UNIVERSIDAD POLITÉCNICA DE VALENCIA

MÁSTER EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

UNIVERSIDAD
POLITECNICA
DE VALENCIA

"Contribución al Diseño del Sistema de Adquisición de Datos para el Experimento NEXT"

TRABAJO FINAL DE MÁSTER

Autor/es:
Alfonso Tarazona Martínez

Director/es:
Dr. José F. Toledo Alarcón

VALECIA, septiembre de 2010

En primer lugar, quiero dar las gracias a mi director de trabajo final de máster, y jefe, Dr. J. Francisco Toledo Alarcón, por confiar en mí y darme la oportunidad de trabar en un proyecto internacional como NEXT.

En segundo lugar, a toda la gente del CERN con la que he trabajado, Hans Müller, Pierre Vande Vyvre, Filippo Costa, Sorin Martoiu,…

Y finalmente, cómo no, a mi familia y amigos. Sin ellos la vida no tendría sentido.

Muchísimas gracias

ÍNDICE

1 Introducción

1.1 Introducción

NEXT es un experimento de física de neutrinos con fuerte participación española en el que el tutor de este trabajo final de máster es coordinador de la electrónica del experimento. El autor de este documento está contratado por la UPV con cargo al proyecto y participa en el diseño de la electrónica de adquisición de datos.

Brevemente, el detector es cilindro lleno de un isótopo de gas xenon que al desintegrarse libera dos electrones y dos neutrinos. Los electrones liberados ionizan el gas. Un campo eléctrico en el interior de volumen conduce los iones hacia una rejilla amplificadora que mediante un campo eléctrico muy elevado produce una avalancha de fotones por electroluminiscencia.

Estos fotones son parcialmente detectados en un plano de 10.000 detectores de silicio junto a la rejilla amplificadora, cuya función es recoger información sobre el recorrido de los electrones primarios (*tracking*). El resto de los fotones son reflejados en las paredes internas del cilindro y llegan hasta el extremo opuesto, donde un plano de unos 400 fotomultiplicadores detecta la energía de los fotones (y por tanto del evento). Estos mismos fotomultiplicadores detectan también la luz primaria originada tras la desintegración, lo que da una marca temporal para la desintegración y añade una tercera coordenada a la información recogida por el plano de *tracking*.

Las señales producidas por los sensores (cerca de 11.000) son convenientemente amplificadas, conformadas, digitalizadas y formateadas por la electrónica de front-end. Ésta entrega los datos a las unidades denominadas FEC (*Front-end Electronic Concentrator*) que es donde se centra este trabajo. Los módulos FEC están basados en FPGA y han sido diseñados en Valencia (grupo de electrónica de la colaboración NEXT, UPV). Estas tarjetas formatean los datos y realizan la comunicación con el PC de adquisición.

El destino final de los datos es el PC encargado de la adquisición de datos, sobre él corre un software denominado DATE (*Data Acquisition and Test Environment*) y que se emplea en ALICE; uno de los cuatro grandes experimentos del LHC en el CERN. DATE actualmente acepta datos sólo desde unos módulos específicos utilizados en ALICE.

El objetivo de este trabajo es ampliar DATE para aceptar datos provenientes desde Gigabit Ethernet. En este sentido se ha iniciado una colaboración con el equipo DATE de ALICE para dar soporte a esta tecnología de red.

1.1.1 El concepto SRS

Un científico del Fermilab dijo una vez que todos los sistemas DAQ están, por definición, más allá del estado del arte: nadie diseña un sistema nuevo cuando uno viejo puede ser mejorado o adaptado para que desempeñe las funciones deseadas. Este principio, reutilizando desarrollos existentes, es potente cuando éste se aplica a colaboraciones pequeñas, tal como NEXT.

Un segundo principio favorece los esfuerzos conjuntos con otras instituciones de investigación para llevar adelante desarrollos comunes. Esto por lo general, implica la adición de características de hardware y *firmware* que un no necesita (aunque los otros si) y encontrase con complicaciones de gestión del proyecto más grandes, pero las ventajas obtenidas, en gran parte, recompensan el esfuerzo invertido.

Teniendo en cuenta ambos principios, NEXT se afilió a la colaboración RD-51 en 2009, dedicado y ayudado a desarrollar un nuevo sistema DAQ llamado *Scalable Redaout System*, liderado por Dr. Hans Muller, PH-AID, CERN. Este planteamiento reusa componentes de experimentos existentes (como DATE, el software DAQ de ALICE) y define nuevos componentes para ser desarrollado, compartiendo el esfuerzo entre instituciones de investigación diferentes.

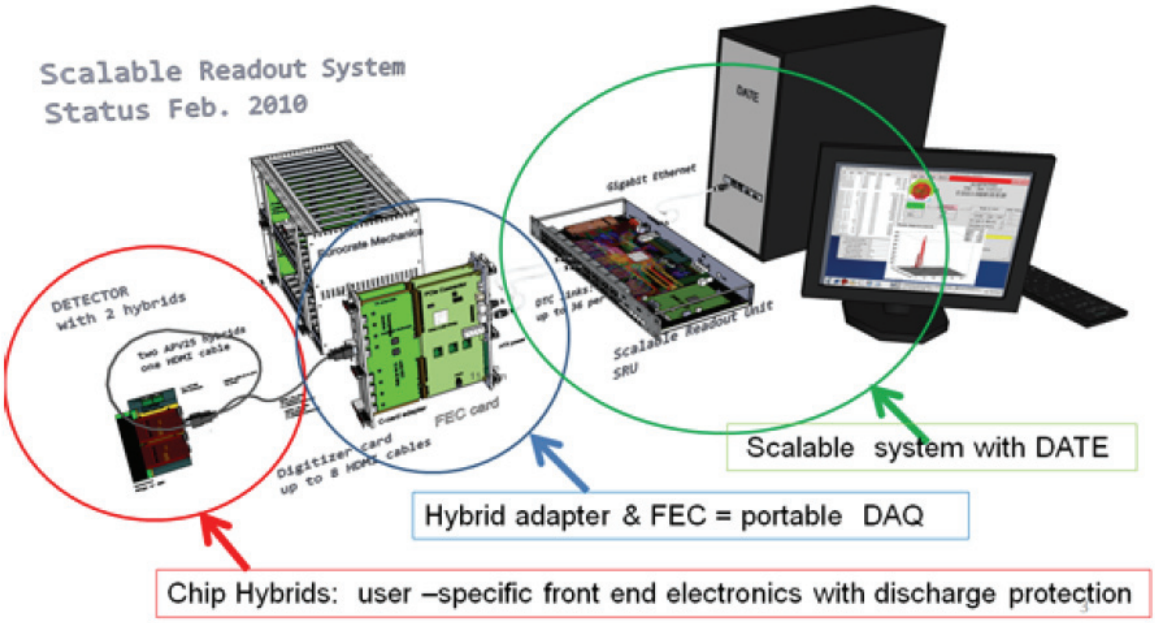


Figura 1-1. Arquitectura SRS

RD-51 (http://rd51-public.web.cern.ch/RD51-Public/Welcome.html) es una Colaboración R&D Colaboración del CERN para el *"Development of Micro-Pattern Gas Detectors Technologies"*. Uno de los objetivos es diseñar un sistema DAQ portable, de varios canales para tales detectores. Cuatro organizaciones de NEXT también, pertenecen a RD-51 (Univ de Zaragoza, Univ. Politécnica de Valencia, Univ de Valencia y Univ. Autónoma de Barcelona).

La idea detrás de SRS es resumida en la Figura 1-1. En el nivel inferior de la jerarquía se encuentran los módulos de *front-end* de aplicación específica para leer los datos entregados por los diferentes sensores. Éstos se definen por los usuarios (experimentos) y no forman parte del SRS. En NEXT-1-EL, se han desarrollado dos módulos de *front-end* específicos para diferentes detectores, PMTs y SiPMs.

En en nivel superior de la jerarquía, se encuentra una red de ordenadores gobernados por el software DAQ, DATE, reciben datos del evento de los módulos DAQ vía enlaces GbE (Gigabit Ethernet). El enlace GbE es una nueva característica de DATE, a consecuencia de un interés común de ALICE y RD-51. NEXT ha firmado un acuerdo con el equipo de ALICE DAQ para usar DATE en NEXT.

Entre el entorno existente de DATE (adaptado a GbE) y el *front-end* específico de usuario (los ya existentes o desarrollos nuevos), se encuentran los componentes genéricos DAQ e interfaz FE-DAQ:

1. La tarjeta FEC (*Front-End Concentrator*) es un interfaz genérico entre el sistema DAQ y una amplia variedad de módulos *front-end*. Esto se logra conectando otras tarjetas a la tarjeta FEC. Estas tarjetas son el interfaz entre la tarjeta principal FEC y en el *front-end*.
2. El módulo SRU (*Scalable Redout Unit*) es un derivado de un módulo existente en el CERN (llamado LCU, *LED Control Unit*). Éste soporta conexiones LVDS mediante el cable de red estándar de hasta 36 FECs y está destinado para sistemas de gran escala.

Aunque GbE sea la tecnología de enlace empleada para el DAQ para NEXT-1-EL, 10-GbE será implementado y chequeado en el módulo SRU en 2011. En otoño, se recibirá un módulo SRU (desde el CERN).

Papel de NEXT en el concepto SRS

¿Cuál es el papel de NEXT en el concepto SRS? Hasta ahora se ha contribuido con:

1. El desarrollo del hardware de la tarjeta FEC y *firmware* básico.
2. El desarrollo y prueba de un interfaz GbE compatible con la DATE para los módulos SRU y FEC. Esto es una parte del *firmware* de FPGA que transmite eventos de datos hacia arriba y recibe datos de configuración y órdenes, empleando tramas UDP a 1 Gbps, también, usando un simple mecanismo de control de flujo. El interfaz permite la transmisión de datos y paquetes de *slow control*.
3. Se ha participado en la prueba y desarrollo de *firmware* para una tarjeta diseñada en el CERN, la cual se conecta a la tarjeta FEC. Esta tarjeta posee dos ADCs de 16 canales, usada en NEXT-1-EL (digitaliza los datos provenientes de los detectores PMTs) y RD-51 (digitaliza los datos provenientes de los detectores APV25).

Se han obtenido beneficios de la adaptación de DATE a GbE, tales como el permiso de usar DATE, la tarjeta de ADCs diseñada en CERN, usada para la lectura de los PMTs en NEXT-1-EL y la solución de varias cuestiones mecánicas y de suministro de energía (como una tarjeta con fusibles y filtros EMI que diseñarán en el CERN que permite la utilización de fuentes de alimentación ATX estándar para la electrónica DAQ).

A finales de 2010 se recibirá un nuevo módulo SRU diseñado en el CERN, que permitirá adquirir experiencia con tecnologías de vanguardia (como las FPGAs Virtex-6 de Xilinx y 10-GbE), el cual puede ser usado en una futura modernización del módulo FEC. En 2011, se espera que los usuarios de DATE y la tarjeta FEC crezcan bastante para intercambiar código y experiencia con otras instituciones.

Resumiendo, la colaboración con PH-AID del CERN y RD-51 ha traído muchos beneficios al sistema de electrónica de NEXT, compartiendo ideas, esfuerzo de diseño y costes de desarrollo.

1.1.2 La electrónica DAQ

Se llama electrónica DAQ (ver Figura 1-2) al conjunto de módulos de electrónica, *firmware* y software que son responsables de obtener datos de la electrónica de *front-end*, extrayendo parámetros de señal y creando subeventos que son enviados a niveles superiores en la

cadena de adquisición hasta alcanzar la red de PCs, en la cual está instalado el software de DAQ DATE.

La tarjeta FEC es el módulo DAQ usado en NEXT-1-EL (ver sección 1.1.3). El módulo SRU (diseño del CERN) también, puede estar usado, aunque es más adecuado usar dicho módulo en sistema de gran escala, como NEXT-100.

Figura 1-2. Principales componentes del hardware DAQ en NEXT-1

Los PCs con DATE (*Local Data Concentrators*, LDCs) no forman parte del DAQ, pero si del sistema Online (se refiere al software DATE y al hardware, en el cual funciona, Local *and Global Data Contentrators*, LDCs y GDCs).

1.1.3 La tarjeta FEC

La tarjeta FEC está concebida como un interfaz genérico entre el sistema DAQ y una amplia variedad de módulos de *front-end*. Como se comentó anteriormente, esto se consigue conectando otras tarjetas de uso específico al módulo FEC. Forma parte del concepto SRS y ha sido diseñada por el la Colaboración NEXT en Valencia (Electrónica UPV).

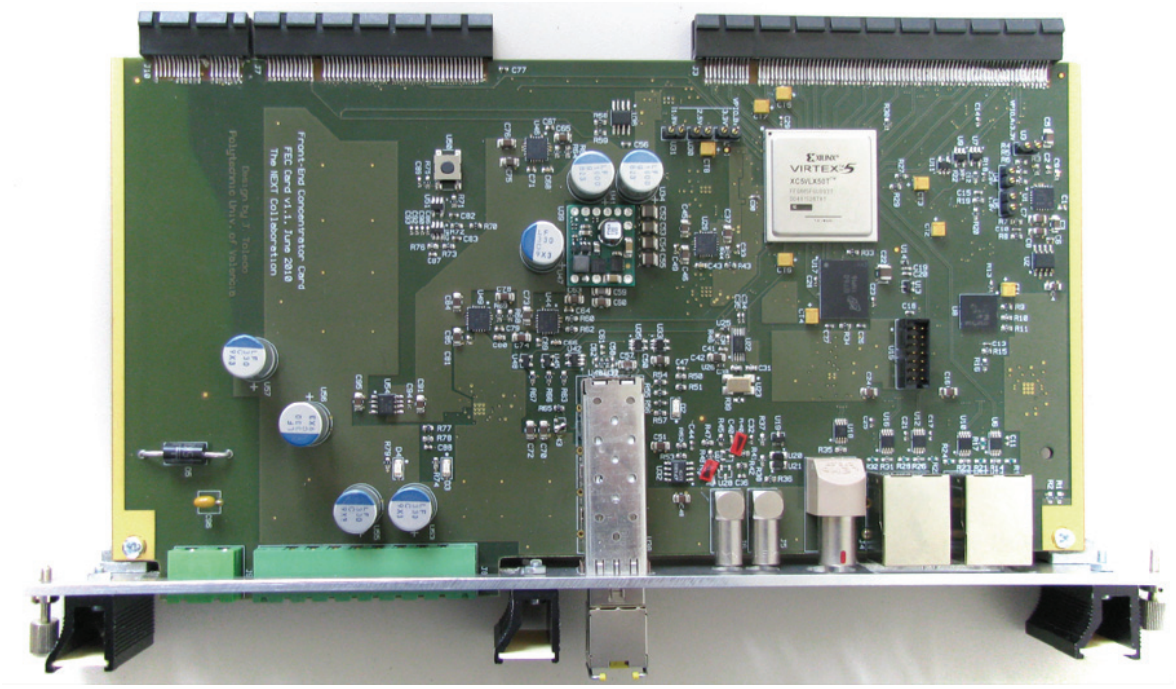


Figura 1-3. Tarjeta FEC

La Figura 1-3 permite destacar sus características principales. El corazón del sistema es una FPGAVirtex-5 LX50T, con un encapsulado de 665 bolas. Está conectada a distintos recursos de I/O (entrada/salida):

- En el panel frontal, de derecha a izquierda en la Figura 1-3, la FPGA está conectada a dos conectores RJ-45 (cada uno con dos pares LVDS de entrada y dos pares LVDS de salida), una entrada LEMO LVDS, una entrada NIM LEMO, una salida NIM LEMO, y finalmente un módulo SFP usado como interfaz GbE. Los dos los conectores verdes de la izquierda son entradas de alimentación.
- En la parte superior de la Figura 1-3 se encuentran tres conectores PCI-Express para conectar otras tarjetas. El conector de la izquierda, PCI-Express x1, es un conector de alimentación. Los otros dos conectores, PCI-Express x8 y x16, proporcionan distintas conexiones de I/O (pares diferenciales de alta velocidad, buses I^2C, señales single-ended) para conectar otras tarjetas.

Otras características que incluye la tarjeta FEC son, una memoria EEPROM, destinada a la identificación del dispositivo y una memoria DDR2 de 256 MB para almacenar eventos.

Figura 1-4. Tarjeta FEC conectada a una tarjeta de expansión (tarjeta ADCs) montada en un rack de 19" 6U

El módulo FEC ha sido parcialmente testeado en el CERN durante el pasado mes de agosto con *firmware* básico que permite trabajar con el interfaz GbE y la tarjeta de expansión, tarjeta de ADCs (ver Figura 1-4). Durante en próximo mes de septiembre se llevarán a cabo más pruebas y se desarrollará *firmware*.

