bus local por parte del chip PLX(ver figura 32), mediante la señal LHOLD_PLX. Una vez que le llega el pedido del bus, lo cede a través de la señal LHOLDA_PLX, y pasa al siguiente estado. En este estado queda esperando por el comienzo del ciclo de transferencia de datos, o sea que le llegue un pulso en la señal ADS_N que indique que en el bus de direcciones hay datos válidos. En este estado, al llegar el pulso en ADS_N se latchean las direcciones a un registro.

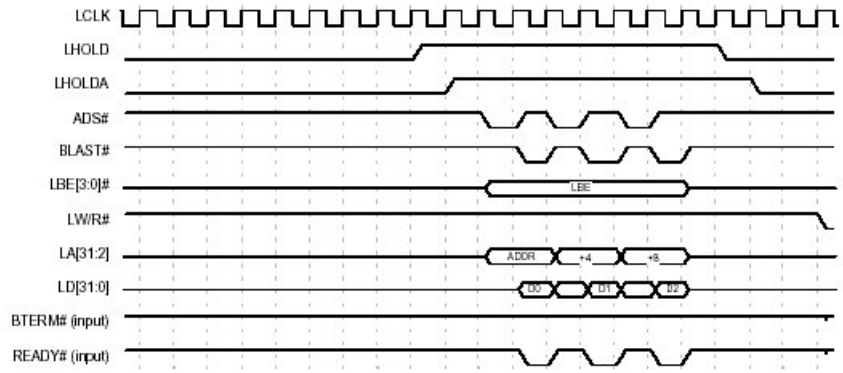


Figura 32: Ciclo de escritura de cuatro bytes del bus local[11]

Se pasa al siguiente estado en el cual se espera un tiempo determinado para respetar los tiempos de hold y setup de la señal READY_N la cuál se lleva a '0'. En el siguiente estado comienza con la escritura de los datos.

La escritura, se realiza en ráfaga y de a bytes. Debido al funcionamiento del software encargado de ello (Windriver), cada escritura en ráfaga es solo de 4 bytes. Por lo tanto en cada escritura tendremos un dato para escribir en la RAM (de ancho 32 bits), cada 4 ciclos de reloj. Durante estos ciclos de reloj se permanece en el mismo estado.

Para poder escribir los datos en la RAM correctamente es necesario concatenar los 4 bytes de cada escritura en ráfaga y formar el double-word a ser grabado. Para ello, lo que se hizo fue crear un shift register de 4 registros, de ancho 8 bits en el que se va grabando cada byte de la escritura, una vez por ciclo de reloj. Una vez que se completan los 4 bytes de la escritura, queda en el shift register la palabra de 32 bits para ser escrita en la RAM.

Al siguiente estado se accede una vez que finaliza la escritura en ráfaga. En este estado se escribe en la RAM el dato recibido y se verifica si aún quedan datos por escribir, lo cual se comprueba mediante el sensado de la señal LHOLD_FPGA. Si aún quedan datos por escribir, se vuelve al estado en el que se espera por la señal ADS_N. De lo contrario, se da por terminada la transferencia y se va a un estado "idle", esperando por un nuevo llamado desde el bus PCI.

22.4. Modificación de core de programación del FPGA

Por como estaba diseñado el core que se graba en el CPLD para la programación del FPGA, debía grabar el archivo de programación y luego dejar el bus local disponible. Debía actuar como árbitro del bus. Sin embargo, nos encontramos con que había partes del código comentadas seguramente debido a que este no funcionaba correctamente. Pudimos advertir que una vez que programaba el FPGA, el core quedaba en un "idle loop" pero no dejaba libre el bus local para ser utilizado por otro core.

Decidimos entonces realizar alguna modificación a este core ya diseñado, y poder lograr que una vez que se programara el FPGA, el CPLD quedara si en un "idle loop" pero que permitiera al core implementado en el FPGA tomar control del bus local. A su vez el core del FPGA cuando se va a programar la placa.

22.5. Árbitro del bus

En nuestro diseño debían coexistir dos cores que manejaran el bus local, uno en el CPLD y otro en el FPGA. Las interfaces hacia el bus local de ambos cores permanecerían conectadas en todo momento, por lo cual

cuando cada uno de los cores no estuviera utilizando el bus debería dejar las señales de su interfaz en tercer estado.

Afortunadamente para nuestros intereses existían en la placa un par de pines de entrada/salida que se encontraban conectados entre el CPLD y el FPGA (físicamente), uno de las cuales utilizamos como árbitro del bus en el CPLD, y su par en el FPGA para informarle al mismo cuando el bus local está disponible o no. La señal utilizada para el control del bus local es LHOLDA_FPGA, la cual es manejada por el core del CPLD. Esto es razonable teniendo en cuenta que el core en el FPGA es volátil. Lo mas lógico es que quien tiene el control sobre el bus sea el CPLD, que está programado en todo momento.

22.6. Reloj utilizado

El chip PCI está dispuesto en la placa para utilizar un reloj fijo de 33 Mhz. Este reloj es independiente del reloj del bus local así como del utilizado en el FPGA. Esto es posible gracias a que el chip PCI 9054 tiene varios FIFOs que utiliza para almacenar datos de ser necesario. El reloj de entrada tanto del bus local como el de nuestro core es configurable en la placa mediante unos jumpers teniendo la posibilidad de llevarlo desde 7,372 Mhz a 58,976 Mhz.

En un principio la placa estaba configurada para trabajar a la máxima frecuencia posible. Durante las pruebas observamos que la ram no quedaba bien grabada, e incluso los valores que se grababan realmente diferían de una prueba a otra con los mismos datos de entrada, lo que nos llevó a pensar que estabamos teniendo problemas con el reloj. Efectivamente al mirar la especificación del chip PCI 9054 observamos que si bien se podía trabajar con una frecuencia diferente a 33MHz en el bus local, ésta estaba limitada a 50 Mhz como máximo para un funcionamiento correcto.

Fijamos la frecuencia a un valor un poco inferior a los 50 Mhz y aún teníamos problemas de tiempo en la grabación de la RAM. Igualmente, debido a la variada posibidad de selección que nos permitían los jumpers pudimos llevar la frecuencia hasta lograr un correcto funcionamiento de los ciclos de escritura en RAM. La implementación final se realizó a una frecuencia de aproximadamente 22 Mhz.

23. Pruebas

Para probar el correcto desempeño del core diseñado, se buscó realizar pruebas similares a las que se hicieron para probar el core diseñado en la primera etapa del proyecto. La principal contra fue que en este caso no contábamos con herramientas para poder simular las puebas. La idea era poder enviar un archivo que contenga un paquete UDP utilizando el software Windriver.

Para ello se debían seguir los siguientes pasos:

1. grabar en el CPLD, el core modificado por nosotros, que se utiliza para programar el FPGA. El CPLD se graba en la placa a través del conector JTag.
2. grabar en el FPGA el core diseñado, a través del puerto PCI. Para programar el FPGA se utiliza el software Windriver. Una vez que se programe el FPGA, debido a las modificaciones hechas en el core que se graba en el CPLD, éste entrega el control del puerto local del chip PCI 9054. El CPLD tomará el control del bus nuevamente, recién cuando se de un pulso de reset con uno de los pulsadores de la placa.
3. se graba el archivo que contiene los datos a ser enviados por el puerto Ethernet (en este caso un paquete UDP) utilizando nuevamente el Windriver. En este caso el core diseñado toma los datos que se le envían en el archivo y los graba en la RAM. Luego le indica al core MAC que envíe el paquete almacenado en RAM tal como se vio en la parte 1 de esta documentación.

La prueba en sí misma de enviar paquetes a través del puerto PCI, a la RAM, para luego ser tomados por la MAC y ser enviados a través de uno de los puertos Ethernet, no parece ser muy interesante desde el punto de vista funcional. Esta prueba fue realizada solamente para probar el funcionamiento de nuestro diseño.

Sin embargo abre la posibilidad, por ejemplo, de correr un algoritmo de procesamiento de datos luego que se recibió un paquete a través del puerto PCI, para luego si ser enviado a través de un puerto Ethernet. De esta forma, todo el procesamiento y el envío de datos se realizarían en paralelo a cualquier tarea que se pudiera estar haciendo en la PC donde este conectada la placa.

Este tipo de aplicaciones no fue realizada en nuestro proyecto, ya que no agregaría nada al objetivo particular de esta etapa del mismo. Igualmente para poder realizar este tipo de implementaciones, bastaría realizar una pequeña modificación en el código VHDL de nuestro diseño, para permitir correr un algoritmo antes de habilitar el envío de paquetes Ethernet.

Parte IV
Conclusiones generales

Como primera conclusión podemos decir que hemos cumplido con los objetivos planteados al iniciar este proyecto.

Se logró implementar en FPGA un sistema de transferencia de datos a través de puertos Ethernet, cumpliendo con las especificaciones de este protocolo tanto para capa física como para la subcapa MAC.

Además se realizó una aplicación simple de pasaje de datos a través del bus PCI, integrando esta aplicación al sistema anteriormente mencionado.

Por otra parte, implementamos un SOC compuesto de un procesador LEON2 de arquitectura SPARC, y un sistema operativo Linux corriendo sobre él, con la capacidad de conectarse a una red local a través de un puerto Ethernet.

Si bien cumplimos con los objetivos planteados, nos hubiera gustado profundizar en algunos aspectos. Para el sistema sin microprocesador, haber realizado una aplicación un poco más atractiva del punto de vista funcional como por ejemplo el diseño de un hub.

Esta opción no estaba alejada de nuestras posibilidades, ya que esto consistía en utilizar el sistema ya diseñado para cada boca del hub y desarrollar una lógica que se encargue de controlarlos adecuadamente.

En lo que refiere al SOC implementado, el espectro de posibilidades de avanzar en el tema es bastante amplio. A manera de ejemplo podemos nombrar la implementación de un servidor web, telnet, FTP, ATAoE, que permitirían resolver la transferencia de datos entre el sistema y un PC remoto.

Tanto en este caso como en el anterior, esto se nos vio imposibilitado por un tema de tiempos. Cabe recalcar que el desarrollo del proyecto nos llevó un mes más de lo planificado inicialmente.

Si bien en este aspecto no quedamos muy conformes, es de resaltar que muchas de las tareas que realizamos no fueron nada sencillas, más aún cuando no se tiene la experiencia necesaria en algunas áreas. Tal es el caso de la compilación de un kernel de Linux. También es interesante puntualizar que se dedicaron muchas horas a la investigación y que integrar a un diseño propio sistemas desarrollados por terceros, no es tarea sencilla y es algo que se presenta cotidianamente en la vida profesional del ingeniero.

Desde el punto de vista de la planificación, podemos decir que fue aceptable. Alguna de las tareas nos llevaron más tiempo que el planificado, pero las causas de estos retrasos en su mayoría fueron previstas en el análisis de riesgos, y las replanificaciones realizadas nos permitieron culminar satisfactoriamente el proyecto.