1.1.4 *Slow Control*

Slow Control es un término que se usa para el *firmware* y software que es responsable de la lectura y la escritura de datos de configuración en el DAQ y electrónica de *front-end* cuando la adquisición está detenida. Esto incluye el encendido y apagado de los sensores individuales de adquisición, el ajuste de umbrales, etc. En resumen, el ajuste de los diferentes parámetros de configuración antes de llevar a cabo una adquisición de datos.

La conexión GbE entre la tarjeta FEC y los LDCs (la primera etapa de un sistema Online basado en DATE) es usado para este fin, usando un esquema de memoria mapeada. Esto significa que cada parte de la electrónica tiene un juego de registros y tablas, los cuales son mapeados en una memoria.

Una aplicación simple ha sido implementada y probada durante el mes de agosto (ver sección 11.4). Ésta permite que el usuario envíe órdenes de escritura/lectura a la tarjeta FEC vía el enlace GbE usando tramas UDP especiales.

Esto puede emplearse para configurar el sistema a través de un fichero (función no implementada todavía). DATE lee los datos de configuración de este fichero y genera las tramas UDP correspondientes de *slow control*.

1.2 Objetivos

Esta tesina de máster tiene como meta el estudio y desarrollo de una solución para la adquisición de datos para el experimento NEXT basada en Gigabit Ethernet y el software DATE del CERN. Objetivos parciales de esta tesina son:

1. Usando una placa de desarrollo, generar paquetes Gigabit Ethernet desde FPGAs Virtex-5 LXT, comprensibles por el software DATE.
2. Medir las prestaciones de un enlace de comunicación directo, en función de la frecuencia de generación de las tramas y del tamaño de las mismas.
3. Definir e implementar un mecanismo de control de flujo entre el PC y la tarjeta FEC que permita optimizar el *throughput* del sistema.
4. Definir e implementar el *firmware* necesario en la tarjeta FEC para probar sobre hardware real los pasos 1, 2 y 3 anteriores.
5. Estudiar los puntos 2 y 3 anteriores para un escenario compuesto por varias unidades FEC conectadas al PC de adquisición a través de un router.
6. Colaborar con equipos científicos internacionales, realizando estancias de corta duración (2-6 semanas) en el CERN.

1.3 Plan de trabajo

El plan de trabajo que se ha seguido se basa en seis fases fundamentales, las cuales se detallan a continuación:

1. Estudio del *Hard IP Core: Embedded Tri-Mode Ethernet MAC Wrapper* de Xilinx.
2. Comprobar el funcionamiento del *core* en una placa de desarrollo.
3. Implementación de un sistema que permitiese enviar y recibir tramas UDP.
4. Comprobar el funcionamiento del sistema anterior (estas pruebas fueron realizadas en el CERN, laboratorio de DATE, ALICE con la ayuda de Filippo Costa empleando placas de desarrollo).
5. Mejora del sistema implementado en el apartado 3, haciéndolo más genérico e incorporando nuevas funciones.

6. Comprar el funcionamiento del sistema anterior (estas pruebas fueron llevadas a cabo en el CERN, laboratorio de DATE, ALICE con la ayuda de Filippo Costa usando tarjetas FEC).

2 Aplicación Gigabit Ethernet basada en FPGA

A la hora de implementar una aplicación de Ethernet sobre FPGA se pueden emplear varias soluciones para conseguir el propósito deseado:

1. *Hard IP Core*
2. *Soft IP Core*

Los ***Hard IP Cores*** son bloques preimplementados, en fábrica, usando parte del silicio de la FPGA, tales como microprocesadores, interfaces gigabit, multiplicadores, sumadores, funciones MAC, etc. Estos bloques se han diseñado para ser lo más eficientes posible, en términos de consumo, silicio y rendimiento [1].

Al otro lado del espectro, se encuentran los ***Soft IP Cores***. Éstos son librerías de código de funciones de alto nivel, las cuales pueden ser usadas en diferentes diseños. Normalmente, estas funciones están codificadas usando un lenguaje de descripción hardware (HDL), tal como Verilog o VHDL [1].

Cada fabricante de FPGAs ofrece su propia selección de *Hard* y *Soft IP Cores*. Además, existen diferentes empresas u organizaciones que desarrollan *Soft IP Cores*. Muchos de estos *cores* son totalmente libres [1].

Además, uno también podría implementar su propio *Soft IP Core*. Esta implementación puede ser mucho más complicada que el diseño global en sí mismo.

En el diseño tratado en estas páginas se emplea una FPGA Virtex 5 (Xilinx), concretamente la XC5VLX50T-1FF665. Esta FPGA de Xilinx posee un *Hard IP Core* para llevar a cabo aplicaciones de Ethenet. A parte de este *Hard IP Core*, Xilinx también, proporciona *Soft IP Cores* para el desarrollo de aplicaciones Ethernet.

Para el diseño descrito aquí, se ha elegido usar un *Hard IP Core* por las siguientes ventajas:

1. Totalmente gratuito (*Soft IP Core* no)
2. Mayor eficiencia
3. No consume silicio[1]

[1] Hace referencia a que no emplea ninguna LUT

2.1 *Hard IP Core* Ethernet en Virtex 5

Como se ha comentado anteriormente, para la implementación de una aplicación de Gigabit Ethernet sobre FPGA se ha empleado un *Hard IP Core*, llamado *Embedded Tri-Mode Ethernet MAC Wrapper* proporcionado por Xilinx.

La familia de dispositivos Virtex 5 contienen un par de Ethernet MAC embebidas, las cuales son independientemente configurables. Acontinuación, se comentan las principales características de este bloque embebido.

La Ethernet MAC embebida en los dispositivos Virtex 5 cumple las especificaciones expuestas en IEEE 802.3. En la Figura 2-1 se muestra un diagrama de bloques de la Ethernet MAC (*core*). El bloque contiene dos Ethernet MAC compartiendo un interfaz host. Ambas son iguales.

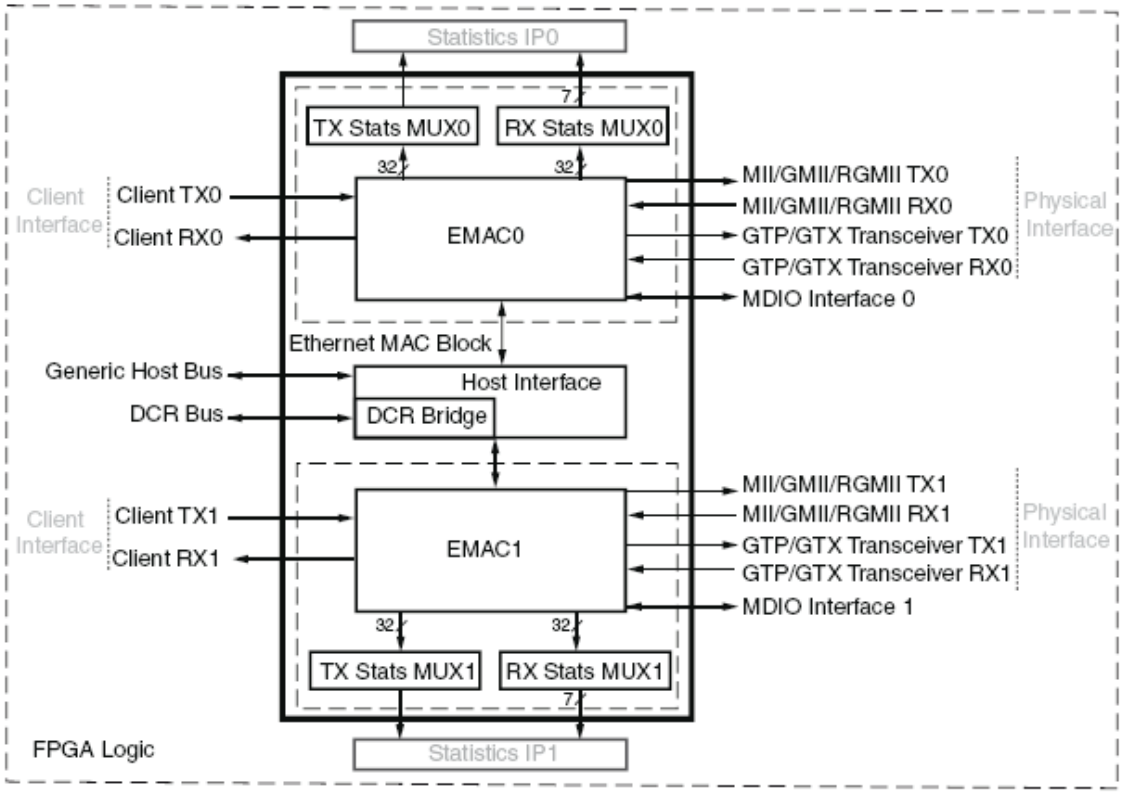


Figura 2-1. Bloque Ethernet MAC embebida [2]

Cada Ethernet MAC tiene un transmisor (TX) y receptor (RX) independientes, proporcionando una capacidad de transmisión full-duplex.

El interfaz host de la Ethernet MAC proporciona un acceso a los registros de configuración de ambas Ethernet MAC. A este interfaz puede accederse de dos formas, empleando el bus host genérico o el bus DCR, ver Figura 2-1.

El interfaz físico de cada Ethernet MAC puede ser configurado como MII, GMII, RGMII, SGMII o 1000BASE-X (ver sección 5.2). La Figura 2-1 muestra todos los posibles puertos de datos para el interfaz físico. Las Ethernet MAC pueden ser configuradas para tener diferentes interfaces.

Cada Ethernet MAC tiene un interfaz opcional *Management Data I/O (MDIO)*, que permite el acceso a los registros de control de un PHY o el acceso a los registros de control del interfaz físico dentro de la Ethernet MAC (usado sólo cuando es configurada en modo 1000BASE-X o SGMII).

Las aplicaciones típicas para la Ethernet MAC *core* incluyen:

- Puerto Tri-speed BASE-T Ethernet (10/100/1000)
- Puerto 1000BASE-X Ethernet

En las Figura 2-2 y Figura 2-3 se muestra un diagrama de bloques de estos dos tipos de aplicaciones.

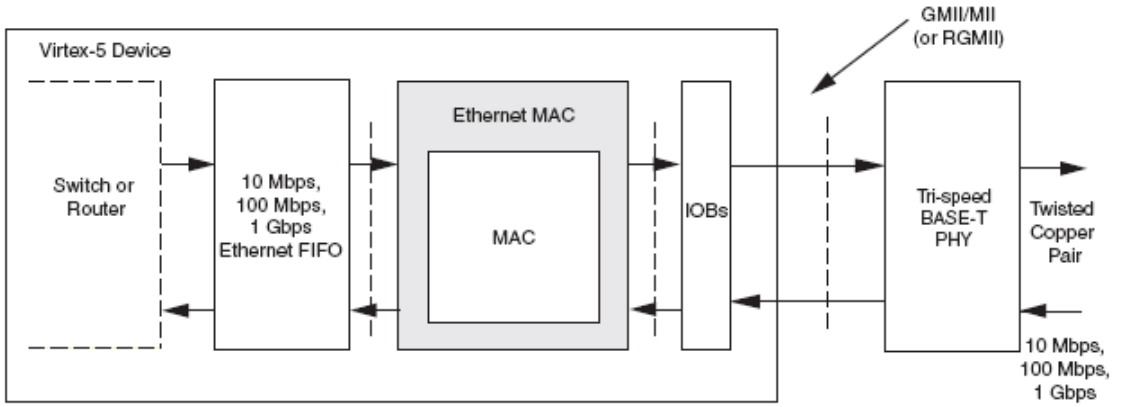


Figura 2-2. Aplicación 10/100/1000BASE-T típica [4]

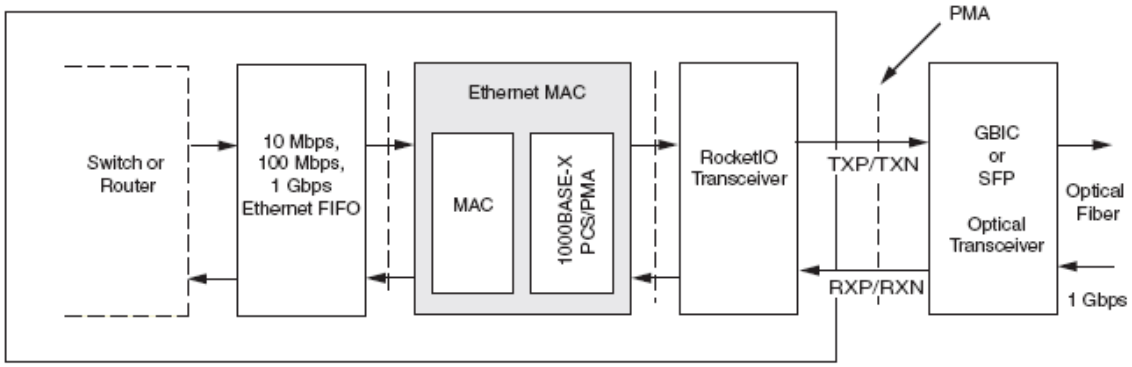


Figura 2-3. Aplicación 1000BASE-X típica [4]

La información anterior ha sido extraída de [2], [3] y [4]. Así que, para más información ver dichas referencias.

3 Interfaz físico

La Figura 3-1 ilustra la relación entre el modelo de referencia OSI y la Ethernet MAC, como es definida en la especificación IEEE 802.3. Las capas en gris muestran la funcionalidad que la Ethernet MAC lleva a cabo. También, la Figura 3-1 muestra donde los interfaces físicos encajan dentro de la arquitectura.

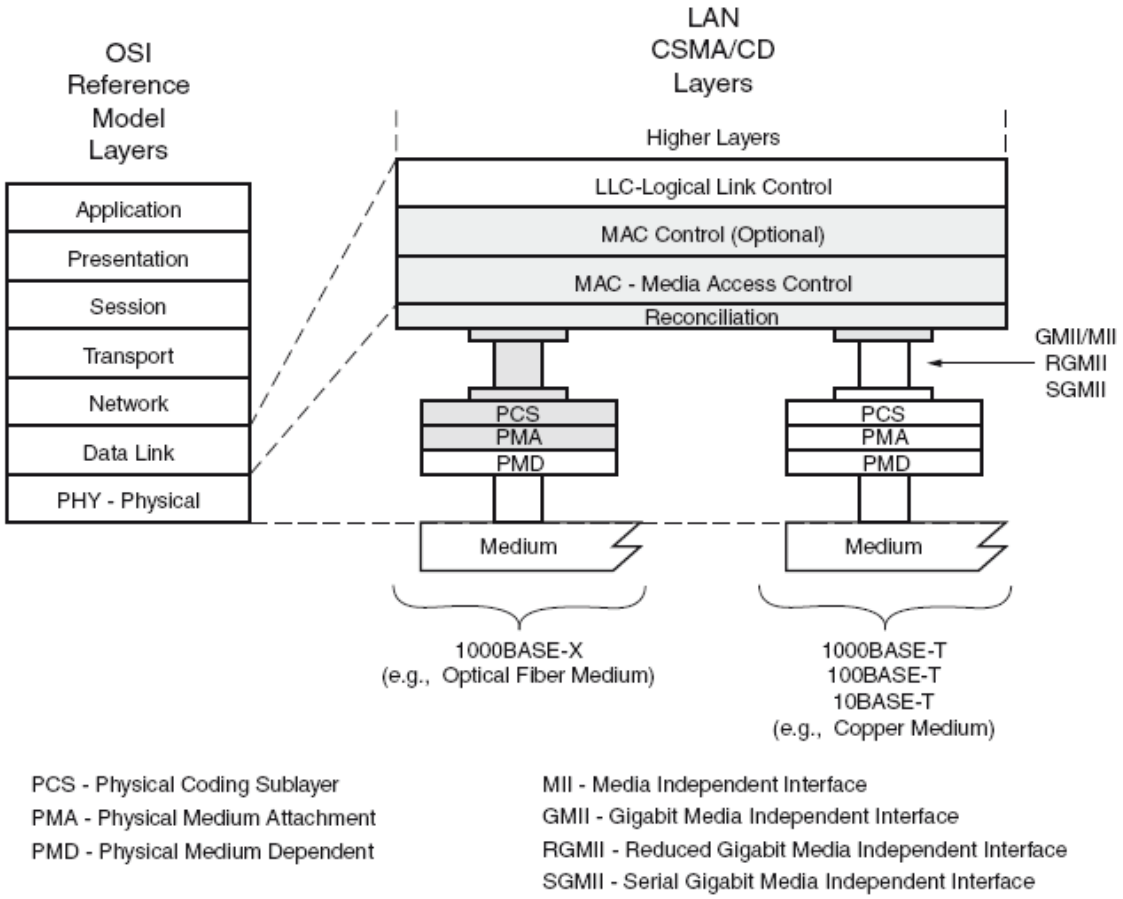


Figura 3-1. Modelo Ethernet IEEE 802_3.2002 [2]

La Ethernet MAC se define en la especificación IEEE 802.3, en las cláusulas 2, 3 y 4. Una MAC es responsable de los protocolos de entramado de Ethernet y detección de error de estas tramas. La MAC es independiente y puede conectarse a cualquier tipo de dispositivo de capa física.