Referencias

[1] *FPGAs*.
URL http://es.wikipedia.org/wiki/FPGA

[2] *Protocolo Ethernet*.
URL http://es.wikipedia.org/wiki/Ethernet

[3] Cisco Systems, Inc.: *Internetworking Technology Handbook*.
URL http://www.cisco.com

[4] Etcheverry, Leonardo, Horta, Juan y Nan, Sergio: *CicloP*. Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, Montevideo, Uruguay, 2007.

[5] Fernández, Sebastián y Mondueri, Ciro: *Especificación Wishbone*. Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, Montevideo, Uruguay.

[6] Gaisler, Jiri y Isomaki, Marko: *GRETH 10/100 Mbit Ethernet MAC*. Gaisler Research, 2006.
URL http://www.gaisler.com

[7] Gaisler Research: *LEON2 XST User's Manual*, 2005.
URL http://www.gaisler.com

[8] Gómez, Silvia, Saporiti, Jimena y Villavedra, Agustín: *PGVIRTEX PCI - BOARD*. Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, Montevideo, Uruguay, 2000.

[9] Intel Corporation: *LXT974/LXT975 Fast Ethernet 10/100 Quad Transceivers*, 2001.
URL http://www.intel.com

[10] Mohor, Igor: *Ethernet IP Core Specification*. Opencores, 2002.
URL http://www.opencores.org

[11] PLX Technology, Inc.: *PCI 9054 Data Book*, 2000.
URL http://www.plxtech.com

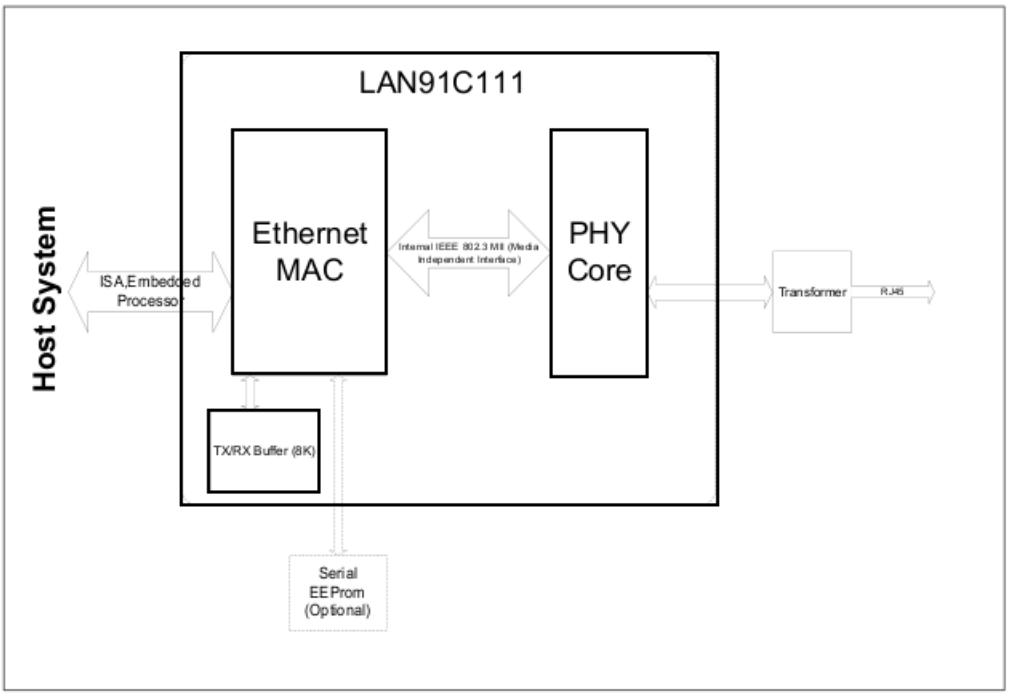
[12] SMSC, Standard Microsystems Corporation, 80 Arkay Drive, Hauppauge, NY: *10/100 Non-PCI Ethernet Single Chip MAC + PHY*, 2004.
URL http://www.smsc.com

Parte V

Anexos

24. Anexo 1: Chip LAN91C111

Además del chip PHY con 4 bocas Ethernet con el que trabajamos durante el proecto, la placa PgVIrtex cuenta con el chip Ethernet LAN91C111 (ver figura 24). Dado que no funciona en ninguna de las dos placas, y que no pudimos dar con el problema que tienen, este chip no fue utilizado durante el proyecto. Sin embargo cuando aún no disponíamos de las placas estudiamos las características del mismo, ya que la idea en principio era diseñar cores para manejar todos los puerto Ethernet de la placa.



m

Figura 33: Arquitectura del chip LAN91C111[12]

Este chip está diseñado para facilitar la implementación de aplicaciones embebidas que utilicen conectividad Fast Ethernet[12]. Es un dispositivo que mezcla señales análogas y digitales y que implementa la parte de PHY y la subcapa MAC del protocolo CSMA/CD funcionando a 10 Mbps y a 100Mbps. Puede utilizar buses de datos de 8, 16 ó 32 bits, y tiene una memoria FIFO interna de 8 Kbytes para almacenar datos de transmisión y rececpión.

El control de memoria se realiza mediante una arquitectura MMU[25] y un bus interno de 32 bits. Esta arquitectura de mapeo de entrada/salida es capaz de realizar envío recepcion de datos con gran performance. Tiene además un esquema de utilización de buffers que le permite alocar datos en memoria dinámicamente, reduciendo así las tareas de software y aliviando el trabajo del procesador.