La subcapa Control de MAC se define en la especificación IEEE 802.3, cláusula 31. Ésta proporciona la manipulación de control de flujo de tiempo real de la subcapa MAC.

Ethernet MAC, en todos los modos de operación, proporciona las subcapas Control MAC y MAC.

La combinación de la Subcapa de Codificación Física (PCS), la subcapa Vinculación al Medio Físico (PMA) y la subcapa Dependiente del Medio Físico (PMD) constituyen las capas físicas para el protocolo. Dos principales son especificadas:

- **BASE-T:** Los PHYs proporcionan un enlace entre la MAC y los medios de cobre. Esta funcionalidad no se ofrece dentro de la *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC*. Sin embargo, existen en el mercado dispositivos PHY BASE-T externos que pueden llevar a cabo esa función. Estos dispositivos pueden conectarse a la Ethernet MAC usando los interfaces GMII/MII, RGMII o SGMII.
- **BASE-X:** Los PHYs proporcionan un link entre la MAC y normalmente, medios de fibra óptica. La *Virtex-5 FPGA Embedded Tri.Mode Ethernet MAC* es capaz de soportar el estándar BASE-X 1 Gbps; Las subcapas 1000BASE-X PCS y PMA pueden ser ofrecidas conectando la Ethernet MAC a un *transceiver* serie RocketIO. Un *transceiver* óptico puede conectarse directamente al *transceiver* serie RocketIO para completar la funcionalidad de la subcapa PMD.

La información anterior ha sido extraída de [2], para más detalles, ver dicha referencia.

4 Formato de datos en Ethernet

Los datos Ethernet son empaquetados en tramas, tal y como se muestra en la Figura 4-1. Los diferentes campos de la trama son transmitidos de izquierda a derecha. Los bytes de dentro de la trama son trasmitidos de izquierda a derecha (del bit menos significativo al bit más significativo, a menos que se especifique lo contrario). La Ethernet MAC puede manejar tramas Ethernet Jumbo (tramas cuyo campo de datos es mayor de 1500 bytes).

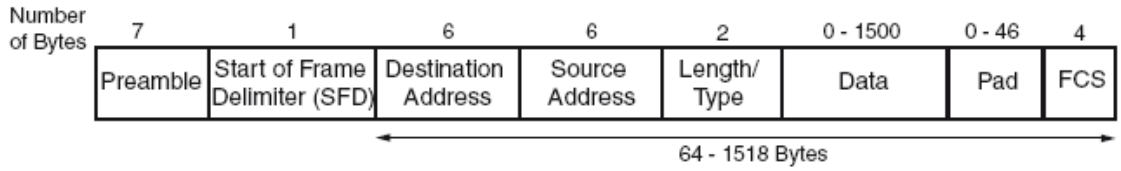


Figura 4-1. Formato de trama Ethernet estándar [2]

Nota: Los diferentes campos no son explicados en estás páginas, ya que quedan fuera del propósito de este documento. Si se necesita más información consultar [2] (referencia de donde se ha obtenido la información expuesta) o cualquier libro de redes, como por ejemplo, [5] ó [6].

4.1 Qué campos genera la Ethernet MAC y cuáles no

Algunos de los campos que componen la trama Ethernet son generados automáticamente por el *core*, mientras que otros, pueden ser proporcionados por el cliente o también, generados por el *core*. A continuación, se detallan las distintas opciones.

4.1.1 Preámbulo (*Preamble*)

Para transmisión, este campo es insertado automáticamente por la Ethernet MAC. En el caso de la recepción, este campo es eliminado de los datos de entrada antes de pasar éstos al cliente. La Ethernet MAC puede recibir datos incluso si el preámbulo no existe, mientras un inicio válido de trama esté disponible.

4.1.2 Delimitador de inicio de trama (SFD)

Para la transmisión en el interfaz físico, este campo es insertado automáticamente por la Ethernet MAC. Para la recepción, este campo es eliminado de los datos de entrada.

4.1.3 Dirección de destino (*Destination Address*)

Es el primer campo de la trama Ethernet. Este campo es proporcionado en los datos del paquete, en el caso de una transmisión. En recepción, este dato es guardado en el buffer del receptor.

La Ethernet MAC soporta la transmisión y recepción de paquetes unicast, multicast y broadcast.

4.1.4 Dirección de origen (*Source Address*)

En transmisión, el cliente debe proporcionar este dato. Mientras que en recepción, este dato es almacenado en el buffer del receptor.

4.1.5 Longitud/Tipo (*Length/Type*)

Para transmisión, la Ethernet MAC no lleva a cabo ningún procesamiento del campo longitud/tipo.

Para recepción, si este campo es un campo de longitud, la máquina de recepción de la Ethernet MAC interpreta este valor y elimina cualquier relleno en el campo de relleno (si es necesario). Si el campo es un campo de longitud y la comprobación longitud/tipo está habilitada, la Ethernet MAC compara la longitud con la longitud del campo de datos actual e indica un error si estas longitudes no son iguales. Si el campo es un campo de tipo, la Ethernet MAC ignora el valor y lo pasa junto con los datos del paquete sin más procesamiento. Este campo siempre es guardado en el buffer del receptor.

4.1.6 Datos (*Data*)

El campo de datos puede variar de 0 a 1500 bytes de longitud para una trama normal. La Ethernet MAC puede manejar tramas jumbo de cualquier longitud (es decir, mayores de 1500 bytes).

Este campo siempre es proporcionado en los datos del paquete para transmisiones y siempre es almacenado en los datos del paquete de recepción.

4.1.7 Relleno (*Pad*)

En el caso de la transmisión, este campo puede ser insertado automáticamente por la Ethernet MAC o puede ser proporcionado por el cliente. Si es insertado por la Ethernet MAC, el campo FCS es calculado e insertado por la Ethernet MAC. Si es proporcionado por el cliente, el FCS puede ser proporcionado por la Ethernet MAC o por el cliente.

En el caso de recepción, si el campo longitud/tipo se interpreta como longitud, cualquier campo de relleno en la trama de entrada no es pasado al cliente, a menos que la Ethernet MAC esté configurada para que el campo FCS pase al cliente.

4.1.8 Secuencia de chequeo de trama (FSC)

Para transmisión, este campo puede ser insertado automáticamente por la Ethernet MAC o proporcionado por el cliente.

Para recepción, el valor de entrada FCS es verificado en cada trama. Si se recibe un valor incorrecto de FCS, la Ethenet MAC indica al cliente que se ha recibido una trama incorrecta. El campo FCS puede ser pasado al cliente o ser eliminado por la Ethernet MAC.

5 Configuración de la Ethernet MAC

A continuación se muestran dos configuraciones diferentes, ya que cuando se comenzó este diseño, la tarjeta para la cuál se estaba implementando este *firmware* no estaba diseñada. Por tanto, en su lugar, se empleó una placa de desarrollo, ML505 de Xilinx [7] (ver Figura 5-1), cuya FPGA es casi idéntica a la FPGA de la tarjeta final (FEC). La única diferencia es el encapsulado. La placa de desarrollo contiene una FPGA XC5VLX50T-1FF1136, mientras que la tarjeta final posee una FPGA XC5VLX50T-1FF665.

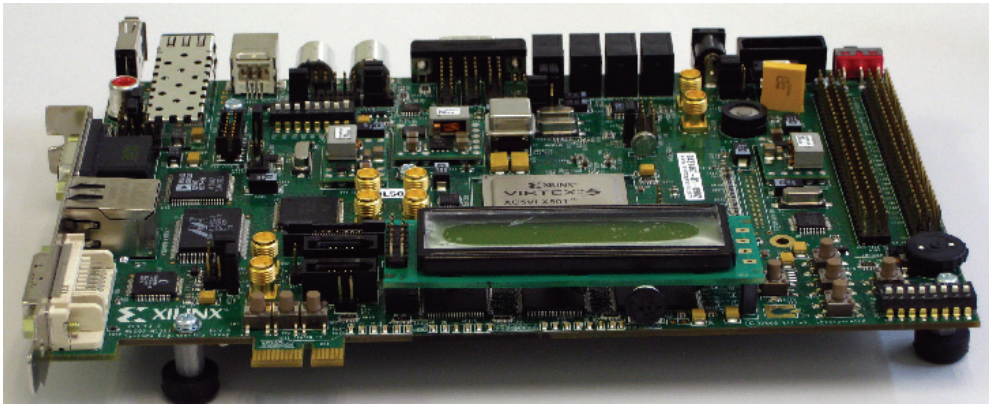


Figura 5-1. Tarjeta de desarrollo ML505 Xilinx

No tener disponible la tarjeta objeto desde el comienzo del diseño del *firmware* (o una versión) puede ser un problema, ya que las pruebas que se llevan a cabo no son totalmente reales, por decirlo de algún modo. En este caso, no fue un problema, sino una «ventaja», ya que se pudieron probar diferentes opciones antes de fabricar la tarjeta final, lo cual supuso un ahorro económico importante.

De todas las muchas y variadas opciones que la Embedded Tri-Mode Ethernet MAC ofrece se han elegido las siguientes opciones:

- BASE-T usando interfaz SGMII / BASE-X PCS PMA
- 1 Gbps
- Bus de datos del cliente 8 bits
- Sin interfaz host
- Sin MDIO
- Tramas Jumbo

Todas estas opciones son elegidas cuando uno se dispone a generar el *core* (ver [3]).

Como se puede observar en la lista anterior, hay dos tipos de configuraciones. Una para llevar a cabo una aplicación 1000BASE-T y otra, para realizar una aplicación 1000BASE-X. Ambas fueron chequeadas empleando la placa de desarrollo y ambas, funcionaron correctamente. En la tarjeta final, se decidió emplear 1000BASE-X, ya que posee una ventaja frente a 1000BASE-T. Para el enlace se puede emplear tanto cobre como fibra

óptica. Además, no es necesario el chip físico, el cual proporciona un enlace entre la MAC y los medios de cobre, lo que facilita el diseño del PCB. Por el contrario, es necesario un adaptador para conectar un cable de cobre en el conector SFP, como el mostrado en la Figura 5-2 [8].

Figura 5-2. *Transceiver* SFP 1000BASE-T 3Com

En las siguientes secciones se exponen las razones de por qué las opciones anteriores han sido elegidas y no otras.

5.1 BASE-T o BASE-X

En primer lugar, se pensó en utilizar BASE-T, ya que es más fácil trabajar con cobre que con fibra óptica, sobre todo cuando uno se dispone a hacer pruebas. Más tarde, se descubrió la existencia del estándar STP 1000BASE-T, conector mostrado en la Figura 5-2, lo que hizo que la opción elegida final fuese BASE-X. Empleando la misma conexión (SFP) y circuitos de la PCB, se puede usar tanto cobre como fibra óptica.

5.2 Interfaz SGMII

Cuando se quiere usar 1000BASE-T hay dos opciones, GMII o SGMII. Se prefirió emplear el interfaz SMII, ya que es un interfaz serie, por tanto emplea menos pines I/O de la FPGA que GMII (interfaz paralelo). Además, SGMII es «compatible» con 1000BASE-X. Ambos son interfaces serie, por lo tanto, la lógica que acompaña al *core* será la misma cuando se emplee 1000BASE-T o 1000BASE-X. Algo a tener muy en cuenta, ya que supone un ahorro significativo en tiempo de implementación.

5.3 1 Gbps

Se quería obtener el mayor *throughput* possible. Por lo tanto, se eligió la máxima velocidad ofrecida por el *core*, 1 Gbps.

5.4 Bus de datos del cliente

En este caso, la opción de 8 bits es trivial, ya que es la única común empleando 1000BASE-T y 1000BASE-X. De esta forma, no habrá que cambiar la lógica que acompaña al *core*.

5.5 Tramas Jumbo

Varios tests fueron llevados a cabo en el CERN junto con Filippo Costa (él pertenece al grupo de DATE dentro del experimento ALICE) usando un simulador software con el propósito de obtener la trama ideal. Es decir, el tamaño de trama para alcanzar el máximo *throughput* posible. De acuerdo con las puebas realizadas, el tamaño de trama ideal es 9000 bytes [9]y [10].

Tener en cuenta, que si se aumenta el tamaño de trama, el *throughput* será mayor pero habrá que aumentar el tamaño de las memorias implementadas dentro de la FPGA, para procesar tramas mucho más grandes. Además, la ganancia conseguida aumentado el tamaño de trama por encima de 9000 bytes, es muy pequeña. Pues con 9000 bytes, el *throughput* conseguido está muy cerca del máximo, 125 MB/s (ver Figura 11-2 a y Figura 11-3, aunque en estos casos es un poco menor, ya que depende del número de tramas que contiene el evento).

6 Formato de las tramas

El protocolo UDP es el protocolo elegido para el envío de tramas desde las tarjetas FEC a DATE (o viceversa). Una de las razones de la elección de este protocolo es su fácil implementación con respecto al protocolo TCP/IP. Además, posee otras ventajas y también desventajas, que se comentarán a continuación.

6.1 Protocolo de datagrama de usuario

El protocolo de datagrama de usuario, UDP (*User Datagram Protocol*), es un protocolo de la capa de transporte no orientado a conexión y no seguro. Es un protocolo muy simple que proporciona solamente dos servicios más que los que proporciona IP: demultiplexación y comprobación de errores en los datos. Hay que recordar que IP sabe cómo entregar paquetes a una computadora, pero no sabe cómo entregarlos a una aplicación específica en la computadora. Hay que recordar también, que IP sólo comprueba la integridad de su cabecera. UDP puede opcionalmente comprobar la integridad del datagrama UDP completo [5].

Por lo tanto, una aplicación funcionando bajo UDP debe tratar directamente con los problemas de comunicación que un protocolo orientado a conexión habría solucionado, p. e.: retransmisión para entrega segura, control de flujo, evitar congestiones, etc, cuando éstas con requeridas [11].

El formato del *Header* UDP se muestra en la Figura 6-1.

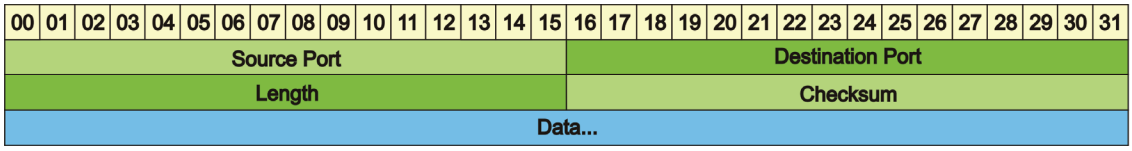


Figura 6-1. *Header* UDP

A continuación se explican brevemente los diferentes campos que componen el datagrama UDP.

- **Puerto Origen (Source Port):** Identifica la aplicación particular en la computadora origen que recibirá las respuestas.
- **Puerto Destino (Destination Port):** Permite al módulo UDP demultiplexar los datagramas a la aplicación correcta en una computadora.
- **Longitud (Length):** Identifica el número de bytes en el datagrama UDP, incluyendo las cabeceras y los datos.
- **Suma de Comprobación (Checksum):** Detecta errores en el datagrama y su uso es opcional.
- **Datos (Data):** Datos a enviar

Como se ha comentado al inicio de la sección, el uso del protocolo UDP fue elegido porque es fácil de implementar en una FPGA (respecto a otras opciones, como TCP/IP).

Además, ofrece mayor velocidad en las conexiones punto a punto. Sin embargo, no ofrece control de flujo. Por lo tanto, habrá que implementar un algoritmo de control de flujo (ver sección 26).

6.2 Formato de la trama a enviar

Como se mencionó en la sección 5.5, el tamaño de trama elegido es de 9000 bytes y, teniendo en cuenta que el protocolo UDP es usado para transmisión/recepción de tramas, las tramas a enviar tendrán un aspecto como el mostrado en la Figura 6-2.
Como se ilustra en esta figura, la trama contiene un campo llamado *Counter*. Este campo es un contador de 32 bits empleado para chequear si durante la transmisión de datos algún paquete fue duplicado o perdido. Mediante este campo y el campo *checksum* se puede saber si la trama transmitida o recibida es válida o no. Si la trama no fuese válida, sólo se sabría que dicha trama no es válida, ya que no se dispone de ningún algoritmo para reenviar una trama cuando en recepción se ha detectado que no es válida. En este sistema se puede permitir esta pequeña probabilidad de error de trama porque los datos no son críticos. Además, según las pruebas realizadas, la probabilidad de que ocurra un error un muy baja (ver sección 11.2).

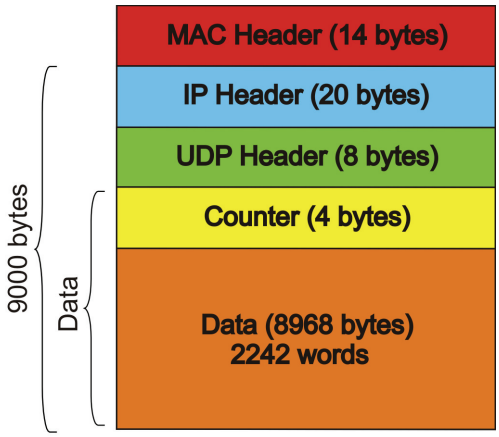


Figura 6-2. Formato de trama

Nota: Los campos *MAC Header* e *IP Header* no son explicados en este documento, por quedar fuera del objetivo de éste. Para más detalles ver [5], [6] y [11].

De acuerdo a lo descrito en el párrafo anterior, cuando la tarjeta FEC envíe los datos digitalizados a DATE, deberá hacerlo empleando este tipo de tramas. Los datos digitalizados se envían por eventos. Un evento está formado por un número determinado de tramas como la mostrada en la Figura 6-2 y una trama que indica final de evento, tal y como se muestra en la Figura 6-3. La finalidad de esta trama es indicar a DATE el final de un bloque de datos. Sin esta trama DATE no sabría cuando un bloque de datos comienza o termina. Además, dicha trama es muy útil para sincronizar varias tarjetas FEC conectadas a DATE. Esta sincronización se explica con detalle en la sección 7.2.

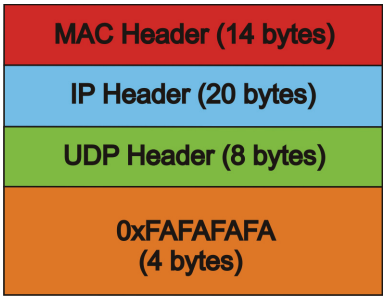


Figura 6-3. Trama final de evento

Por tanto, un evento tendrá un aspecto similar al mostrado en la Figura 6-4. En este caso, el evento está formado por tres tramas de 9000 bytes. Cabe destacar, que el número de tramas por evento es configurable y que el tamaño de trama puede ser variable. En la sección 9.4, se detallan qué parámetros pueden ser configurados. De esta forma, el sistema puede ser adaptado a unas necesidades dadas.

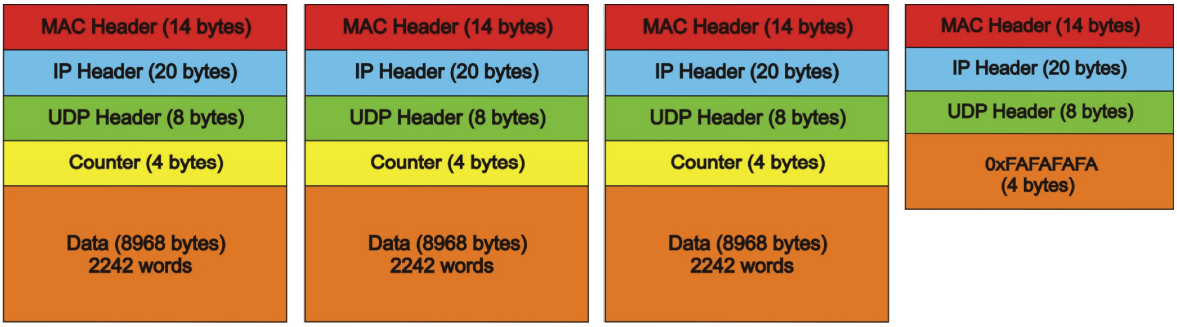


Figura 6-4. Evento formado por tres tramas de 9000 Bytes

7 Comunicación tarjeta FEC - DATE

En primer lugar, antes de transmitir información entre la tarjeta FEC y DATE, este último debe establecer la comunicación con la tarjeta FEC. Para llevar a cabo esta tarea, DATE envía una trama ARP a la tarjeta (ver sección 9.2). Si ésta esta no responde, DATE vuelve a enviar de nuevo otra trama ARP, así hasta que la tarjeta FEC responde con otra trama ARP. En este momento, cuando DATE recibe la trama ARP desde la tarjeta FEC, la comunicación queda establecida entre ambos puntos. Ahora, la tarjeta FEC puede enviar tramas de datos. Un diagrama de este proceso se muestra en la Figura 7-1.

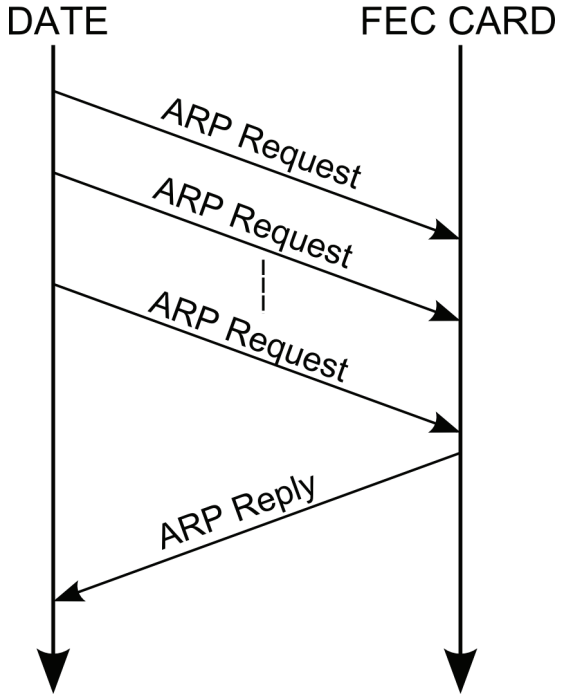


Figura 7-1. Establecimiento de conexión entre DATE y la tarjeta FEC

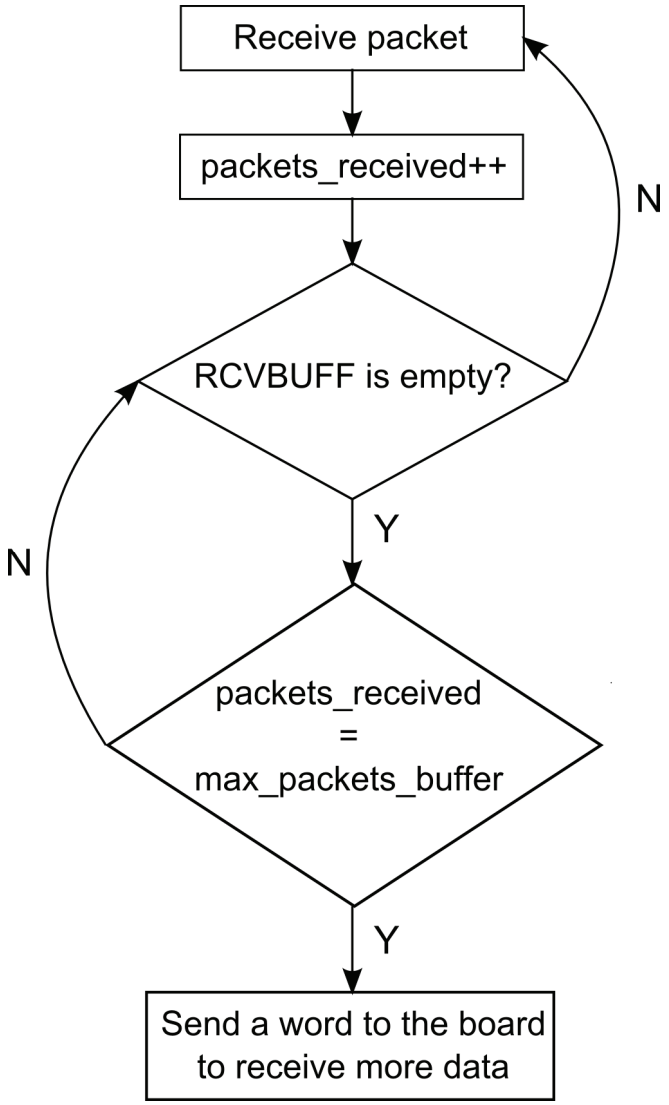
Hasta aquí todo perfecto. Sin embargo, cuando la tarjeta FEC transmite datagramas al máximo *throughput*, el buffer de la máquina donde DATE está corriendo puede ser saturado, y como consecuencia, muchos paquetes se pierden. Esto ocurre porque, como ya se sabe, el protocolo UDP no soporta control de flujo. Por tanto, será necesario implementar un algoritmo de control de flujo para evitar que el buffer de DATE se desborde y así, evitar la pérdida de paquetes en la transmisión.

Otro punto a tener en cuenta respecto al desbordamiento del buffer, es la optimización de determinadas variables del *kernel* y de la tarjeta de red. Este ajuste es muy necesario cuando se trabaja a velocidades de 1 ó 10 Gbps. Este tema será tratado en la sección 10.

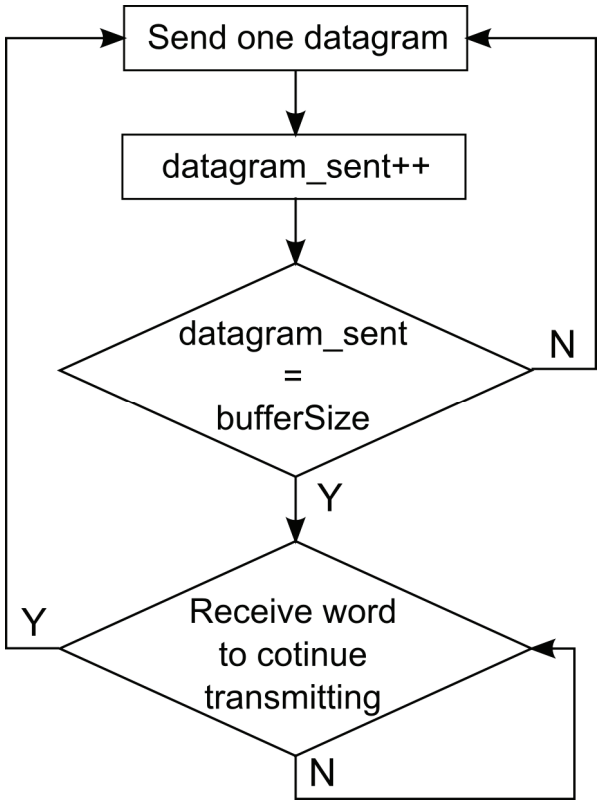
7.1 Control de flujo

En esta sección se pretende explicar el algoritmo empleado para evitar que el buffer de DATE se desborde. Este algoritmo consiste en pausar la transmisión cuando se ha enviado un número determinado de datagramas. Este número sería el número máximo de tramas que pueden ser enviadas sin llegar a saturar el buffer. Una vez que el buffer se ha vaciado, DATE avisa a la tarjeta FEC para indicar que puede continuar transmitiendo.

La idea es sencilla, pero uno puede preguntarse qué ocurre si el tamaño de trama varía (el número de tramas necesario para saturar el buffer también varía) o si las variables del *kernel* del sistema operativo están ajustadas de diferente forma en diferentes máquinas. Pues no hay ningún problema, ya que el *firmware* se adapta a estas variaciones. El parámetro que controla el tamaño del buffer es dinámico, así que, éste puede ser ajustado antes de iniciar DATE.



(a)



(b)

Figura 7-2. Algoritmo empleado para el control de flujo, (a) Software y (b) *Firmware*

En la Figura 7-2 se muestra el algoritmo empleado como control de flujo, tanto en la parte del software (DATE) como en la parte del *firmware* (tarjeta FEC). Ambos algoritmos deben funcionar de forma paralela, es decir, cuando el contador que controla las tramas enviadas en el *firmware* vale x, el contador que controla las tramas recibidas en el software debe tomar el mismo valor. Así, cuando ambos contadores alcancen el tamaño del buffer, el *firmware* parará la transmisión y quedará a la espera y, el software esperará a que el buffer se vacíe. Una vez que el buffer se ha vaciado, el software enviará una palabra a la FEC para indicar que puede continuar transmitiendo. El datagrama que contiene esta palabra es similar al datagrama de final de evento (Figura 6-3). En este caso, la palabra que contiene el campo de datos es 0xAAAAAAAA, en vez de 0xFAFAFAFA.

7.2 Conectando dos o más tarjetas FEC a DATE

DATE soporta hasta cuatro tarjetas de red por ordenador. Por lo tanto, se pueden adquirir datos de cuatro fuentes de datos (en este caso, tarjetas FEC) al mismo tiempo. Para que esto pueda llevarse a cabo, debe haber una sincronización entre dichas fuentes de datos. Actualmente (todo el sistema está en fase de pruebas), esta sincronización se realiza de la siguiente forma:

DATE usa un contador local de eventos recibidos para cada fuente de datos. También, usa un contador global de eventos, el cual es incrementado cuando DATE recibe información de todas las fuentes de datos.

¿Qué ocurre si el tamaño de los eventos es diferente en las fuentes de datos?

Si un equipo (fuente de datos) es más rápido que otro (porque ésta enviando eventos más pequeños que el otro equipo), éste puede enviar más eventos que el otro. Por ejemplo, teniendo en cuenta que hay dos equipos conectados a DATE:

- Equipo A envía 2 eventos por segundo
- Equipo B envía 1 evento por segundo

Cuando A está esperando la palabra desde DATE para continuar enviando datos, DATE chequeará cuántos eventos ha enviado A ya. Si el número de eventos enviados por A es mayor que el contador global de eventos (incrementado por DATE sólo cuando recibe eventos de A y de B) DATE no envía la palabra a A para indicar que puede seguir transmitiendo. De esta forma, B tendrá tiempo para enviar el resto de eventos. Normalmente, esta situación no se da tan drásticamente, ya que las fuentes de datos están sincronizadas mediante un *trigger*. Esto es una protección.

DATE sólo envía datos a las tarjetas FEC (en la fase de adquisición de datos, no en *slow control*) cuando:

- El buffer del ordenador donde está corriendo DATE está vacío
- El número de paquetes recibidos es igual al número de paquetes que puede ser almacenado en el buffer
- El contador local de eventos toma es mismo valor que el contador global de eventos

8 Usando el enlace de 1 Gbps como *slow control*

El objetivo principal del enlace Gigabit Ethernet es transmitir los datos digitalizados por la tarjeta FEC a DATE. Además de realizar esta función, se puede aprovechar el enlace para configurar el sistema. Es decir, ajustar todos los parámetros de la tarjeta FEC y también, de las tarjetas conectadas a ésta, etc. Como por ejemplo, ajustar ADCs, obtener valores de temperatura, voltaje, etc.

Hay varias opciones para implementar dicho control. La primera de ellas, es utilizar los puertos de la trama UDP. Es decir, asociar un comando a cada puerto. La otra opción, es asignar a cada dirección de una memoria un comando.
Se ha decido emplear esta última opción porque es más genérica y además, permite ser ampliada cuanto uno quiera. Al final, sólo se trata de escribir y leer datos de una memoria.

La última idea desarrollada y testeada se ilustra en la Figura 8-1. Esta sería la forma de configurar los distintos registros del sistema completo, enviando una o varias tramas como la mostrada en la Figura 8-1.

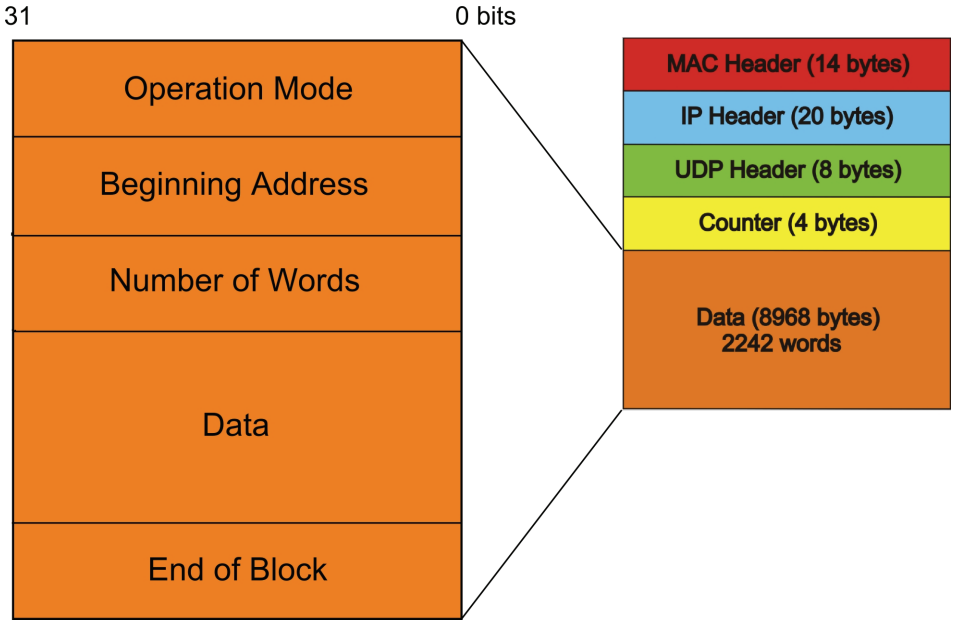


Figura 8-1. Formato del datagrama para *slow control* (el campo contador es opcional)

DATE podría leer los datos de un fichero de configuración y generar las tramas UDP correspondientes (pendiente de implementar).