El 91C111 provee una interfaz esclava flexible que permite una sencilla conexión con buses estándar de la industria. El BIU [26] maneja tanto señales síncronas como asíncronas por separado. Es compaatible con el protocolo ISA, aunque este protocolo no funciona con tráfico de 100 Mbps.

Del lado de la red, el chip soporta dos tipos de interfaces. La primera es un par de transmisión/recepción estándar de Magnetics con una interfaz 10/100Base-T utilizando un bloque de capa física interno. La segunda implementa la especificación estándar de MII[27], utilizando transferencias de datos de a nibbles. Esta interfaz es aplicable tanto a 10 Mbps como a 100 Mbps. Tres de los pines del LAN91C111 son utilizados para la interfaz serie de contorl de la MII. A través de la interfaz MII podemos conectar este chip a otro que solo maneje la capa PHY y así aprovechar la capa MAC brindada por el 91C111 y tener dos bocas de red.

El bloque analógico que implementa el PHY consta de:

- Transmisor con driver de salida y conformador de señales.
- Receptor de par trenzado con chip ecualizador.
- Codificador y decodificador Manchester
- Scrambler y de-scrambler.
- Auto-negociación.
- Interfaz de control (MII).
- Puerto serie para el manejo de la MII.

El circuito conformador de señales de salida y los filtros on-chip eliminan la necesidad de filtro externos, que generalmente son necesarios en aplicaciones 10Base-T y 100Base-T. El chip se configura automáticamente para trabajar a 10 ó 100 Mbps y operación de half-duplex o full-duplex

[25] Memory Management Unit
[26] Bus Interface Unit
[27] Media Independent Interface

mediante el algoritmo interno de auto-negociación.

25. Anexo 2: Protocolo AMBA

La especificación AMBA[28] define un estándar de comunicaciones on-chip para el diseño de microcontroladores embebidos de alta performance. Se definen tres buses distintos en esta especificación:

- AHB (Advanced High-Performance Bus).
- ASB (Advanced System Bus).
- APB (Advanced Peripheral Bus).

Las señales de AMBA se nombran de manera tal que la primer letra indica a qué bus está asociada la señal. Una "n"minúscula en el nombre indica que la señal es activa por nivel bajo. De no ser el caso, los nombres de las señales deben ser todos en mayúscula. Las señales de testing tienen como prefijoo una "T"sin importar el tipo de bus.

25.1. Advanced High-Performance Bus

El AHB AMBA está diseñado para módulos de sistemas con reloj de alta frecuencia y alta performance. El AHB actúa como el bus principal del sistema. AHB soporta una conexión eficiente de procesadores, memorias on-chip e interfaces de memorias externas off-chip. AHB está especificado además, para asegurar un uso sencillo con un flujo de diseño eficiente y técnicas de testing automatizadas.

Un diseño con AMBA AHB puede contener uno ó más buses maestros. Generalmente un sistema tiene al menos la interfaz maestra del procesador y la de testing. Puede además tener por ejemplo una interfaz maestra de acceso directo a memoria (DMA). La interfaz a una memoria externa, un puente APB y cualquier memoria interna son los AHB esclavos más comunes.

Un sistema típico AHB contiene:

- Maestro AHB: Un bus maestro tiene la capacidad de iniciar operaciones de lectura y escritura, entregando una dirección e información de control. Solamente se permite que un bus maestro esté activo en cualquier momento.

[28] Advanced Microcontroller Bus Architecture

- Esclavo AHB: Un bus esclavo responde a operaciones de lectura y escritura a un rango de espacio de direcciones dada. Las señales del bus esclavo responden al bus maestro activo si el proceso fue realizado, si hubo error o si aún debe esperar.
- Árbitro AHB: El árbitro del bus asegura que solo un bus maestro por vez tenga permitido iniciar transferencias. Aunque el protocolo de arbitraje es fijo, se puede implementar cualquier algoritmo como dar acceso con prioridades o por igual a todos los buses.
- Decodificador AHB: El decodificador AHB es utilizado para decodificar la dirección de cada transferencia y entrega una señal de selección para el esclavo que está involucrando en la transferencia. Un solo decodificador centralizado es requerido en toda implementación AHB.

Antes de que una transferencia AHB AMBA pueda comenzar, el bus master debe garantizar el acceso al bus. Este proceso es iniciado por el master seteando una señal de request al árbitro. Luego, el árbitro indica cuando el master tiene garantizado el acceso a las señles del bus. Estas señales proveen información sobre las direcciones, sentido y ancho de la transferecia, además de indicar si la transferencia forma pate de una ráfaga. Dos formas diferentes de ráfaga están permitidas:

- ráfagas en forma incremental, que no vuelven al comienzo en una dirección dada.
- ráfagas cíclicas, que comienzan de nuevo en una dirección particular.

Un bus de escritura de datos es utilizado para mover los datos desde el master al slave, mientras que un bus de lectura de datos se utiliza para mover datos desde el slave al master. Cada transferencia consiste en un ciclo de direcciones y control, y uno ó más ciclos para los datos.

La dirección no se puede mantener en el bus, por lo tanto todos los esclavos deben latchearla durante este período. Los datos sin embargo pueden mantenerse usando las señal HREADY. Mientras esta señal esté en nivel bajo, generará tiempos de espera en la transferencia, otorgando tiempo extra a los esclavos para tomar el dato.

Durante una transferencia el esclavo muestra el estado utilizando las señales de respuesta, HRESP[1:0]:

- OKAY: es utilizada para indicar que la transferencia se está procesando con normalidad y cuando HREADY se lleva a '1' muestra que la tranferencia fue completada con éxito.
- ERROR: Indica que ha ocurrido un error en la transferencia.
- RETRY y SPLIT: ambas indican que la transferencia no puede ser completada en ese momento pero que el maestro debería continuar intentando realizarla.