A continuación se muestra un ejemplo.

En primer lugar, se definen unos registros, por ejemplo, *Reset*, *Size of Frame* y *Number of Frames per Event*. Estos registros se asocian con una dirección de una memoria RAM. De manera que, cuando se cambie el valor de una determinada dirección de memoria, se cambia el valor del registro.

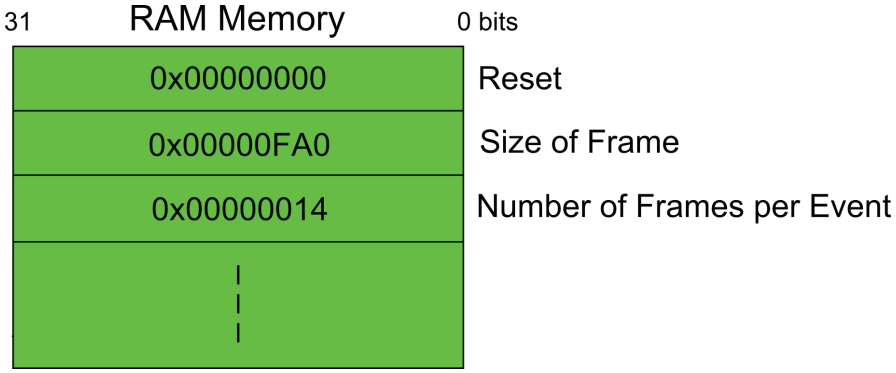


Figura 8-2. Memoria RAM asociada a registros

Según la Figura 8-2, el registro Reset (*Reset*) equivale a la dirección cero, el registro Tamaño de Trama (*Size of Frame*) a la dirección uno y el registro Número de Tramas por Evento (*Number of Frames per Event*) a la dirección dos. El valor actual de estos registros es el mostrado en la Figura 8-2.

Si la tarjeta FEC recibe el datagrama representado en la Figura 8-3, ¿cómo lo interpreta?

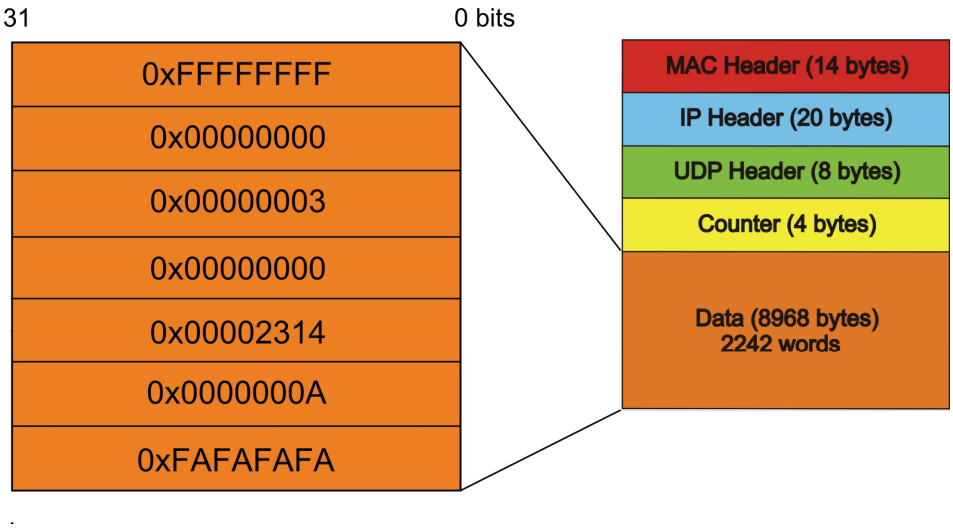


Figura 8-3. Datagrama de configuración de registros

Considerando la Figura 8-1, el primer campo que se recibe es el modo de operación, que en este caso vale 0xFFFFFFFF, lo cual indica que es una operación de escritura (0xFFFFFFFFF se definió como operación de escritura y 0x00000000 como operación de lectura). El segundo campo que se recibe es la dirección inicial, que toma el valor cero. El siguiente campo es el número de palabras (o registros que quieren configurarse), que vale tres. Por tanto, los registros que quieren configurarse son los que se encuentran en la dirección cero, uno y dos. Según la Figura 8-2, estos registros se corresponden con Reset (*Reset*), Tamaño de Trama (*Size of Frame*) y Número de Tramas por Evento (*Number of Frames per Event*).

Los registros no serán actualizados hasta que no se reciba la palabra final de bloque (0xFAFAFAFA). Por tanto, si ésta no se recibe dichos registros no serán actualizados y mantendrán su valor anterior. Si la palabra de final de bloque es recibida, el contenido de la memoria quedará tal y como se ilustra en la Figura 8-4.

En el caso de una operación de lectura se procede de una forma similar.

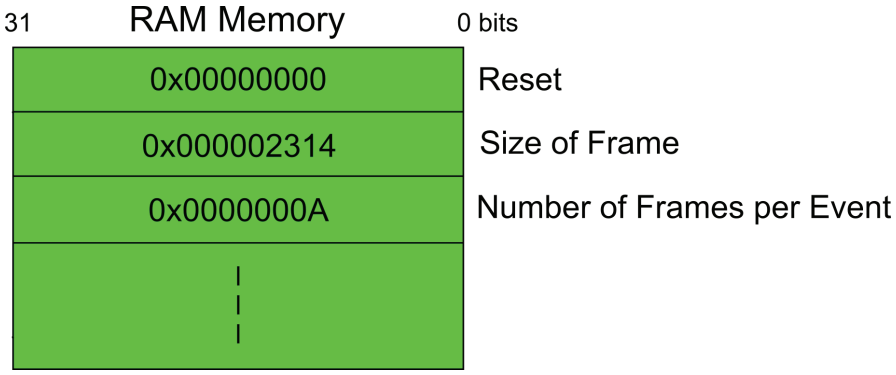


Figura 8-4. Memoria RAM asociada a registros después de recibir la trama mostrada en la Figura 8-3

Esta estructura permite definir diferentes bloques de registros, tales como críticos, básicos, comunes,… La configuración de estos bloques se haría tal y como se ha explicado en las líneas anteriores, ya que los registros de un mismo bloque se sitúan de forma contigua.
Esta forma de configurar los registros es muy versátil, permite modificar bloques de registros y hasta un solo determinado registro, tal y como se muestra la Figura 8-5. En este caso, el registro 0x300 toma el valor 0x24.

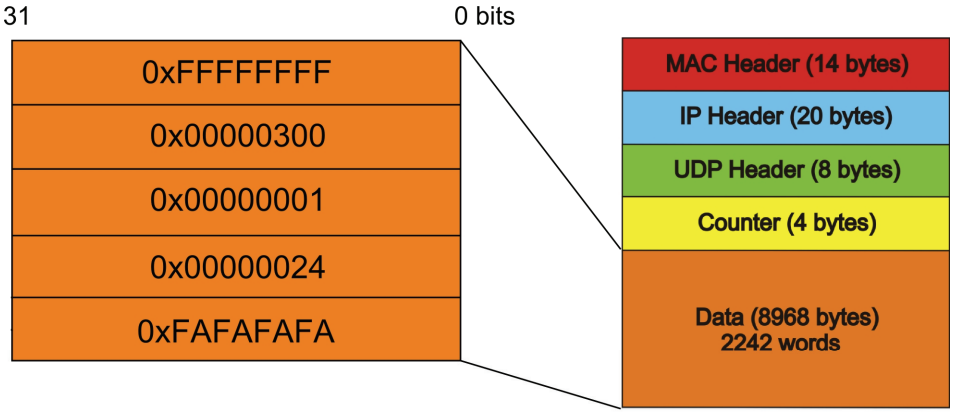


Figura 8-5. Trama de configuración para modificar un sólo registro

Otra de opciones es restringir la escritura de determinados registros a determinados usuarios. Ya que hay que evitar que cualquier usuario pueda cambiar registros críticos. Hay dos opciones, prohibir desde DATE la modificación de algunos registros, o introducir un palabra en el datagrama a modo de contraseña, cuando se quiera cambiar algún registro crítico.

9 Estructura del código HDL

Como se ha comentado a lo largo del texto se quiere implementar una aplicación capaz de enviar y recibir tramas UDP, además de llevar a cabo otras funciones, como establecer la comunicación entre los dos puntos del enlace empleando tramas ARP. Como es esperar, todas estas funciones no las proporciona el *core*. Así que, habrá que implementar una determinada lógica que lleve a cabo el resto de las funciones necesarias.

En las siguientes secciones se expondrán las funciones que se obtienen cuando se instancia el *core* y, se explicará la lógica adicional necesaria para realizar el resto de funciones que no proporciona el *core*.

9.1 Qué proporciona el *core*

En primer lugar, se genera el *core* (*CORE Generator*) con la configuración deseada (ver sección 5). Una vez generado se obtiene un directorio llamado *example_design*, en el cual se encuentran los ficheros Verilog o VHDL del diseño [3]. En la Figura 9-1 se muestra un diagrama de bloques del ejemplo proporcionado por el *core*. Este ejemplo está listo para ser usado. Además, permite comprobar de forma sencilla si el enlace de red funciona correctamente.

El ejemplo de diseño es un bucle, es decir, todos los datagramas recibidos por la FPGA son devueltos al origen. Esto se consigue intercambiando las direcciones origen y destino de las tramas recibidas. Esta la función la lleva a cabo el fichero llamado `address_swap_module_[8 | 16].v[hd]`.

Por tanto, a partir de este ejemplo se comprueba que el *core* proporciona cada una de las palabras de la tramas y nada más. Es decir, si se quiere enviar una trama determinada, en primer lugar habrá que construirla y luego pasársela al *core* de byte en byte (depende de cómo se haya configurado el *core*). En la Figura 9-2 se ilustra la transferencia de una trama de cinco bytes, donde las señales `src_rdy_n` y `dst_rdy_n` son usadas como control de flujo de los datos. Cuando estas señales están activas simultáneamente, el dato es pasado del origen al destino. Las señales `sof_n` y `eof_n` indican inicio de trama y final de trama, respectivamente. Como se puede observar, en este caso, la transmisión se realiza de byte en byte (si se tratase de una recepción de trama el proceso llevado a cabo es similar al expuesto).

Según lo expuesto en los párrafos anteriores, si se quiere enviar tramas UDP, ARP,… debe implementarse una lógica adicional que se encargue de construir estas tramas. Este es el objetivo de la siguientes secciones, explicar la lógica adicional necesaria y cómo conectarla con el *core*.

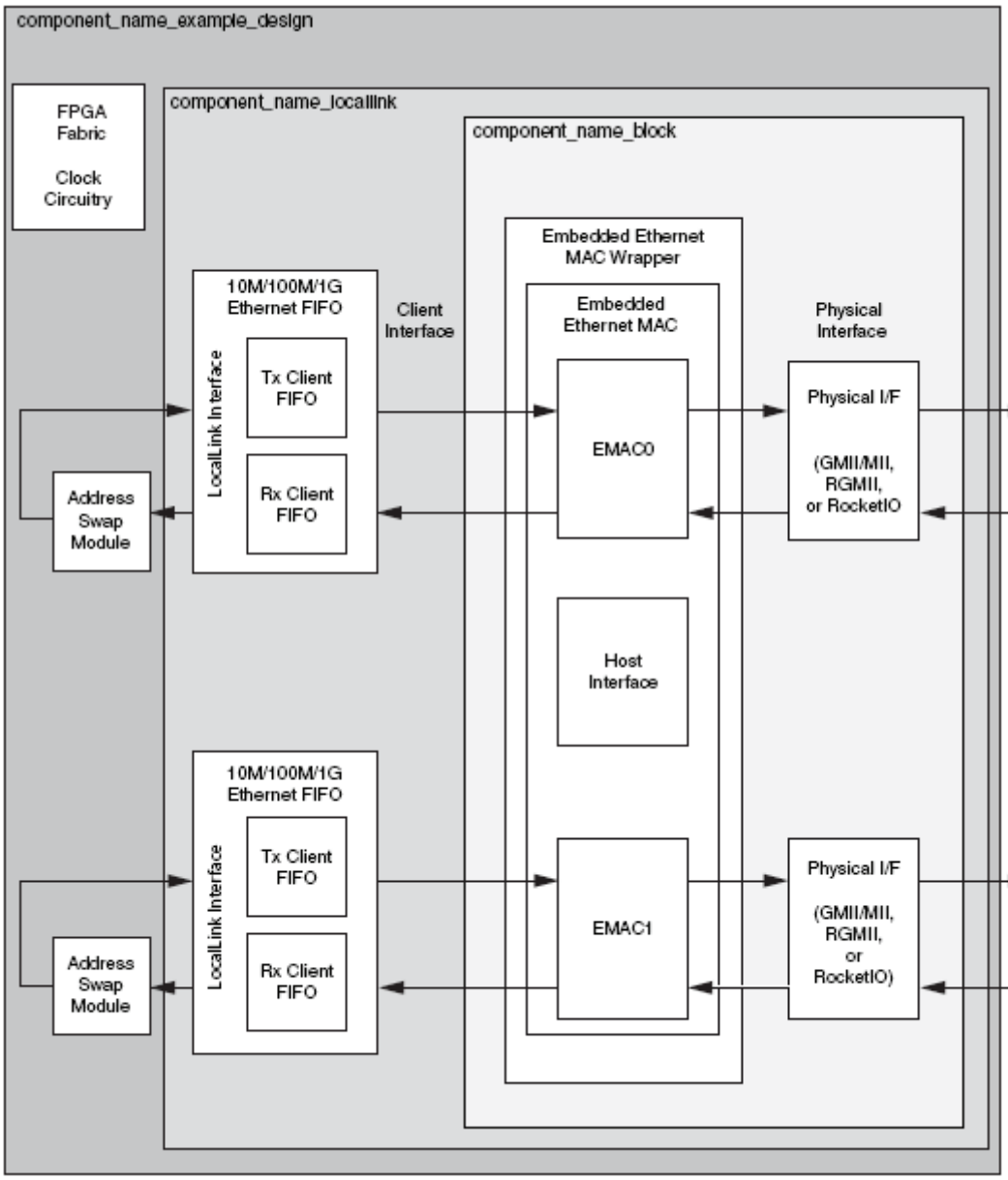


Figura 9-1. Diseño ejemplo HDL (puedo contener una o dos MAC) [3]

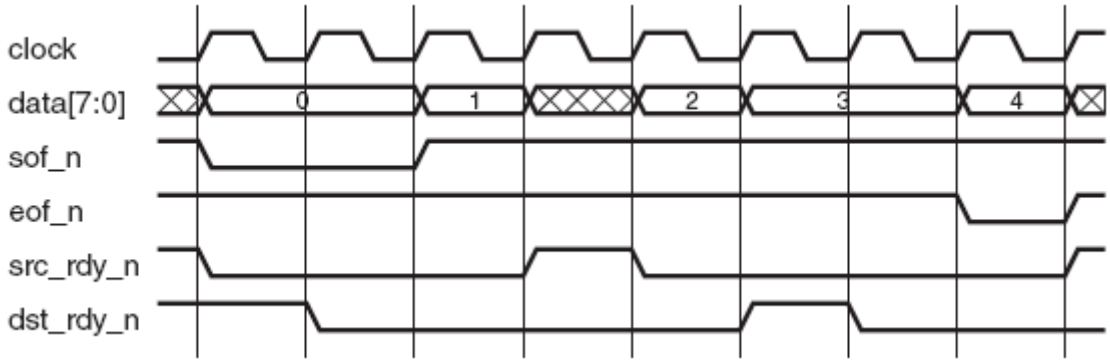


Figura 9-2. Transferencia de trama con control de flujo [3]

9.2 Lógica adicional

Antes de comenzar con la explicación de la estructura del código, se detallan brevemente las funciones que desean implementarse.

El sistema en su conjunto (*core* más la lógica adicional) debe ser capaz de enviar y recibir tramas UDP, ARP e ICMP.

Las tramas **UDP** fueron explicadas en la sección 6.1. Aquí que, si se desea información relacionada con este tema ir a dicha sección.

ARP son las siglas en inglés de *Address Resolution Protocol* (protocolo de resolución de direcciones). Es un protocolo de nivel de red responsable de encontrar la dirección hardware (*Ethernet MAC*) que corresponde a una determinada dirección IP. Para ello se envía un paquete (*ARP request*) a la dirección de difusión de la red que contiene la dirección IP por la que se pregunta, y se espera a que esa máquina (u otra) responda (*ARP reply*) con la dirección Ethernet MAC que le corresponde.

El protocolo de mensajes de control de Internet o **ICMP** (*Intenet Control Message Protocol*) es el subprotocolo de control y notificación de errores del protocolo de Internet (IP). Como tal, se usa para enviar mensajes de error, indicando por ejemplo que un servicio determinado no está disponible o que un router o host no puede ser localizado.
En este caso, este protocolo se utiliza para llevar a cabo la función *ping*. Esta función comprueba el estado de la conexión con uno o varios equipos remotos por medio de los paquetes de solicitud de eco y respuesta de eco para determinar si un sistema IP específico es accesible en una red. Es útil para diagnosticar los errores en redes o enrutadores IP.
Muchas veces se utiliza para medir la latencia o tiempo que tardan en comunicarse dos puntos remotos.

¿Cómo implementar los protocolos anteriores en código HDL?

En mi opinión, pienso que una de las mejores formas de implementar dichas funciones es seguir la estructura de capas del modelo OSI [6]. Esta idea fue obtenida de [12]. Ahí, se expone la implementación de un diseño similar al explicado en este documento. Con la excepción de que, esa implementación trabaja a una velocidad máxima de 10 Mb/s y la presente a 1 Gbps. Algo a tener muy en cuenta, ya que en este caso, la frecuencia de reloj en bastante elevada, 125 MHz (cada 8 ns un byte es enviado o recibido a los *transceivers* serie). Siempre que se trabaja con frecuencias elevadas en diseños grandes y complejos, uno debe ser muy cauto a lo hora de emplazar la lógica en la FPGA. Si no, será difícil alcanzar frecuencias elevadas al sintetizar el diseño. Además, se limita el tiempo de procesado de los datos. Es decir, los deben ser procesados consumiendo los mínimos ciclos de reloj, ya que si se consumen demasiados ciclos de reloj en procesar los datos, el valor del *throughput* puede disminuir considerablemente.

En la Figura 9-3 se muestra un diagrama de bloques del código VHDL implementado.

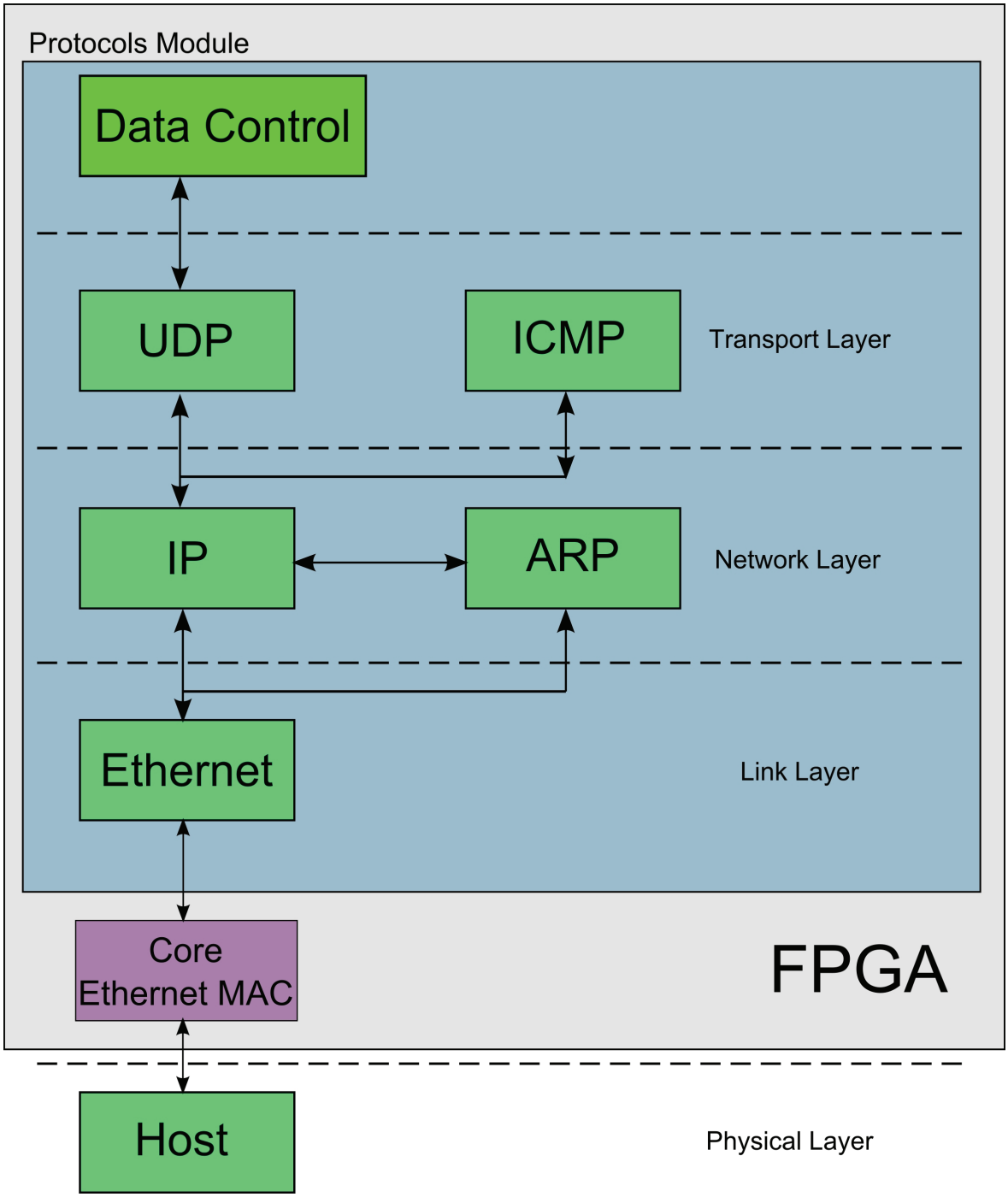


Figura 9-3. Lógica adicional HDL (aunque los mensajes ICMP se encapsulan en paquetes IP, se considera que ICMP está en la misma capa IP [5]. Sin embargo, aquí se ha situado en la capa superior para conseguir una mejor estructuración del código)

Todos los protocolos que forman el denominado módulo *Protocols Module*, excepto el protocolo ICMP, se dividen en dos bloques, transmisor y receptor. Por ejemplo, el bloque Ethernet se divide en *EthernetSender* y en *EthernetReceiver*, tal y como se ilustra en la Figura 9-4. Los bloques denominados *Sender* se encargar de desencapsular la trama, es decir, de comprobar y eliminar las distintas cabeceras hasta conseguir los datos. Mientras que los bloques denominados *Receiver* realizan todo lo contrario. Toman los datos a enviar y los encapsulan con las diferentes cabeceras hasta formar la trama completa.

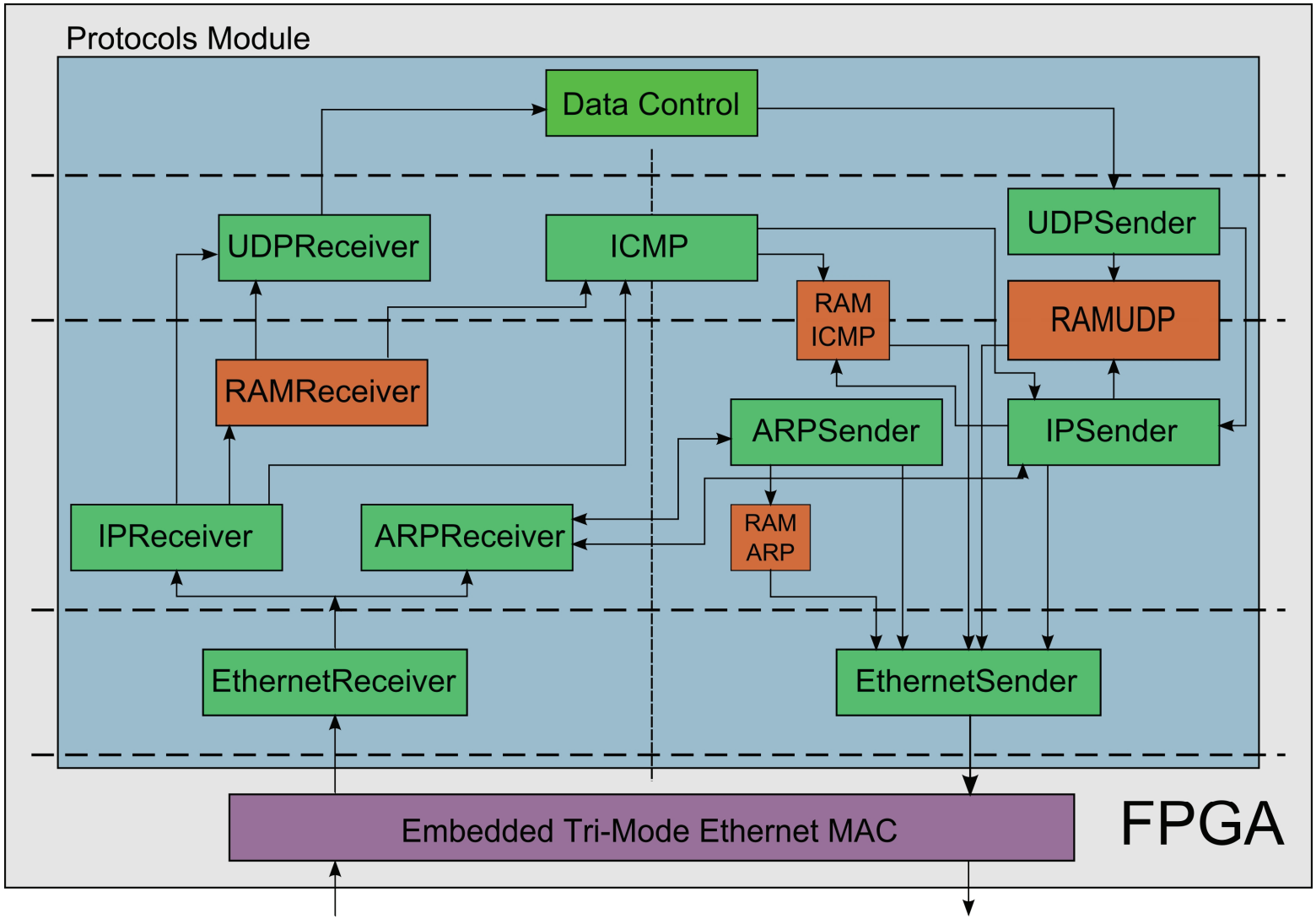


Figura 9-4. Diagrama de bloques detallado de la lógica adicional HDL

De esta forma, se consigue dividir el diseño en bloques más pequeños, consiguiendo una implementación más sencilla y estructurada del código HDL. En la Figura 9-5 se ilustra la jerarquía del código HDL visto desde el software ISE de Xilinx. Aquí el bloque *top* se llama *gbe_top*, en vez de *Protocols Module*.

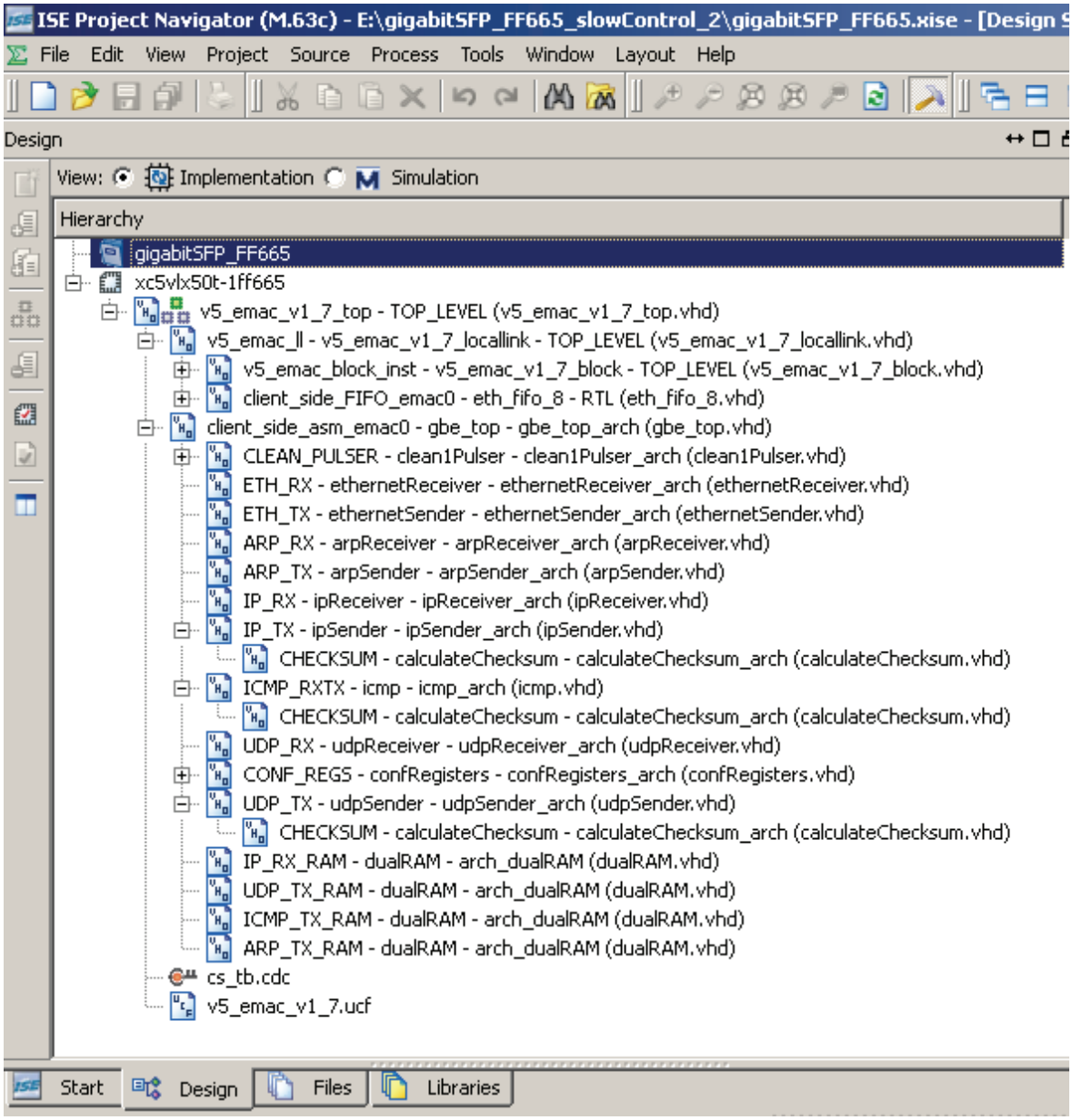


Figura 9-5. Jerarquía del código HDL ISE Xilinx

A continuación, se explica de forma breve, la funcionalidad de cada uno de los bloques situados dentro de la FPGA (ver Figura 9-4).

9.2.1 Ethernet

EthernetReceiver

A partir de las señales proporcionadas por el *core* se detecta la llegada de un datagrama y se comprueba y elimina, la *MAC Header* (ver Figura 6-2). Si ésta es correcta se indica a la capa superior que una trama nueva está disponible, indicando el tipo de protocolo (la capa superior debe saber si se trata de un protocolo IP o ARP). Los datos de la trama son

pasados a la capa superior de byte en byte. Si la *MAC Header* no fuese correcta la trama es desechada.

EthernetSender

La capa superior le indica que una trama está lista para ser enviada (esta trama se encuentra almacenada en una memoria RAM). Entonces, se encapsulan los datos proporcionados por la capa superior junto con la *MAC Header* correspondiente y, la trama final es pasada al *core* (ver Figura 9-2). También, el algoritmo de control de flujo expuesto en la sección 7.1 es implementado en este bloque.

9.2.2 ARP

ARPReceiver

Se encarga de procesar tramas *ARP Request*. Cuando recibe este tipo de tramas le dice a *ARPSender* que genere su correspondiente trama *ARP Reply*. También, almacena las dos últimas direcciones MAC asociadas a su IP correspondiente. Ya que cuado las capas superiores quieran enviar un datagrama, solicitarán la dirección MAC (a través de la IP) a este bloque. Si este bloque no tuviese almacenada la MAC solicitada, ésta es solicitada a través de la red. Es decir, *ARPReceiver* le dice indica a *ARPSender* que necesita una MAC determinada, éste último envía una trama *ARP Request* a la red, el ordenador en cuestión contestará con una trama *ARP Reply*, siendo recibida por *ARPReceiver*. Por tanto, la dirección MAC asociada a una IP determinada ha sido localizada.

ARPSender

Se encargar de generar tramas *ARP Request* y *ARP Reply*.

9.2.3 IP

IPReceiver

La capa inferior le indica que una trama está lista para ser recibida, entonces IPReceiver comprueba que el tipo de protocolo es correcto (ya que éste puede ser IP o ARP). Si es así, la trama es almacenada en una memoria RAM. Cuando el primer byte de la trama está listo para ser leído se lo comunica a la capa superior. De esta forma, se reduce al máximo el tiempo de procesamiento de los datos. Si el protocolo no fuese correcto (o porque es erróneo o es una trama ARP) se desecha la trama. También, este bloque es capaz de comprobar si el campo *checksum* de la *IP Header* es correcto. Si lo es, el procesamiento de los datos sigue adelante y si no, la trama es desechada.