Operando normalmente un maestro tiene permitido completar todas las transferencias en una ráfaga particular, antes que el árbitro le de acceso al bus a otro master. Sin embargo, para evitar latencias excesivas del árbitro es posible terminar una ráfaga, en cuyo caso el maestro debe pedir nuevamente el bus para completar la transferencia.

25.2. Advanced System Bus

ASB es la primera generación de buses del sistema AMBA. Implementa las características requeridas para sistemas de alta performance incluyendo:

- transferencias en ráfaga.
- operaciones de transferencia en pipeline.
- multiples buses maestros.

Un sistema con AMBA ASB puede contener uno ó más buses maestros. Generalmente un sistema tiene al menos la interfaz maestra del procesador y la de testing. Puede además tener por ejemplo una interfaz maestra de acceso directo a memoria (DMA). La interfaz a una memoria externa, un puente APB y cualquier memoria interna son los AHB esclavos más comunes.

Un sistema típico ASB contiene:

- Maestro ASB: Un bus maestro tiene la capacidad de iniciar operaciones de lectura y escritura, entregando una dirección e información de control. Solamente se permite que un bus maestro esté activo en cualquier momento.
- Esclavo ASB: Un bus esclavo responde a operaciones de lectura y escritura a un rango de espacio de direcciones dada. Las señales del bus esclavo responden al bus maestro activo si el proceso fue realizado, si hubo error o si aún debe esperar.

- Árbitro ASB: El árbitro del bus asegura que solo un bus maestro por vez tenga permitido iniciar transferencias. Aunque el protocolo de arbitraje es fijo, se puede implementar cualquier algoritmo como dar acceso con prioridades o por igual a todos los buses.
- Decodificador ASB: El decodificador del bus realiza la decodificación de las direcciones de transferencia y elige apropiadamente el esclavo. Además asegura que el bus permanezca operacional cuando no se requieren transferencias. Un solo decodificador centralizado es requerido en toda implementación ASB.

25.3. Advanced Peripheral Bus

El APB es parte de la jerarquía de buses AMBA y está optimizado para consumo de potencia mínimo y reducida complejidad en la interfaz. El AMBA APB aparece como un bus local secundario que está encapsulado como un único dispositivo esclavo ASB o AHB. APB provee una extensión de bajo consumo directamente conectado a señales ASB ó AHB.

El puente APB aparece como un módulo esclavo que maneja el handshake del bus y controla la señal de retiming del lado del bus periférico local. El APB AMBA debería ser usado como interfaz de cualquier periférico que tenga poco ancho de banda y que no requiera la performance de una interfaz de un bus con pipeline.

La última revisión del APB está especificada de manera tal que las transiciones de todas las señales están relacionadas solamente con el flanco de subida del reloj. Esta mejora permite asegurar que los periféricos APB puedan ser integrados fácilmente en cualquier flujo de diseño, con las siguientes ventajas:

- operación en alta frecuancia más fácil de lograr.
- performance independiente del ciclo de trabajo del reloj.
- análisis de tiempo estático simplificado por el uso de un solo flanco de reloj.
- no existen consideraciones especiales a la hora de insertar tests automáticos.

Estos cambios en el APB hacen que la interfaz a los AHB sea más sencilla. Una implementación típica de un AMBA APB contiene un solo puente APB, el cual se requiere para convertir las transferencias AHB ó ASB al

formato adecuado para los esclavos en el APB. El puente provee el latcheo de todas las señales de direcciones, datos y control además de proveer un segundo nivel de decodificado para generar señales de selección del dispositivo esclavo para los periféricos APB.

Fuera del puente APB, los demás módulos son esclavos. Los esclavos APB tienen la siguiente especificación de interfaz:

- las direcciones y control deben ser válidos surante el acceso (sin pipeline).
- consumo cero en la interfaz mientras no haya actividad en el bus.
- las señales de reloj pueden ser provistas por decodificación con un strobe.
- la escritura de datos debe ser válida durante todo el acceso (permitiendo implementaciones de latcheos transparentes y libres de glitch).

25.4. Elección del bus del sistema

Antes de decidir qué bus ó buses deben ser utilizados en el sistema, se debe considerar lo siguiente:

Elección del bus principal del sistema

AMBA AHB y ASB son candidatos a ser usados como bus principal del sistema. Generalmente la eleccion dependerá de la interfaz provista por los módulos que requiere el sistema. El AHB es recomendado para todos los diseños nuevos.

Buses del sistema y periféricos

Crear todos los periféricos como módulos funcionales ASB ó AHB es factible, pero no sempre puede ser lo deseado. En diseños con un gran número de periféricos con macroceldas, el incremento en la carga de un bus puede resultar en un incremento del consumo requerido y sacrificar el renimiento. Además hay que recordar que en un diseño, el elemento más lento del bus limita el rendimiento del sistema, y muchas veces los periféricos solo requieren un latch de direcciones y operaciones de lectura y escritura que pueden ser realizados con un APB.

Cuando elegir ASB/AHB ó APB

Una interfaz AHB ó ASB es utilizada para:

- maestros de bus.
- bloques de memoria on-chip.
- interfaces de memoria externa.
- periféricos con alto ancho de banda y con interfaces FIFO.
- periféricos esclavos con DMA.

Una interfaz APB es recomendad para:

- registros simples como dispositivos esclavos.
- interfaces de poco consumo donde los relojes no puedan ser asignados como globales.
- agrupar periféricos que utilizen pocas señales de bus para evitar cargar el bus del sistema.

26. Anexo 3: Plan inicial de proyecto

26.1. Descripción de los entregables intermedios.

Primer entregable : Febrero de 2008

Para esta etapa se prevé tener un bloque de hardware que maneje los chips Ethernet de la placa PgVirtex. Se entregará un informe de lo realizado, que consistirá en tener montado una plataforma de prueba para los puertos Ethernet.

Segundo entregable : Junio de 2008.

Para esta nueva instancia tendremos implementadas las pruebas de funcionamiento de los puertos Ethernet funcionando con el procesador LEON.

26.2. Descripción del proyecto

Para el proyecto se utilizará la placa PgVirtex, desarrollada en el IIE por un grupo de proyecto del año 2005. Dicha placa posee un FPGA Virtex-E de Xilinx y distintos puertos de comunicación, algunos de los cuales no han sido utilizados aún. Entre ellos varios puertos ETHERNET y USB. Uno de los objetivos del proyecto será el de hacer funcionar los anteriormente nombrados puertos Ethernet, y dejar módulos preparados para que puedan ser utilizados en otros proyectos.

Además nuestro proyecto será la continuación de otro proyecto llamado CICLOP, que consistió en integrar en el FPGA antes mencionado un "System on a Chip" basado en el microprocesador LEON2 de la arquitectura SPARC. Utilizando este sistema y los puertos anteriormente mencionados, correremos una aplicación a definir, posiblemente hagamos correr un LINUX para SPARC que maneje los puertos Ethernet.

Además debemos estudiar la compatibilidad del LEON con PCI y la transferencia de datos hacia y desde la memoria de nuestro sistema a través de PCI.

26.3. Objetivo general del proyecto

¿Qué?

Implementar un bloque de Hardware genérico que sea capaz de manejar los dos tipos de puertos Ethernet de la placa PG-VIRTEX. Luego de implementar este bloque y dejarlo funcionando, se lo integrará a un sistema compuesto por un microprocesador Leon de la familia Sparc. El bus PCI solo ha sido probado para programar la placa. Probar si es posible el bus PCI en una aplicación con Leon.

¿Para qué?

Principalmente para que futuros proyectos a realizarse con la PgVirtex puedan utilizar los puertos Ethernet sin tener que preocuparse por implementar el controlador. Además para que se pueda correr una aplicación que utilice Ethernet con el sistema implementado por el proyecto CicloP.

Prioridades

Lo más importante es que todos los puertos Ethernet queden funcionando y listos para ser usados en aplicaciones futuras.

Criterios de aprobación

Se correrá una aplicación sencilla que permita verificar el correcto funcionamiento de los puertos Ethernet y bus PCI, ya sea en el caso con o sin Leon.

26.4. Actores, supuestos y restricciones

Actores

- El equipo del proyecto.
- Nuestro tutor, Juan Pablo Oliver.
- El equipo de proyecto que está trabajando con las placas.

Supuestos

Suponemos que contaremos con las placas PgVirtex a más tardar en Enero del 2008. Solo contamos con dos placas por lo que suponemos que el grupo de fin de carrera que está trabajando con ellas no las romperá (y nosotros tampoco). Nadie ha utilizado los puertos Ethernet ni estos han sido probados luego de la fabricación de las placas, por lo tanto suponemos que funcionan los chips y que están bien soldados.

A los efectos del cronograma, se asume que no surgirán ausencias de algún integrante del grupo ó del tutor por causa de fuerza mayor. Se asume que las documentaciones de PgVirtex y CicloP están exentas de errores.

Suponemos que la interfaz PCI esta probada y anda para programar la PgVirtex.

Restricciones

- El proyecto debe de cumplir con el cronograma establecido.
- Solo contamos con dos placas PgVirtex. Una de ellas se sabe que tiene fallas.
- Sólo contaremos con las placas después de Enero del 2008.
- El tamaño del FPGA es limitado.

26.5. Especificación funcional del proyecto

Nuestro proyecto se basa en la implementación de bloques VHDL genéricos fácilmente parametrizables por el usuario para el manejo de los puertos Ethernet y PCI. Estos bloques deberán cumplir los protocolos y normas establecidas por el estándar y como prueba del buen funcionamiento de nuestro diseño deberemos desarrollar una plataforma de prueba para estos puertos.

26.6. Definición detallada del alcance

- Hacer funcionar los puertos Ethernet sin el Leon.
- Hacer funcionar los puertos Ethernet con el procesador Leon.

- Hacer funcionar la interfaz PCI-PgVirtex.

Los entregables de los dos primeros objetivos consistirán en el informe de una prueba a realizar sobre los puertos Ethernet que verifique el correcto funcionamiento de los mismos y de los cores utilizados. Se informarán los resultados obtenidos y se los comparará con lo exigido por el protocolo Ethernet.

Para el tercer objetivo específico el entregable consistirá en un informe que detalle los resultados obtenidos en las aplicaciones con PCI a implementar. Aún no están definidas las aplicaciones a implementar para este caso, pero seguramente se tratará de utilizar el bus PCI para pasaje de datos entre la placa y la PC además de configurar el FPGA a través del mismo.

26.7. Análisis de riesgos

- Ausencia de algún integrante del grupo por causas de fuerza mayor.
- Fallas en el hardware de la placa.
- No contar con la placa en la fecha supuesta.
- Alguna de las tareas no sea implementable, o no implementable en el tiempo supuesto.

Tomando en cuenta los criterios de impacto y probabilidad de ocurrencia, vamos a analizar dos posibles hechos que puedan afectar el normal desarrollo de nuestro proyecto.

Fallas en el hardware de la placa

Los chips que controlan los puertos Ethernet no han sido testeados por lo que corremos el riesgo de que existan fallas en el hardware que nos imposibiliten su utilización. Contamos con dos placas, por lo cual la probabilidad de que esto nos afecte se ve reducida.

Como medida de prevención se testearán las placas bastante antes de las pruebas de hardware, ya que de esta manera tendríamos tiempo de buscar alguna solución como podría ser alguna tarea de reparación.

No contar con la placa en la fecha supuesta

Este hecho lo consideramos de alto impacto (sobre todo para el primer entregable) y alta probabilidad de ocurrencia ya que depende de factores externos (está siendo utilizada por otro grupo).

Para prevenir la ocurrencia de esto tendremos que estar en contacto con el grupo que está utilizando las placas para tener mas controlada la situación.

En caso de que igualmente suceda, para minimizar el impacto sobre nuestro proyecto nos veremos obligados a modificar nuestro cronograma. Es decir, comenzaremos con el desarrollo del segundo objetivo específico antes de realizar las pruebas hardware del primero.

27. Anexo 4: Diagramas de Gantt y estimación de horas: Plan vs. Real

A continuación se presenta la comparación entre los diagramas de Gantt realizados para el plan de proyecto y el realizado al finalizar el mismo. Aquí veremos la diferencia entre la planificación y lo que resultó durante el desarrollo del proyecto explicando la razón de las modificaciones realizadas, además de una reseña sobre las horas dedicadas al proyecto en comparación con las estimadas.

Primera etapa

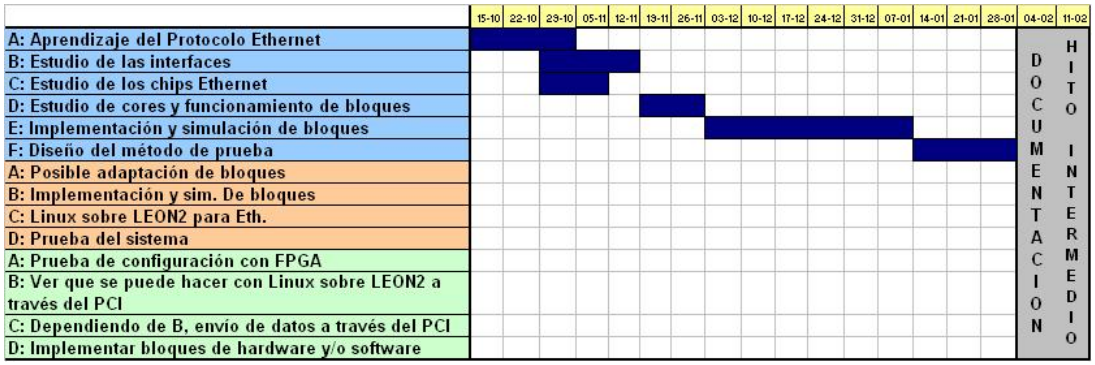


Figura 34: Diagrama de Gantt: plan (1ª parte)

Figura 35: Diagrama de Gantt: real (1ª parte)

En la primera etapa del proyecto se estimó llegar a fin de Enero con los

objetivos del primer hito intermedio cumplidos. Este primer entregable se realizó una semana después de lo que supusimos en un principio, y sin embargo no pudimos llegar con todos los objetivos cumplidos.

Las dos razones principales para este retraso fueron una mala estimación en lo que fue el estudio de los cores utilizados para nuestro diseño, además de haber recibido las placas para implementar nuestro diseño y probarlo tan solo una semana antes de este primer hito intermedio. El no contar con las placas hasta Febrero, provocó un cambio en nuestro plan, ya que implementamos un ambiente para simular nuestro diseño. Este ambiente de simulación, si bien nos sirvió para detectar y corregir algunos errores, no nos fue útil para simular por ejemplo los relojes del sistema, que fue el principal problema encontrado durante las pruebas.

Para intentar recuperar el tiempo perdido, y tal como estaba previsto en el plan de proyecto se comenzó con la segunda etapa del proyecto unas semans antes, en un intento por no atrasarnos en el calendario general.

Segunda etapa

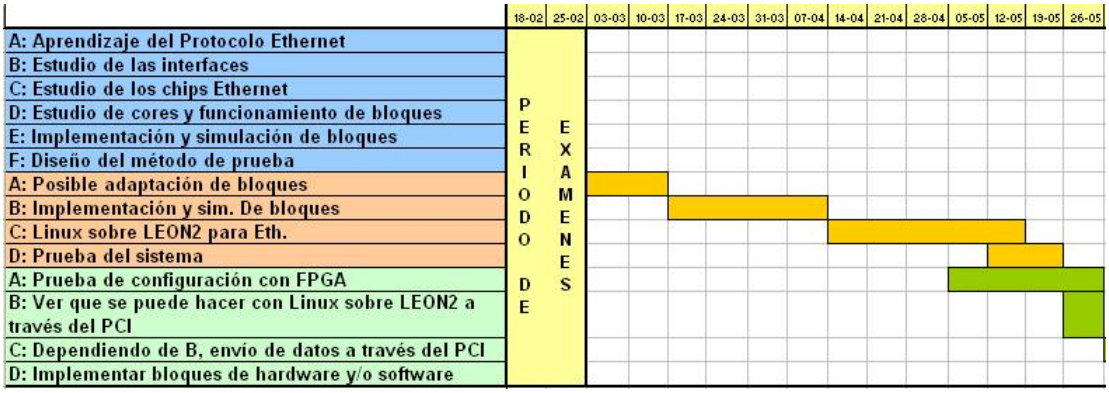


Figura 36: Diagrama de Gantt: plan (2ª parte)

La solución de los problemas encontrados durante las pruebas en la primera etapa del proyecto no fueron de sencilla solución, y nos llevó más tiempo del estimado. Por lo tanto, pese a las medidas tomadas para contrarrestarlo, el retraso en el calendario persistió. Durante la segunda etapa nos encontramos ante un nuevo problema: la implementación y simulación del diseño nos llevó casi el doble de tiempo que el previsto como se puede ver

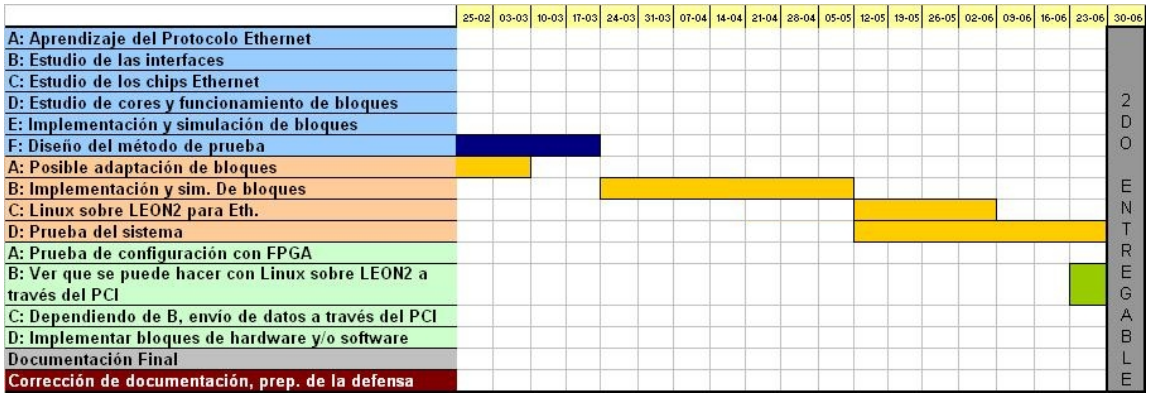


Figura 37: Diagrama de Gantt: real (2ª parte)

en las figuras 36 y 37.

Influyó en esta mala estimación la escasa documentación existente acerca de los requerimientos para la configuración del SOC LEON2-Linux y de la configuración en si. Además, de nuestra parte, la experiencia en el manejo de los sistemas operativos linux era muy limitada a esta altura. Esto también repercutió sensiblemente al no encontrar rápidamente solución a ciertos problemas.

En consecuencia, llegamos al segundo hito intermedio con los objetivos de la segunda etapa cumplidos, pero practicamente sin haber comenzado con la última etapa que consistió en el manejo del puerto PCI de la placa PgVirtex. En este punto el retraso era de un mes y medio, lo cual nos podía comprometer seriamente en el tramo final.

Tercera etapa

Llegamos al comienzo de esta última etapa con un retraso de aproximadamente un mes, arrastrado por los problemas de las etapas anteriores. En este punto, nos encontramos muy comprometidos con el desarrollo de nuestro proyecto, ya que estábamos muy apretados por el calendario si queríamos llegar a comenzar con la documentación en la última semana de Julio. De hecho fuimos advertidos al respecto en la presentación del segundo entregable.

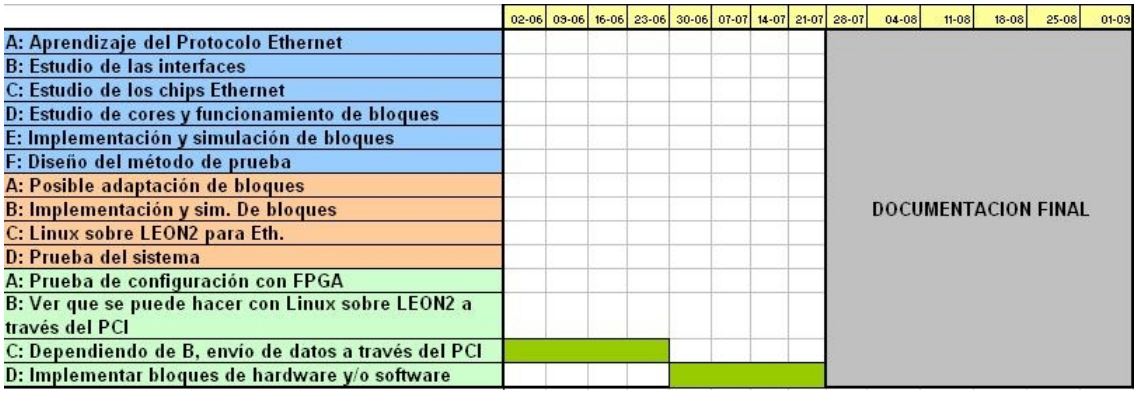


Figura 38: Diagrama de Gantt: plan (3ª parte)

Figura 39: Diagrama de Gantt: real (3ª parte)

El desarrollo de esta última etapa nos llevó la cantidad de semanas estimadas en un principio, pese a que en la estimación aún no estaba bien definido lo que se iba a realizar.

Para intentar achicar las diferencias del calendario real respecto al planificado, se comenzó a documentar unas semanas antes de terminar con las pruebas del diseño que utiliza el puerto PCI. Igualmente se llegó a la culminación de la documentación un mes después de lo previsto y sobre la fecha límite de entrega.

Las últimas semanas previas a la defensa se dedican a correcciones en la documentación, y a la preparación de la defensa.