IPSender

Este bloque se encarga de generar y almacenar en una memoria RAM la *IP Header* de la trama, cuando es solicitado por la capa superior. Cuando la *IP Header* es generada y almacenada avisa a la capa inferior.

9.2.4 ICMP

Este bloque genera paquetes los denominados paquetes de respuesta a eco [6]. Es decir, genera la respuesta a una función *ping* (paquete eco). El bloque no genera paquetes eco, ya que en el caso expuesto en este documento no tiene demasiado sentido o mejor dicho, no es de gran utilidad (al ser una conexión punto a punto la tarjeta donde está alojado el *firmaware* será controlada remotamente a través de un ordenador. Por lo tanto, no tiene sentido que la tarjeta haga un *ping* a ordenador para comprobar si está listo).

9.2.5 UDP

UDPReceiver

Procesa los datagramas UDP entrantes. Para ello comprueba si el protocolo, el puerto origen y puerto destino son correctos. Si alguno de estos campos no fuese correcto desecha la trama. También, es capaz de comprobar el campo *checksum* de la trama UDP.
Los datos obtenidos por este bloque se pasan a capas (bloques) superiores, en este caso, al bloque *Data Control*.

UDPSender

Compone datagramas UDP a través de los datos obtenidos del bloque *Data Control*. Cuando la trama está lista, es pasada a capas inferiores.
El bloque es capaz de generar el campo *checksum* de la trama UDP.

9.2.6 Data Control

Este bloque está todavía en fase de desarrollo. Dentro de este bloque se encuentra todo lo relacionado con *slow control* y control de flujo de los datos que quieren ser enviados empleando datagramas UDP.

9.2.7 Memorias RAM

El bloque *Protocols Module* contiene cuatro memorias RAM. Una de ellas se emplea para almacenar los datos de las tramas recibidas, teniendo en cuenta que el tamaño de ésta debe ser igual al tamaño máximo de las tramas que se reciban. El resto de las memorias se utilizan como memorias de transmisión. Es decir, en una de ellas se escriben los datos UDP que desean enviarse, en otra, los datos ICMP y en la última, la trama ARP. La memoria en la cuál se escribe los datos UDP debe ser lo suficientemente grande para procesar la trama más grande que se desee enviar y además, que permita escribir una trama nueva mientras otra se está enviando. Para ello, los datos son escritos en la memoria de forma circular.

Estas memorias deben ser arbitradas adecuadamente, ya que si no pueden surgir conflictos en el flujo de datos. Por ejemplo, el bloque *IPSender* debe escribir la *IP Header* en la memoria UDP o en la ICMP, según sea el caso.

Al diseñar estás memorias se ha pretendido consumir los mínimos recursos posibles de memorias RAM embebidas en la FPGA, ya que es un recurso bastante limitado. No para el diseño expuesto en este documento, sino para todo el *firmware* que debe ser implementado en la FPGA para dotar a la tarjeta FEC de las funciones deseadas.

9.3 Conexión *core* – lógica adicional

La lógica adicional pertenece a un fichero HDL *top* denominado *Protocols Module* (ver Figura 9-3). Pues dicho fichero debe ser sustituido por el fichero `address_swap_module_[8 | 16].v[hd]` (ver sección 9.1). Según las señales de entrada/salida que posea el fichero *top* habrá que modificar el fichero *top* proporcionado por el core.

En los ficheros proporcionados por el core se encuentran los ficheros `tx_client_fifo_[8 | 16].v[hd]` y `rx_client_fifo_[8 | 16].v[hd]`. Dichos ficheros son FIFOs que almacenan las tramas de transmisión y recepción, respectivamente. Según la documentación de Xilinx [2], estas FIFOs son de un tamaño de 4096 bytes y deben ser tan

grandes como la trama máxima que se quiera enviar o recibir. Por lo tanto, como se quieren enviar y recibir tramas de 9000 bytes estos ficheros deben ser modificados, para que al menos, las FIFOs puedan almacenar tramas de 9000 bytes.

9.4 Parámetros de configuración

Al implementar el *core* (*core* más la lógica adicional) se ha pretendido hacerlo de la forma más genérica y dinámica posible. A continuación se detallan los parámetros que pueden ser configurados empleando tramas de *slow control* (ver sección 8):

- Comprobar *checksum* de las tramas UDP recibidas
- Generar *checksum* de las tramas UDP que se desean enviar
- Número de tramas por evento
- Puerto origen de la trama UDP
- Puerto destino de la trama UDP
- Tamaño del las tramas UDP
- IP dónde se quieren enviar/recibir las tramas UDP

En cualquier caso, si algún parámetro (o parte del código) quiere ser modificado y no puede modificarse empleando las tramas de configuración, puede hacerse cambiando el código. Ya que por su estructura no es especialmente laborioso hacer modificaciones. Lo mismo ocurre, si se quiere emplear este código para otra aplicación totalmente diferente para la que ha sido diseñado. Al final, se trata de un *core*, capaz de enviar/recibir tramas ARP, ICMP y UDP.

9.5 Ejemplo de funcionamiento

En esta sección se pretende dar una explicación de cómo el *firmware* reacciona cuando se reciben o transmiten los distintos datagramas.

El ejemplo trata sobre el envío de un bloques de datos (la explicación hace uso de la Figura 9-4) a una dirección IP concreta, por ejemplo, la 10.0.0.3. La secuencia que sigue el *firmware* (a grosso modo) es la siguiente:

1. Se configura el sistema con los parámetros deseados, entre ellos la IP
2. El bloque *Data Control* le pasa los datos al bloque *UDPSenser*
3. *UDPSender* genera la trama UDP con los datos enviados por *Data Control* y la almacena en la memoria *RAMUDP*
4. *UDPSender* le indica a *Data Control* que no le pasa más datos de momento, ya que si no la trama siguiente se sobrescribe con la actual. *UDPSender* tendrá que esperar a que la trama actual comience a enviarse por el bolque *EthernetSender*. Mientras *UDPSender* espera, *IPSenser* genera la cabecera IP header y la almacena en el lugar adecuado dentro de la memoria *RAMUDP*
5. *IPSender* es el encargado de pasar la dirección MAC al bloque *EthernetSender*. Por lo tanto, *IPSender* le pregunta a *ARPReceiver* si tiene la MAC asociada a la IP 10.0.0.3. Si la tiene se la pasa y si no, *ARPReceiver* le indica a *ARPSender* que necesita dicha dirección MAC. Entonces, *ARPSender* envía una trama *ARP Request* a la red para obtener la dirección MAC en cuestión. *ARPReceiver* recibe la trama

ARP Reply correspondiente y actualiza los registros donde almacena las direcciones MAC (sólo almacena las dos últimas). Una vez actualizados los registros, *ARPReceiver* le pasa la dirección MAC solicitada a *IPSender*.

6. *IPSender* ya está lista para pasar la trama a la capa inferior, *EthernetSender*. Antes de hacerlo, comprueba que el canal de transmisión esté libre. Si no lo está, espera a que se libere.
7. *EthernetSender* le pasa la trama al *core* para que la envíe al exterior. Cuando *EthernetSender* comienza a enviar la trama, se activa una señal de sincronización para indicar a *UDPSender* que puede comenzar a procesar la siguiente trama. De esta forma, mientras una trama está siendo enviada la siguiente está siendo procesada. Así, se aprovechan todos los ciclos de reloj y por consiguiente, se obtiene el máximo *throughput* ofrecido por el core. El ciclo se repite hasta que se terminan de transmitir los datos deseados.

10 Ajustando la red

Incluso empleando un algoritmo para el control de flujo, las pruebas realizadas no resultaban totalmente satisfactorias. A veces, se perdían paquetes. Investigando sobre el problema se vio que cuando se trabaja a velocidades de transmisión elevadas, tales como 1 ó 10 Gbps, se deben ajustar diferentes parámetros del *kernel* del sistema operativo y de la tarjeta de red, para conseguir un funcionamiento óptimo, y así, evitar la pérdida de paquetes.

Hay tres formas de optimizar el sistema [14]:

- Incrementar el tamaño de MTU y la longitud `txqueuelen`
- Ajustar algunos parámetros del *kernel* del SO
- Modificar MMRBC

A continuación se detallan estos tipos de parámetros.
Parámetros del *kernel* del SO (Linux):

- **`net.core.rmem_default:`** Representa el tamaño por defecto del buffer del socket receptor.
- **`net.core.rem_max:`** Máximo tamaño del buffer del socket receptor
- **`net.core.wem_default:`** Representa el tamaño por defecto del buffer del socket transmisor
- **`net.core.wem_max:`** Máximo tamaño del buffer del socket transmisor
- **`net.core.netdev_max_backlog:`** Número de paquetes de entrada no procesados antes que el *kernel* del SO comience a perderlos.

`txqueuelen` es el tamaño del buffer de transmisión en un interfaz Ethernet. Incrementar este valor es extremadamente efectivo cuando los usuarios quieren transmitir gran cantidad de datos. También, es importante modificar **MTU**. Este parámetro limita el tamaño máximo de trama que puede ser enviado.

Finalmente, modificando el campo **MMRBC** en el espacio de configuración PCI-X se produce un gran efecto en el rendimiento. Un tamaño grande de MMRBC incrementa las longitudes de ráfaga de transmisión en el bus.

11 Pruebas realizadas

Esta sección trata de exponer los resultados obtenidos en las diferentes pruebas llevadas a cabo en el CERN durante los meses julio-agosto de 2010. Estas pruebas fueron realizadas con la ayuda de Filippo Costa, al cuál, desde aquí, le agradezco toda su ayuda.

11.1 Setup

Las pruebas fueron llevadas a cabo en el laboratorio de DATE en el CERN. El ordenador empleado estaba configurado como sigue:

- Linux pcalref15 2.6.9-55.0.2.EL.cernsmp
- Processor: 0, 1, 2, 3, 4, 5, 6, 7
- Vendor id: GenuineIntel
- CPU family: 6
- Model: 7
- Model name: Intel (R) Xenon (R) CPU E5405 2.00 GHz
- Stepping: 6
- CPU MHz: 2000.377
- 10 Gigabit Ethernet Card
- DATE: 6.40.xx

En la Figura 11-1 se muestra una fotografía del setup empleado para realizar las pruebas. En ella se pueden observar las dos tarjetas FEC, conectadas a una fuente de alimentación y a un ordenador. Para realizar las conexiones se red se emplearon cables UTP cat5E.

En la fotografía no se aprecia muy bien pero las tarjetas FEC estaban provistas de un pulsador provisional para comenzar la transmisión. Cuando la transmisión era iniciada no podía ser detenida hasta pulsar el reset. Por lo tanto, continuamente se estaban enviando eventos. Los datos de los eventos enviados estaban formados por un contador de 32 bits. De esta forma, en destino se podía medir la calidad de los datos de una forma sencilla.

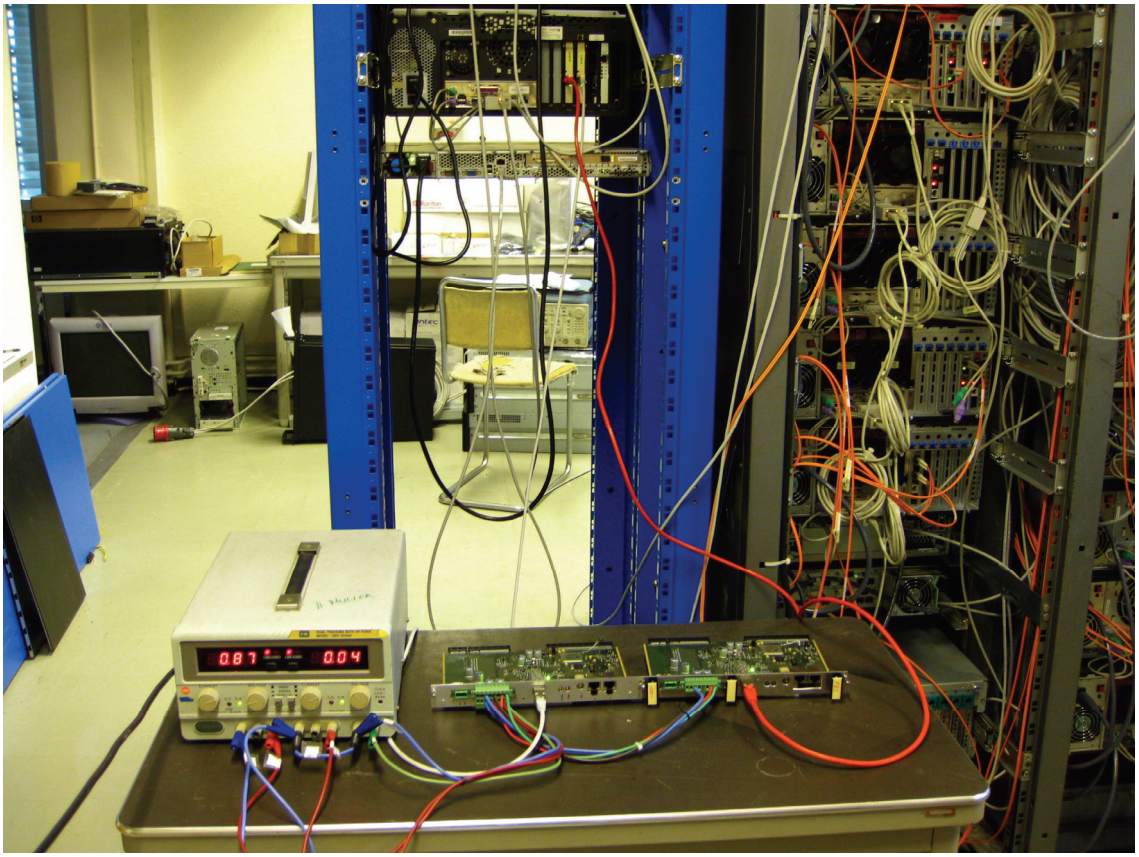


Figura 11-1. Setup agosto 2010 CERN (laboratorio de DATE, ALICE)

11.2 Conectando una tarjeta FEC a DATE

Antes de comenzar la transmisión se ajustaron los siguientes parámetros:

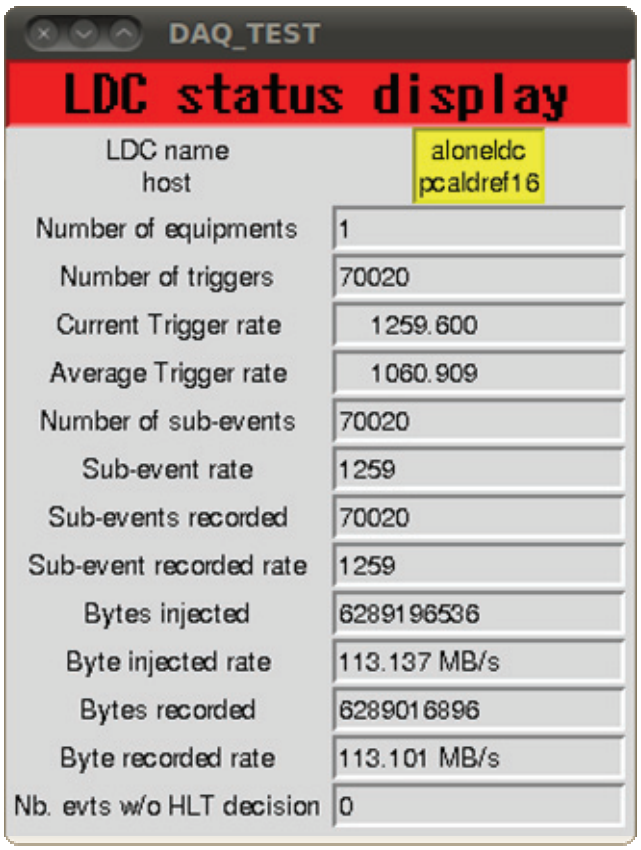
- IP de la tarjeta FEC: 10.0.0.2
- IP de la tarjeta de red: 10.0.0.3
- MTU de la tarjeta de red: 9000
- Tamaño de los paquetes UDP: 9000 bytes
- Tamaño del evento: 10 tramas por evento (89820 bytes)

Una vez ajustados los parámetros anteriores se hace un *ping* para comprobar que la tarjeta FEC funciona correctamente. Finalmente, se inicia la transmisión.

En la Figura 2-1 se exponen los resultados obtenidos. La Figura 11-2 a muestra una captura de pantalla de los distintos parámetros que se obtienen a través de DATE. Un de ellos, es el *throughput*, el cual toma un valor de 113,101 MB/s. Un valor bastante bueno, aunque puede aumentarse (hasta 120/124 MB/s), incrementando el número de tramas por evento. La Figura 11-2 b muestra el tamaño de los eventos recibidos, el cual es correcto, ya que cada evento está formado por diez tramas de 9000 bytes (*DATE header* + *Equipment header* + *payload* = 68 + 28 + 8972*10 + 4 *end of event* = 89820 bytes). También, en esta prueba se comprobó la calidad de los datos. Es decir, si los datos enviados eran los mismos que los recibidos. Se enviaron unos noventa y cinco millones de eventos y no se obtuvo ningún error. Este dato no puede mostrarse gráficamente, ya que DATE sólo muestra

mediante un mensaje en pantalla, si se ha producido algún error. No se produjo ningún error.

De acuerdo a los resultados obtenidos, se puede decir que la prueba fue satisfactoria. Ya tanto el valor del *throughput* como del tamaño de los eventos obtenidos, fueron los esperados. Además, no se perdió ningún paquete y, los datos recibidos coincidieron con los transmitidos.



(a)

(b)

Figura 11-2. Tamaño del paquete recibido en DATE

11.3 Conectando dos tarjetas FEC a DATE

Se procede igual que en el caso anterior.

Parámetros:

- IP de la tarjeta FEC A: 10.0.0.2
- IP de la tarjeta FEC B: 10.0.1.2
- IP de la tarjeta de red A: 10.0.0.3
- IP de la tarjeta de red B: 10.0.1.3
- MTU de la tarjeta de red A: 9000

- MTU de la tarjeta de red B: 9000
- Tamaño de los paquetes UDP en ambas tarjetas FEC: 9000 bytes
- Tamaño del evento en ambas tarjetas FEC: 10 tramas por evento (89820 bytes)

Al igual que el apartado anterior, se comprueban los mismos parámetros. Y según éstos, la prueba fue satisfactoria. En este caso, el valor del *throughput* es el doble (ver Figura 11-3) que el obtenido en el apartado anterior porque se están recibiendo datos desde dos tarjetas FEC (aproximadamente 117 MB/s por tarjeta). Por la misma razón, el tamaño del evento recibido es diferente, 179572 bytes (68 + 28+8972*10 + 4 + 28+8972*10 + 4).

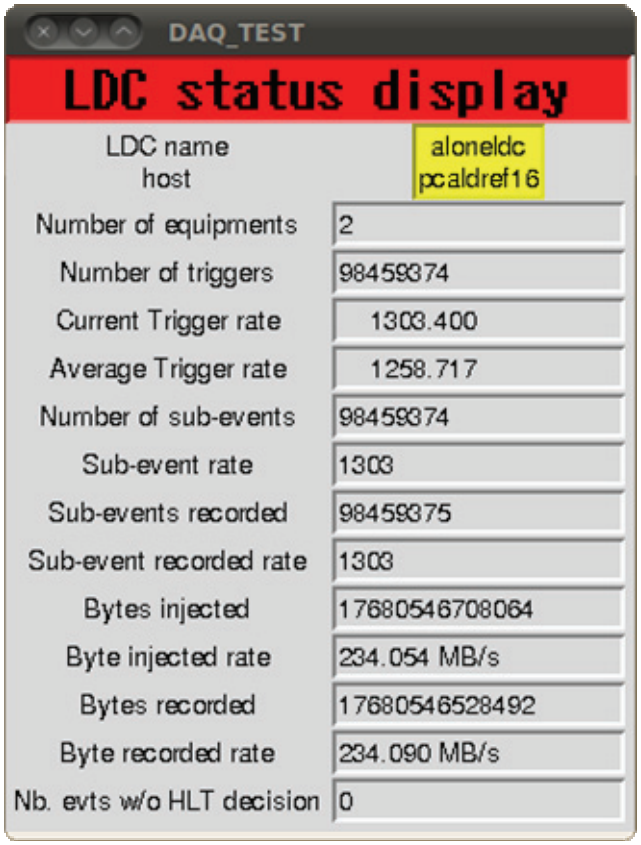


Figura 11-3. Resultados obtenidos enviando datos desde dos tarjetas FEC a DATE

11.4 Slow control

Actualmente, todo lo relacionado con *slow control* está en fase de desarrollo, ya que un *firmware* básico va a ser empleado en varios experimentos (a parte de NEXT). Por tanto, es necesario alcanzar un acuerdo respecto a los registros comunes necesarios. Sin embargo, si se han llevado a cabo pequeñas pruebas. Una de ellas es la explicada en la sección 8.

¿Cómo fue llevada a cabo la prueba?

Inicialmente, como puede observarse en la Figura 8-2, los registros tamaño de trama y número de tramas por eventos toman el valor 4000 y 14 respectivamente. Sabiendo, estos parámetros se iniciaba la transmisión y se comprobaba que el tamaño del evento recibido era el correcto (se procede igual que el la sección 11.2). Ahora, se resetea la tarjeta FEC (el valor inicial de los parámetros es el mismo, es decir, 4000 y 14) y se envía una trama como la mostrada en la Figura 8-3. Para enviar esta trama se empleó un pequeño programa escrito en C (por Luis Serra, IFIC, NEXT). Una vez que, el datagrama con los parámetros de

configuración ha sido enviado, se inicia la transmisión y se comprueba que los eventos recibidos están formados por 10 tramas de tamaño 9000 bytes.

12 Conclusiones

Se ha implementado satisfactoriamente un sistema de comunicación Gigabit Ethernet sobre FPGA, capaz de comunicarse con DATE. Este sistema posee las siguientes características:

1. Es capaz de enviar/transmitir tramas UDP de una forma totalmente dinámica. Es decir, el usuario sólo tiene que proporcionar los datos al sistema, que éste se encarga de generar las tramas. Si se recibe una trama, el sistema elimina todas las cabeceras que posee la trama (comprobando los distintos campos de estas cabeceras) y entrega los datos.
2. Es capaz de establecer comunicación con el otro punto de la red empleando tramas ARP.
3. Tiene implementada la famosa función *ping*. Empleada para comprobar el estado de la conexión y también, para medir la latencia que tardan los dos puntos remotos en comunicarse.

Para conseguir el correcto funcionamiento del sistema descrito en el texto, ha sido necesario definir diferentes tipos de tramas, tales como la trama principal, trama final de evento, trama de *slow control*, etc. Para ello se ha tenido que estar en contacto permanente con físicos del CERN.

Se han comprobado las prestaciones del sistema para elegir el formato de trama correcto. Para ello se implementó un sistema básico y con éste, se llevaron a cabo diferente pruebas en el CERN junto a Filippo Costa durante el mes de agosto de 2009 (estas pruebas fueron realizadas con la tarjeta de desarrollo ML505, ver Figura 5-1). Estas pruebas hicieron de guía para desarrollar el sistema final, detectando qué funciones debería desempeñar el *firmware*.

Durante el pasado mes de agosto, se realizaron diferentes pruebas en el CERN, nuevamente junto a Filippo Costa, empleando la tarjeta para la cual se ha diseñado este *firmware*, tarjeta FEC. Con estas pruebas se comprobó el estado de la conexión entre la tarjeta FEC y DATE. Se comprobó que el *throughput* conseguido era el esperado y la calidad de los datos (siendo los mismos los enviados que los recibidos y, no detectando pérdida de datos), conectado primero una, y luego dos tarjetas FEC a DATE.

También, durante la estancia pasada en el CERN, junto con Sorin Martoiu (ingeniéro electrónico para RD51, CERN), se comprobó prácticamente la cadena completa, en lo que se refiere a la electrónica. Se capturaron datos mediante un detector y éstos, mediante una conexión HDMI se pasaban a una tarjeta digitalizadora (con dos ADCs), la cual era controlada por la tarjeta FEC (ver Figura 12-1). La adquisición fue correcta. Para completar totalmente la cadena, habría que haber enviado los datos tomados por los ADCs a DATE, a través de la conexión de Gigabit Ethernet.
Actualmente, se está poniendo a punto un demostrador con toda la cadena en funcionamiento para el próximo meeting de NEXT, a finales de septiembre en Gandia.

Como conclusión final, se puede decir que se ha implementado un sistema Gigabit Ethernet sobre FPGA totalmente genérico y dinámico, lo cual hace que pueda ser usado en cualquier otra aplicación que necesite una conexión Gigabit Ethenet sobre FPGA.
A día de hoy, hay varias aplicaciones en la que se quiere emplear el sistema expuesto en estás páginas.

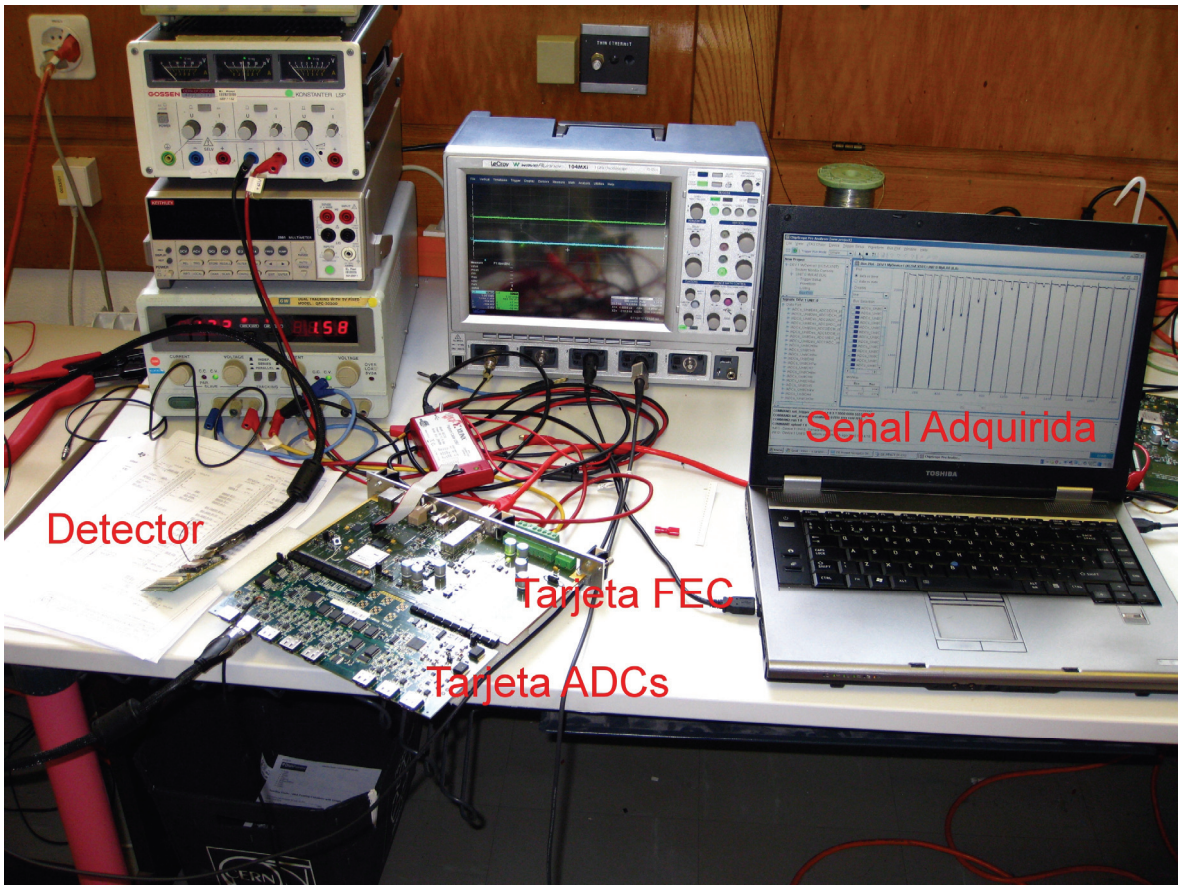


Figura 12-1. Cadena de adquisición de datos (Detector-Tarjeta ADCs-Tarjeta FEC)

13 Trabajo futuro

En primer lugar, desarrollar un conjunto de registros de configuración. Algo a tener muy en cuenta, ya que éstos permitirán ajustar de manera óptima el sistema para el buen funcionamiento del experimento NEXT (y resto de experimentos).

En segundo lugar, adaptar este sistema a una tarjeta con tecnología más avanzada que la tarjeta FEC (en concreto, la tarjeta SRU, de la que se habla en la sección 1.1). También, es una tarjeta de adquisición de datos, controlada por FPGA (Virtex 6), la cual va a ser usada en EMCal, ALICE, CERN.

Por último, migrar el sistema expuesto en este documento a 10 Gigabit Ethernet. Para ello, se trabajará con la tecnología más avanzada que existe en el mercado, como Virtex 6 y 10-GbE, usando el estándar SFP+.

Actualmente, estas son las tres tareas principales que se prevén abordar. Y por supuesto, actualizar el sistema actual.

14 Bibliografía

[1] Clive “Max” Maxfield, “FPGAs, Instant Access”, Newnes 2008

[2] Xilinx,“Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.5-7”, http://www.xilinx.com/support/documentation/ipcommunicationnetwork_ethernet_v5embedtemacwrap.htm agosto de 2010

[3] Xilinx, “Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide”, http://www.xilinx.com/support/documentation/ipcommunicationnetwork_ethernet_v5embedtemacwrap.htm agosto de 2010

[4] Xilinx, “Virtex-5 Embedded Tri-Mode Ethernet MAC Wrapper v1.7 Data Sheet”, http://www.xilinx.com/support/documentation/ipcommunicationnetwork_ethernet_v5embedtemacwrap.htm agosto de 2010

[5] Alberto «León-García e Indra Widjaja, Redes de Comunicación, Conceptos Fundamentals y Arquitecturas Básicas», McGraw Hill 2002

[6] William Stallings, «Comunicaciones y Redes de Computadores», Prentice Hall 6ª Edición

[7] Xilinx, “ML505/ML506/ML507 Evaluation Platform User Guide”

[8] Adaptador SFP, http://www.3com.com diciembre de 2009

[9] Filippo Costa, “DATE UDP readout test results and future proposal”, September 2, 2009

[10] Filippo Costa , “RD51 working group meeting at CERN”, November 23, 2009

[11] Protocolo UDP, http://www.networksorcery.com/enp/default1101.htm diciembre 2009

[12] IP Stack, http://www.itee.uq.edu.au/~peters/xsvboard/stack/stack.htm diciembre de 2009

[13] Gary Stringham, “Hardware/Firmaware Interface Design, Best Practices for Improving Embedded System Development”, Newnes 2010-09-05

[14] Jinyong Jo, Junguk Kong and Okhwan Byeon, “Optimizing System Performance for Uncompressed HD over Gigabit Ethernet: A case Study”, High Performance Research Networking Dep., Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea

15 Anexo

15.1 Glosario

Acrónimos y abreviaciones usadas a lo largo del texto.

ADC	Analog-to-Digital Converter
APV25	The APV25 is a 128-channel analogue pipeline chip for the readout of silicon microstrip detectors in the CMS tracker at the LHC
ARP	Address Resolution Protocol
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CPU	Central Processing Unit
DATE	Data Acquisition and Test Environment
DCR	Device Control Register
EMI	Electromagnetic Interference
FCS	Frame Check Sequence (MAC frame)
FEC	Front-End Concentrator
FIFO	First In, First Out memory
Firmware	Drivers de dispositivo o software de bajo nivel que controla el hardware escribiendo valores en los registros, interpretando los valores leídos de los registros, y responde a solicitudes de interrupción del hardware [13]
FPGA	Field Programmable Gate Array
GBIC	Gigabit Interface Converter (optical transceiver)
Gbps o Gb/s	Gigabits per second
GMII	Gigabit Media Independent Interface
GTP	High-speed serial transceivers offered in the Virtex-5 LXT and SXT platforms
GTX	High-speed serial transceivers offered in the Virtex-5 FXT and TXT platforms
HDL	Hardware Description Language
ICMP	Intenet Control Message Protocol
IFIC	Instituto de Física Corpuscular
IP	Intellectual Property or Internet Protocol
IOB	Input/Output Block
LED	Light-Emitting Diode
LHC	Large Hadron Collider
LUT	Look Up Table. Es básicamente una pequeña memoria, usada para implementar funciones lógicas dentro de la FPGA junto con otros elementos (mutiplexores, flip-flops, etc.)
MAC	Media Access Controller

Mb/s	Megabits per second
MDIO	Management Data Input/Output
MII	Media Independent Interface
MMRBC	Maximum Memory Read Byte Count
MTU	Maximum Transmission Unit
NEXT	Neutrinoless double beta decay searches with a high-pressure Xe TPC
OSI	Open Systems Interconection
PC	Personal Cumputer
PCS	Physical Coding Sublayer
PHY	Physical Layer The term refers to all physical sublayers (PCS, PMA, PMD). Often applied to a device, e.g., BASE-T PHY: an external chip which can connect to the Ethernet MAC to perform this physical standard.
PMA	Physical Medium Attachment
PMD	Physical Medium Dependent
PMT	Photomultiplier Tube, an extremely sensitive light detector
RGMII	Reduced Gigabit Media Independent Interface
RX	Receiver
SFP	Small Form-factor Pluggable (optical transceiver)
SGMII	Serial Gigabit Media Independent Interface
SiPM	Silicon Photomultipliers
SO	Sistema Operativo
SRU	Scalable Readout Unit
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol