UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de grado

Grado en ingeniería de sistemas de telecomunicación, sonido e imagen

Departamento de ingeniería electrónica

Josep Rioja Ibañez

Generador de formas de onda arbitrario con una FPGA

Tutor:
Javier Valls Coquillat

Agraïments

Agraïsc tant a la meua família com el Professor Javier Valls la paciència que han tingut en el transcurs de l'aprenentatge i confecció d'aquest treball.

Índice

Resumen

En este trabajo se va a explicar cómo se ha diseñado e implementado utilizando un PC con el programa Matlab y una FPGA, un generador de formas de onda arbitrario de alta velocidad (el conversor muestrea a 5 Gsps). Un generador de formas de ondas arbitrario es un sistema que repite cíclicamente un tipo de señal seleccionada por el usuario (una señal seno, una rampa, etc.). El sistema propuesto en este trabajo de fin de grado cumple con este principio de funcionamiento ya que, desde el Matlab el usuario va a programar una función o cargar desde un archivo los datos de la señal a utilizar. Esta señal seleccionada por el usuario será enviada vía ethernet desde el PC por medio del programa Matlab hasta la FPGA. Una vez llegada la señal a la FPGA será procesada y almacenada dentro de una memoria DRAM para su después reproducción. Por último, el sistema está preparado para seleccionar el canal de transmisión de los datos. Existen dos opciones de uso del sistema para el usuario, hacer una comprobación de vuelta de los datos almacenados en la DRAM o seleccionar su transmisión cíclica por el canal auxiliar implementado.

Palabras clave: *FPGA*, *Matlab*, *DRAM*, *Ethernet* y *Generador*.

Abstract

In this final paper we will explain how an arbitrary hihg-speed waveform generator was designed and implented, by using a PC with the Matlab program and a FPGA, an arbitrary high-speed waveform generator (the converter working at 5 Giga samples per second). An arbitrary waveform generator is a system that cyclically repeats a type of signal selected by the user (a sine signal, a ramp, etc.). The system proposed in this end-of-degree project meets this principle of functionality since, the user will be able to program a function from Matlab, or load the data of the signal from a file to be used. This signal selected by the user will be sent via ethernet from the PC via the Matlab program to the FPGA. Once the signal arrives to the FPGA it will be processed and stored inside a DRAM memory for its later reproduction. Finally, the system is ready to select the transmission channel of the data. There are two options for using the system for the user, checking the data stored in the DRAM or selecting its cyclic transmission via the implemented auxiliary channel.

Keywords: *FPGA*, *Matlab*, *DRAM*, *Ethernet* y *Generator*.

Resum

En aquest treball s'explica com s'ha dissenyat i implementat utilitzant un PC amb el programa Matlab i una FPGA, un generador de formes d'ona arbitrari d'alta velocitat (el conversor mostreja a 5 Gsps). Un generador de formes d'ona és un sistema que repeteix cíclicament un tipus de senyal la qual l'usuari vol reproduir (una senyal seno, una rampa, etc.). El sistema proposat pel treball final de grau acompleix amb aquest principi de funcionalitat, ja que, des del programa Matlab l'usuari programarà una funció o carregarà un arxiu que contindrà la senyal que es vol utilitzar. Aquesta senyal seleccionada per l'usuari serà enviada a través d'ethernet des del PC fins a la FPGA on serà processada i emmagatzemades dins d'una memòria DRAM per a la seva posterior reproducció. Per últim, el sistema està preparat per a seleccionar el canal de transmissió de dades. Existeixen dues opcions d'ús del sistema per a l'usuari. La primera opció és fer una comprovació de volta de les dades almasenades en la DRAM a través de la connexió ethernet amb el PC. La segona és la reproducció cíclica encaminant les dades des de la memòria DRAM fins al conversor de 5 Gsps.

Paraules clau: *FPGA*, *Matlab*, *DRAM*, *Ethernet* y *Generador*.

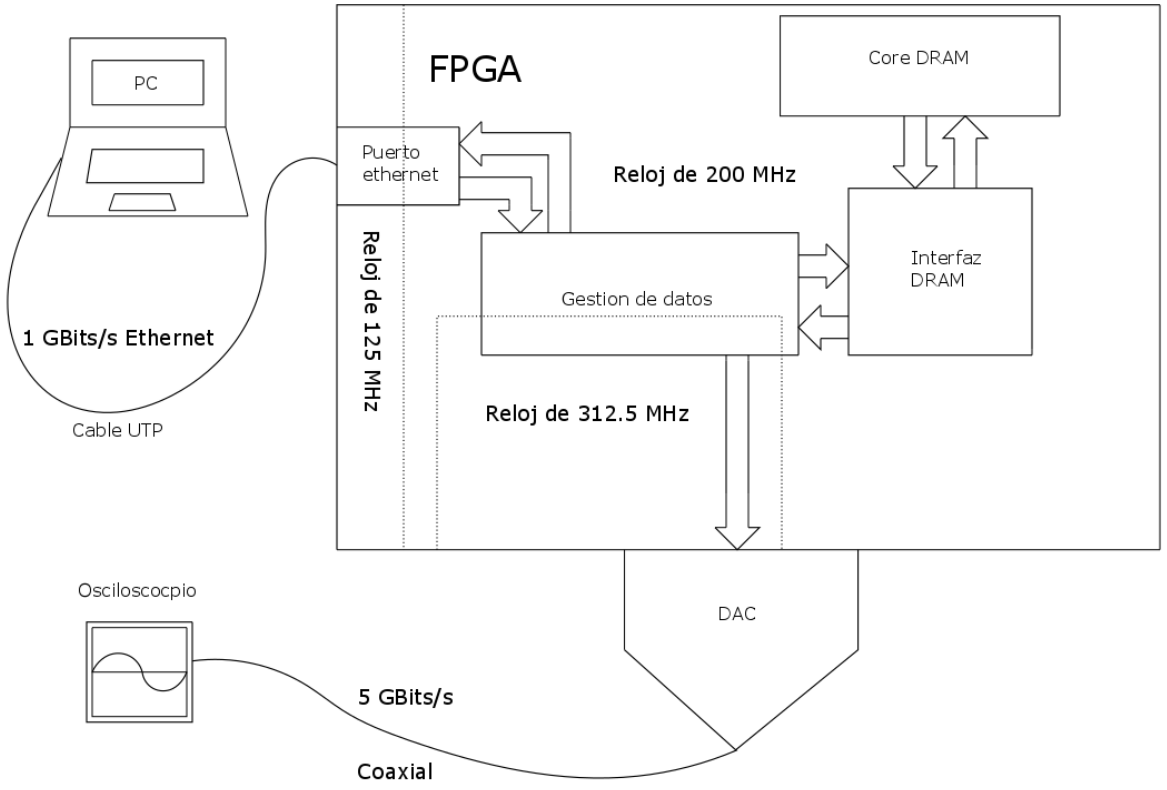
Capítulo 1

Introducción

En este capítulo se hace una presentación del diseño inicial de la estructura general del generador de formas de onda. A continuación, se hará una breve descripción de los obstáculos que tendremos que sortear para unir los diferentes componentes que conformaran el generador de formas de onda.

1.1. Estructura general del generador de formas de onda

En esta parte se va a hacer un esbozo inicial del proyecto. En la figura siguiente vemos el boceto de la estructura del generador de formas de onda que se va a implementar.



En la estructura general del generador de formas de onda podemos diferenciar las 3 principales partes de la estructura. Estas son la transmisión de datos desde el ordenador hacia la FPGA por vía ethernet, el procesado y almacenamiento de los datos en la memoria DRAM y por último la lectura, procesado y envío de datos hacia el conversor DAC. A continuación, en los siguientes puntos se explica las principales características y principales problemas que vamos a tener que sortear por los diferentes bloques programados.

1.1.1. Principales características de la transmisión ethernet

Para el envío de los datos desde el programa Matlab, se ha elegido la transmisión por vía ethernet. Dentro del protocolo ethernet, se va a utilizar el protocolo Gigabit ethernet que estaba implementado para la FPGA Xilinx Virtex y Kintex por medio de un Firmware que teníamos disponible.

Otro obstáculo que se va a sortear es la configuración del protocolo TCP/IPv4 tanto a nivel software del sistema operativo como a nivel hardware (modificar los datos del protocolo TCP/IPv4 que están implementado dentro del firmware).

1.1.2. Principales características del procesado y gestión de datos

Como podemos observar en el boceto del sistema, tenemos una serie de bloque que trabajan a diferentes velocidades de reloj y con diferentes tamaños de tramas. Para unir estos diferentes periféricos de transmisión/recepción o almacenamiento de datos, hay que diseñar e implementar una serie de bloques Hardware dentro de la FPGA. Estos bloques se dividirán en dos grupos, uno se encargará de gestionar la escritura y lectura de datos hacia la DRAM para que tanto escritura como la lectura de la DRAM sean configurables.

Otra serie de bloques se encargarán del transporte y transmisión de datos hacia el puerto ethernet o hacia el conversor DAC.

1.1.3. Principales características del conversor DAC

En el diseño inicial tenemos que vamos a utilizar un conversor digital a analógico (EUVIS FMC 2657 5-GSPS 12-bit Dual DAC FMC Module). Este conversor tiene 16 canales, donde en cada canal se transmitirán 12 bits en cada uno de los ciclos de reloj. Este conversor tiene frecuencia total de muestreo de 5 Gigamuestras por segundo y que hay que abastecerlo constantemente de datos que, en sí, van a ser las muestras que conforman la onda que se quiera transmitir. Estas muestras serán suministradas por la memoria DRAM que tiene integrada los Kits tanto Virtex como Kintex de la Serie 7.

1.1.4. Principales características de la memoria DRAM

La memoria DRAM que tiene implementada la FPGA tanto la Virtex como la Kintex de la serie 7 es una memoria DRAM ddr3 de 1 Gigabyte. Haciendo unos cálculos rápidos iniciales tenemos que, el conversor va a tener que alimentarse constantemente con una tasa total de datos de 12 bits por 16 canales con una frecuencia de reloj de cada uno de los canales de 312.5 MHz. Esto hace que el conversor tenga una tasa neta total de datos de

60 GBits/s. Para mantener una tasa tan alta de suministro de datos, se ha pensado en la memoria DRAM de la FPGA la cual tiene una tasa en bruto de entrega de datos de 102.4 GBits/s (512 bits por 200 MHz de frecuencia de reloj).

1.2. Objetivos del TFG

En este apartado se van a definir los objetivos que deberá cumplir el sistema que se ha diseñado e implementado para emular el comportamiento de un Generador de formas de onda arbitrario. El principal objetivo que se quiere alcanzar es tener un sistema sin fallos que emule el funcionamiento de un Generador de formas de onda arbitrario. Aunque el objetivo principal es el anteriormente descrito, existen otros objetivos secundarios que tendremos que cumplir. Para ello se van a describir en este apartado los objetivos que se quieren alcanzar en cada uno de los puntos fundamentales del trabajo. Más adelante, en el capítulo Conclusiones se hará una valoración de cada uno de los puntos descritos.

Profundizando más en los objetivos, vemos que tenemos dos partes diferenciadas en el TFG. Un parte está diseñada e implementada con programas Software y otra parte diseñada e implementada con programas Hardware.

Los objetivos dentro de la implementación Software que se van a valorar en las conclusiones son:

- Configuración automatizada del sistema operativo Windows.
- Configuración automatizada de los parámetros de transmisión vía ethernet en la tarjeta de red.
- Elección y control de los ciclos de escritura y lectura de los programas Hardware implementados en la FPGA.

En cuanto a los objetivos dentro de la parte implementada en la FPGA son:

- Los programas deben tener una probabilidad de error igual a cero.
- La escritura y lectura de la DRAM deben ser configurable en cuanto al tamaño de las tramas de datos.
- Los programas deben trabajar sincronizados entre ellos mismos y sin ningún tipo de problema de propagación tanto de las señales de control como las señales de transporte de datos.
- El conjunto de programas Hardware deben cumplir las tasas de datos que se desea para abastecer el conversor de 5 Gsps.

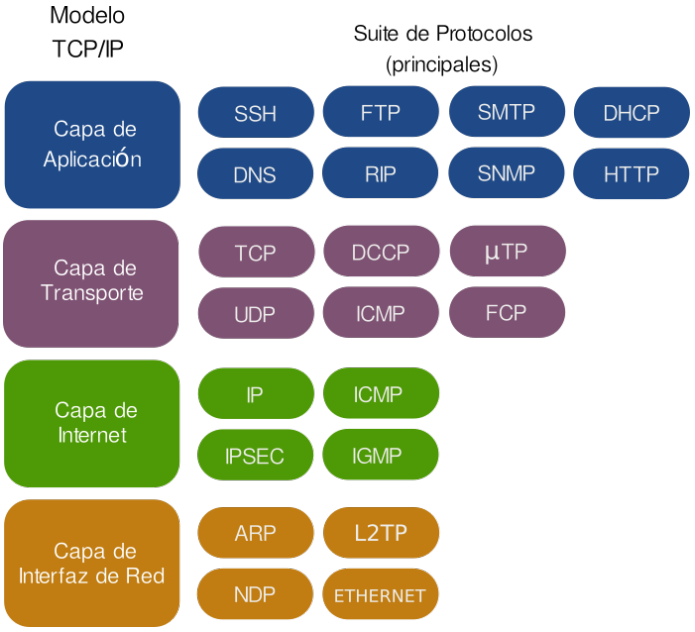
Capítulo 2

Diseño Software

En este capítulo se va a explicar más detalladamente el diseño Software implementado dentro de la FPGA. En este diseño Software se observan una serie de módulos principales que controlan el flujo de datos. El primer módulo importante es la interfaz del puerto ethernet por donde se van a recibir o transmitir los datos hacia el dispositivo conectado. Otra parte importante del diseño hardware es el módulo protocolo. Este módulo interpreta los datos que llegan desde el módulo ethernet como comandos o datos. El tercer y principal programa del trabajo es el módulo que controla la interfaz DRAM. A su vez existen otras tres interfaces que regulan el flujo de datos y adaptan el tamaño de las tramas para la conexión entre interfaces.

2.1. Protocolos utilizados para la comunicación punto a punto entre dispositivos

La forma escogida para conectar los dos dispositivos es realizar una conexión punto a punto utilizando el protocolo TCP/IPv4. La arquitectura del Protocolo TCP/IPv4, como podemos observar en la imagen siguiente, nos permite realizar una conexión punto a punto de fácil implementación. Para nuestro sistema, las funciones programadas actúan sobre la capa de transporte y la capa de internet (Protocolo ARP).



En nuestro programa software para transmitir datos hacia la FPGA solo configuramos las variables del protocolo TCP/IPv4 que necesitamos para que se pueda encapsular en tramas la información y pueda ser enviada por el puerto correspondiente para que el protocolo UDP interconecte la capa de transporte con la capa de internet. Esto se debe a que ya está implementada esta funcionalidad en Matlab. Una vez los paquetes llegan a la capa de internet, es el protocolo ARP el que entra a gestionar los datos que se quieren transmitir. En cuanto al protocolo ARP (Address Resolution Protocol), se ha optado por hacer una configuración estática. En una tabla estática, todos los paquetes con la dirección IP y MAC que coincidan en la tabla con la asignada a la FPGA, automáticamente el PC los enviará mediante el puerto asignado para esta transmisión. En cuanto a lectura, todos los paquetes que reciba con la IP y la Mac asignadas en la tabla y que hacen referencia al PC (en concreto a la tarjeta de red ethernet), serán enviados por el puerto UDP asignado y podrán ser leídos, por el programa Matlab, si así se requiere por la ejecución de algún comando de lectura. Esta configuración estática solo es viable en redes punto a punto como la diseñada.

2.2. Funciones del sistema programadas en Matlab

En este apartado se explica cómo funcionan las principales funciones programadas en el compilador Matlab. Estas funciones, a su vez, pueden contener dentro de ellas otras funciones propias del compilador Matlab o programadas por nosotros. Estas funciones de menor interés se encargan de procesar y adaptar las tramas de datos para la transmisión o en la recepción.

2.2.1. Funciones relacionadas con la configuración del protocolo TCP/IPv4 y ARP

En primer lugar, se va a explicar las funciones relacionadas con la configuración del protocolo TCP/IPv4 tanto en el compilador Matlab, como en el sistema operativo utilizado. Para ello tenemos una serie de funciones programadas que van configurando y almacenando los parámetros necesarios tanto del propio PC, como de la FPGA.

La primera función que se va a explicar es la función **mac_and_name**, que devuelve dos variables de tipo string, que son la MAC de la tarjeta de red que seleccionemos y el nombre del dispositivo que tiene esta tarjeta red en el sistema operativo. Al ejecutar está función aparecerá en la consola un listado de dispositivos donde el usuario tendrá que elegir la tarjeta ethernet donde está conectado mediante un cable UDP a la FPGA. La variable de entrada start, tiene que ser igual a 1 (solo es una variable de ejecución).

[mac_user, device_ethernet_name]=mac_and_name(start)

La segunda función que se utiliza para la configuración es **default_configuration**. Esta función borra todas las entradas de la tabla ARP de la dirección de la red que se va a configurar y vuelve a habilitar el protocolo DHCP. La variable que devuelve es de tipo double (0 indica que los comandos se han ejecutado correctamente).

[ok]= default_configuration(device_ethernet_name)

La tercera función es la **tcp_ip_parameters** que consta de dos partes. Una primera parte que contiene las variables internas que configurarán la red punto a punto entre los dos dispositivos. En el caso de haber introducido una MAC nueva en el código Verilog, tendría cambiarse también aquí la MAC de la FPGA por la nueva. La segunda parte de la función devuelve tres variables. La variable UDP contiene la configuración del puerto UDP y las variables **TCP_filter_dst** y **TCP_filter_scr**. Estas tres variables son del tipo struct y contienen los datos relacionados con la configuración de la red privada con el protocolo

TCP/IPv4.

[udp, TCP_filter_scr, TCP_filter_dst,data_path]=tcp_ip_parameters(mac_user)

La cuarta función es la función **arp**, que configura al sistema operativo la tabla estática de direcciones para la transmisión del sistema. A esta función, se le introduce las variables que necesita para ejecutar los comandos internos que hay programados para configurar el protocolo TCP/IPv4 en el sistema operativo Windows.

arp(device_ethernet_name,TCP_filter_scr,TCP_filter_dst)

Llegados a este punto, ya tenemos configurado el protocolo TCP/IPv4 en el sistema operativo. Las dos últimas funciones que vamos a explicar en este apartado pertenecen a la configuración del protocolo que hemos implementado en el programa hardware. Para ello, en primera instancia, enviaremos un paquete con la función **reset_fpga** con la orden de activar la señal interna de reset para resetear el programa hardware. Solo hay que dar a esta función la variable que contenga los datos del puerto UDP. Al hacer un reset dentro de la FPGA, también se borran todos los datos almacenados en la configuración TCP/IPv4 interna por defecto, por lo tanto, se tendría que volver a configurar el protocolo TCP/IPv4.

reset_fpga(udp)

La otra función de esta sección y última que se necesita para la configuración del sistema es la función **cmd_configuration_tcp_ip** que crea una serie de paquetes destinado a la configuración del protocolo TCP/IPv4 que hay implementado dentro de la FPGA. Estos paquetes son enviados vía ethernet, por lo tanto, hay que introducir en la función las variables que contengan el protocolo TCP/IPv4 (variables u, TCP_filter_scr, TCP_filter_dst explicadas en un punto anterior).

cmd_configuration_tcp_ip (udp,TCP_filter_scr,TCP_filter_dst)

Una vez ejecutadas todas estas funciones, ya tendremos configurado los dos dispositivos para poder transmitir y recibir datos via ethernet o ejecutar la orden de transmisión para la transmisión vía DAC.

2.2.2. Función de escritura de datos en la DRAM

En este apartado se va a explicar el proceso de escritura que realiza la función **write_fpga**. Con esta función, damos la orden de escritura al programa que controla el proceso de escritura de la memoria DRAM de la FPGA. Una vez abierta la comunicación, se enviará la información que contenga el archivo **data.mat**.Este archivo tiene que estar ubicado dentro de la carpeta **mensaje**. esta función hay que pasarle tanto la variable que contenga los datos del puerto abierto entre Matlab y la salida ethernet como la ruta de datos donde se encuentra la información a transmitir, en el caso que se quiera transmitir desde otra carpeta del sistema operativo.

write_fpga (udp, data_path)

2.2.3. Función de lectura vía ethernet

En este apartado se va a explicar el proceso de lectura de la información que contenga la memoria DRAM en su interior. Esta lectura se realiza a través de la conexión ethernet. La función **read_eth_fpga** se le pasa tanto la variable que contenga los datos de la transmisión TCP/IPv4 como la ruta de datos donde se encuentra la información a transmitir. Una vez ejecutada, aparecerá una pregunta en la consola, preguntado el número de bytes que se quiere leer de la DRAM.

read_eth_fpga (udp,data_path)

2.2.4. Función lectura vía DAC

Las últimas funciones que vamos a explicar es la función **read_eth_dac** y la función **stop**. En la función **read_eth_dac** damos la orden de lectura en la DRAM de la FPGA. Esta función hay que pasarle la variable que contenga los datos que han sido configurados para la transmisión TCP/IPv4. Una vez ejecutada, al usuario le aparecerán 2 preguntas en la consola. La primera está relacionada con la posición inicial de lectura de la DRAM. Esta opción está programada para poder guardar diferentes tipos de ondas dentro de la memoria DRAM, para su posterior reproducción. La segunda pregunta es el número de bytes que quieres reproducir a partir de la posición indicada anteriormente. El segundo número tiene que ser múltiplo de 64 bytes para que la función lo acepte. Como recordatorio, el sistema se ha programado para que una vez dada esta orden, cuando se termine de leer de la memoria

DRAM la cantidad de información indicada, se volverá a ejecutar desde el hardware cíclicamente la orden de lectura.

read_eth_dac(udp)

En el caso que queramos finalizar esta orden, solo tendremos que ejecutar la orden de stop, es decir, ejecutaremos a la función **stop**, a esta función solo tenemos que introducirle el parámetro de transmisión TCP/IPv4.

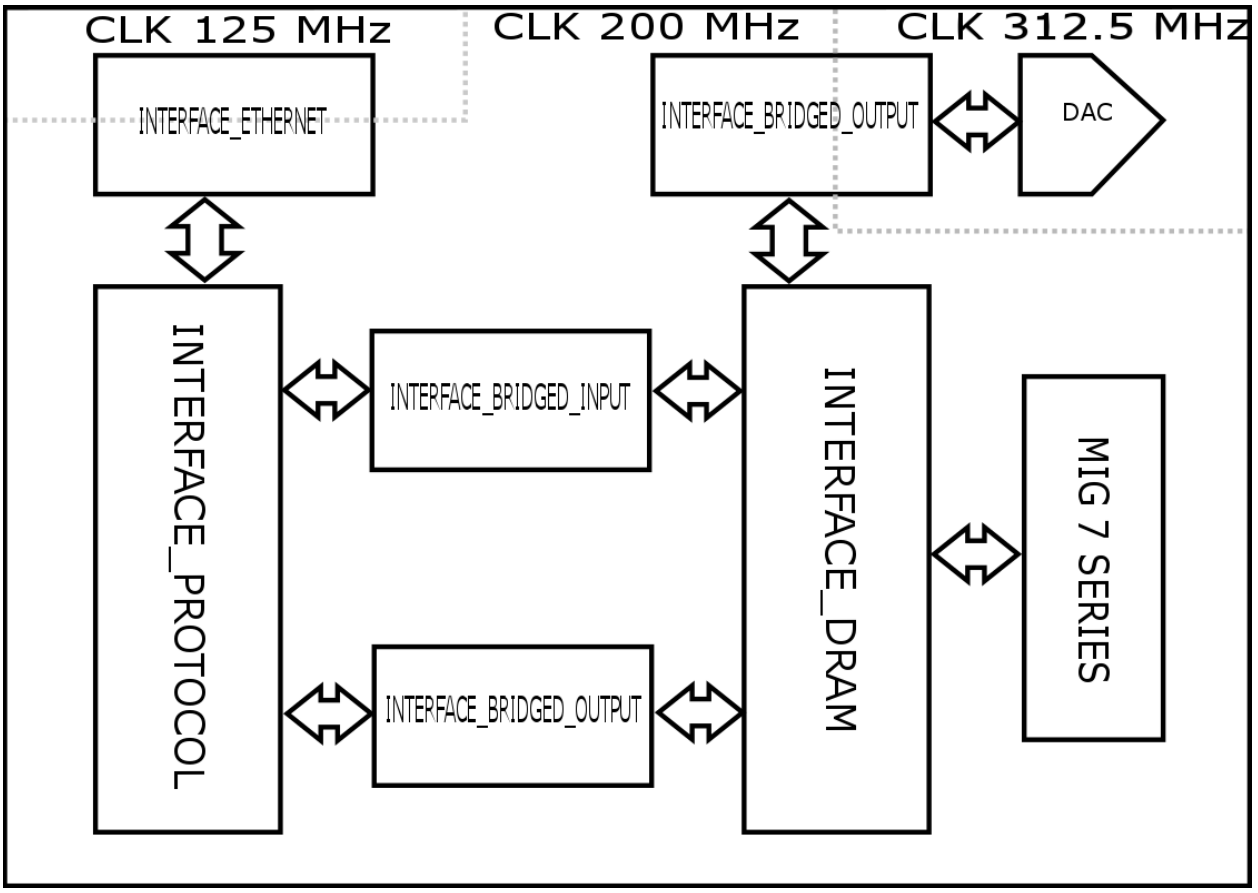
stop(udp)

Capítulo 3

Diseño Hardware

En este capítulo se va a explicar más detalladamente el diseño hardware implementado dentro de la FPGA. En este diseño hardware se observan una serie de módulos importantes. El primer módulo importante es la interfaz del puerto ethernet por donde se van a recibir o transmitir los datos hacia el dispositivo conectado. Otra parte importante del diseño hardware es el módulo protocolo. Este módulo interpreta los datos que llegan desde el módulo ethernet como comandos o datos. El tercer y principal programa del trabajo es el módulo que controla la interfaz. A su vez existen otras tres interfaces que regulan el flujo de datos y adaptan el tamaño de las tramas para la conexión entre interfaces.

3.1. Esquema hardware



3.2. Dominios de reloj

Como hemos podido observar en el dibujo del esquema hardware, tenemos diferentes tipos de relojes que interconectan diferentes regiones de nuestro diseño hardware. En concreto tenemos 3 tipos de reloj diferentes, uno de 125 MHz que trabaja la transmisión y recepción de datos vía ethernet. Este reloj es el definido en las transmisiones hardware del protocolo de Gigabit ethernet. Está generado internamente por el módulo ethernet a partir del reloj del sistema de la FPGA de 200 MHz. Aunque para nuestro diseño no hay ningún programa que funcione con este reloj, sí que hay que tenerlo en cuenta, ya que puede generar problemas con la conexión a otros módulos por problemas de timing pudiendo causar un mal funcionamiento del sistema. El reloj de 200 MHz está generado por la interfaz MIG del Core de la DRAM. Es el reloj que predomina en la arquitectura hardware diseñada. Por último, tenemos el reloj con el que trabaja la salida de datos de la interfaz que conecta con el DAC. Este reloj está generado por un generador de reloj externo de 5 GHz que, mediante dos divisores (dos divisores de frecuencia a la cuarta parte), llega a la interfaz DAC con una frecuencia de 312.5 MHz.

3.3. Interfaz hardware ethernet

En este apartado se va a explicar el funcionamiento del módulo ethernet. El módulo ethernet tiene dos partes que interactúan con la interface ethernet. Una son los puertos de entrada de configuración del protocolo TCP/IPv4 y otra parte son los puertos relacionados con la transmisión y recepción de datos. Todo este módulo está basado en el protocolo Gigabyte ethernet.

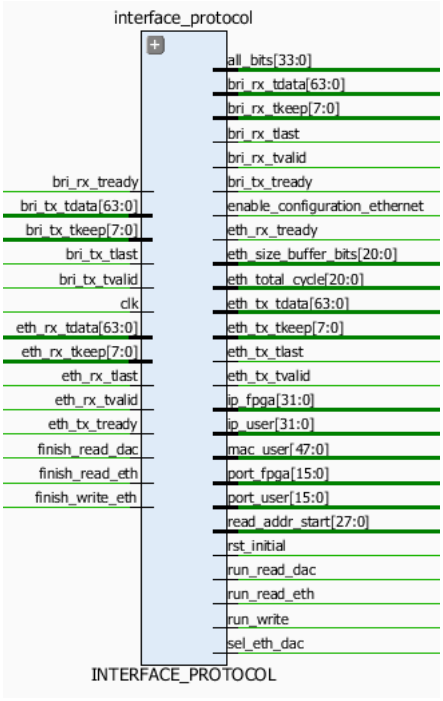
3.3.1. Configuración hardware ethernet del protocolo TCP/IPv4

En la configuración TCP/IPv4 al ser una conexión punto a punto entre el PC y la FPGA, la única variable que tiene que tener un valor inalterable, es decir, asignado con el valor de 48 bits en el código, es la MAC. Para cambiar la MAC, se tendría que acceder al código, cambiar la línea por la que se quiera introducir y volver a generar el bitstream (el archivo que se carga en el dispositivo). Para la configuración del protocolo TCP/IPv4 ambos dispositivos tienen que conocer sus direcciones físicas. Esto está explicado en el apartado de la conexión punto a punto. El resto de parámetros del protocolo se configuran con un valor que viene transmitidos vía ethernet por el programa software de Matlab. Solo se cambia este valor si el usuario quiere o porque se ha hecho un reset del sistema cuyo valor de defecto será todas a cero .

3.3.2. Transmisión y recepción de datos

Los puertos relacionados con la entrada y salida de datos se dividen en dos grupos, los puertos de entrada de datos y los puertos que transportan las señales de control. Tanto el puerto de entrada y salida de datos son de 64 bits. Estos bits transportan datos cuando el puerto que contiene la señal de validación de datos está activo. La transmisión o recepción de datos se termina cuando el bus de la señal de control de fin de transmisión esté activo, es decir, en el momento en que esté activa esta señal de control, será el último ciclo de reloj que entren datos válidos. El tamaño del envío o recepción de datos total de cada ciclo de escritura o lectura es el mismo en ambos extremos del canal.

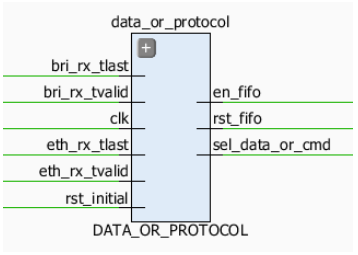
3.4. Función del módulo INTERFACE_PROTOCOL



En esta parte se va a explicar cómo trabaja el módulo INTERFACE_PROTOCOL, que es el módulo que recibe los comandos generados desde el programa software que controla el tráfico de datos en nuestro sistema. Lo más importante que vamos a explicar es cómo discernir que los datos que entran por el puerto ethernet son datos o comandos que transportan datos para configurar registros internos, ya sea variables del protocolo TCP/IPv4, configuración de contadores u órdenes de lectura y escritura en la DRAM, etc.

3.4.1. Datos o comandos

Este módulo está compuesto por una máquina de estados (ver los diseños en anexos: Máquinas de Estados) que gestiona los datos que llegan por los puertos de entrada de la interfaz ethernet.



Tanto los buses que transportan los datos como las señales de validación de datos como de fin de datos que le acompañan son almacenadas en riguroso orden temporal dentro de unas FIFO's hasta que sean procesadas.

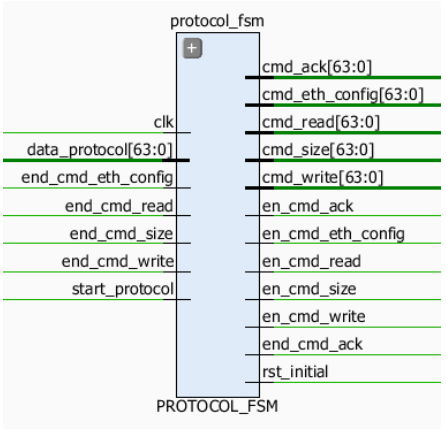
Cabe destacar que, para no tener que hacer uso de cabeceras y empaquetar los datos, se ha utilizado una forma de discernir entre recepción de datos y comandos por medio de

la distancia de llegada entre el primer y último dato (señalizado por la variable que indica fin de la recepción de datos), aclarando que los datos tienen que llegar en ciclos de reloj consecutivos. Utilizando esta forma de discernir la llegada de datos vía la interfaz ethernet, vamos a tener tres supuestos. El primer supuesto es que si la distancia es menor de 8 ciclos de segundo, los datos se desechan, es decir, se borrará el contenido automáticamente del interior de las FIFO's. El segundo supuesto es que si la distancia es igual a 8 ciclos de reloj, significa que los datos provenientes de la interfaz ethernet son comandos. Por último, si la distancia es mayor a 8 ciclos, los datos que llegan son datos para guardar dentro de la DRAM.

Una vez tenemos diferenciado los datos de los comandos, en los siguientes puntos se va a explicar cómo se tratan los comandos ya que si este módulo, por medio de lo explicado anteriormente detecta que los datos llegados son datos para escribir en la DRAM, automáticamente se dejan pasar los datos en el mismo orden de llegada para que el módulo INTERFACE_BRIDGED_INPUT los gestione.

3.4.2. Comandos

En este apartado se explica el procesado de cada uno de los comandos que se reciben desde el programa ejecutado en Matlab y cómo son redirigidos a sus módulos de destino para su posterior ejecución.



Un comando es el primer paquete de 64 bits que llega a través de la interfaz ethernet. Consta de una cabecera de 4 bits junto a una carga útil de 60 bits. Hay implementados dentro de diferentes máquinas de estado trece tipos diferentes de comandos (trece tipos diferente de valor de cabecera) agrupados en diferentes tipos de ciclos de acciones.

El primer comando es el de valor de la cabecera igual a cero. Este comando provoca que la señal de reset interno que tienen todos los módulos implementada pase a nivel alto provocando que todas las variables y todas las máquinas de estado implementadas vuelvan a un estado inicial o en el caso de las variables, a un valor por defecto.

El segundo grupo de comandos que vamos a explicar son los comandos de configuración de variables relacionadas con el protocolo TCP/IPv4 (cabeceras con valor inclusivo desde el siete hasta el valor trece). La cabecera de valor siete corresponde al comando que habilita la configuración de las variables de ethernet y con el valor ocho deshabilitamos la configuración de estas variables. Las cabeceras con el valor de nueve al trece transportan el valor de las variables de IP del usuario, MAC del usuario, puerto usado por el usuario, IP asignada a la FPGA y puerto asignado a la FPGA. Como se ha explicado en puntos anteriores del trabajo,

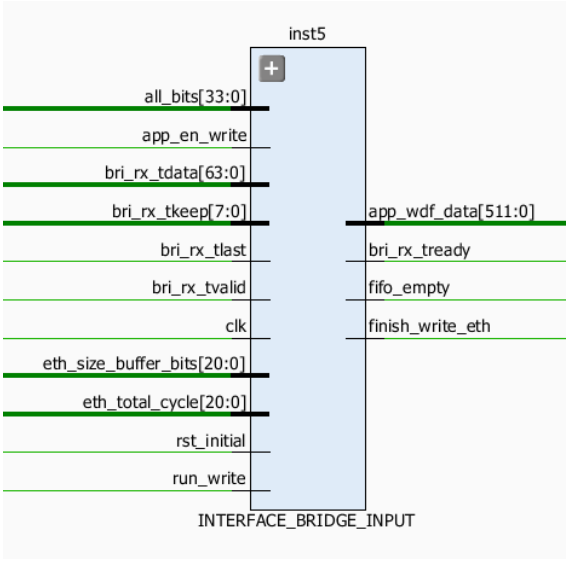
la MAC de la FPGA está metida en código y no es configurable.

En un tercer grupo de comandos encontramos los comandos que transportan el valor de las variables que configuran los contadores internos (cabecera de valor cinco). Dentro de la carga útil de este comando (60 bits de carga útil) encontramos el total de bits que se van a escribir o leer de la memoria DRAM.

En el último grupo encontramos los comandos que ejecutan los ciclos de escritura o lectura de la DRAM. En escritura tenemos solo el comando de cabecera de valor uno, que activa la señal de principio de ciclo de escritura, que se propagará hacia los módulos que intervengan en el ciclo de escritura. En lectura tenemos varios comandos. El comando de valor de cabecera dos es el que ejecuta la orden de empezar a leer los datos desde la DRAM hasta el módulo que gestiona los datos hacia el DAC. El envío de datos vía DAC es cíclico, es decir, una vez se termine la lectura de la DRAM, volverá a empezar otra vez salvo que llegue el comando de valor de cabecera cuatro que es el comando que parará la lectura cíclica de la DRAM. Con el valor tres de cabecera tenemos el comando que ejecuta la orden de lectura de la DRAM y envío de datos vía ethernet. Esta lectura a diferencia de la anterior no es cíclica, y una vez se llegue a la posición final de lectura de la DRAM se finalizarán todos los procesos.

3.5. Función del módulo INTERFACE_BRIDGE_INPUT

Este módulo interconecta la salida de datos de la interfaz protocolo con la interfaz de la memoria DRAM. Las funciones principales de este módulo son respetar los tiempos de escritura impuestos por la controladora de la DRAM y agrupar los datos llegados de la interfaz ethernet en grupos 64 en grupos de datos de 512 bits, como se requiere en la interfaz delcore MIG que controla la DRAM (explicado en el apartado de la DRAM).

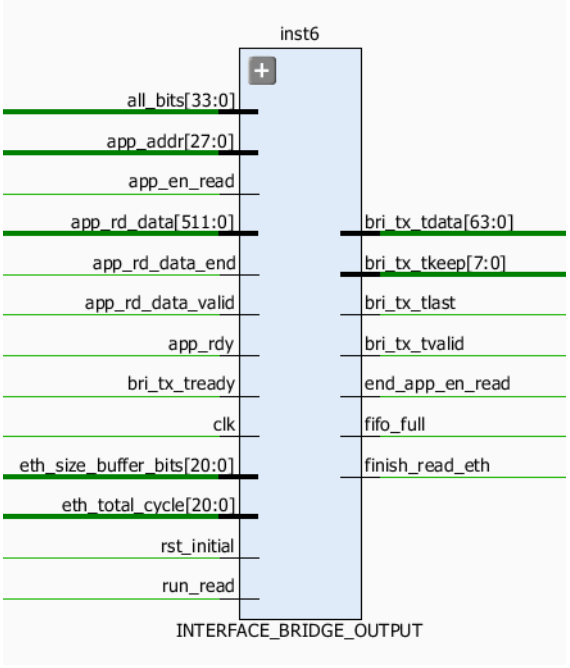


Respetar los tiempos de escritura es respetar las señales que regulan la escritura de datos en la DRAM. Estas variables se gestionan todas en el módulo INTERFACE_DRAM, pero afectan al flujo de datos, que proviene de este módulo. Esta danza sincrónica hace que sea posible escribir los datos dentro de la DRAM sin que se pierda ningún dato.

El otro aspecto fundamental de este módulo es agrupar los datos en un bus de salida de 512 bits. Esto se produce en la FIFO interna que hay dentro de este módulo. A la FIFO se introducen los datos de 64 bits y, cada 8 datos de 64 bits, los agrupa en un solo dato de 512 bits que envía por el bus de salida dirección a la DRAM. Este agrupamiento en sí, es una limitación que hace que los envíos de datos desde el programa software de Matlab tienen que ser múltiplos de 512 bits, aunque se ha implementado en el software de Matlab medidas de prevención para que todos los ciclos de escritura sean múltiplos de 512 bits (relleno de ceros si es necesario al transmitir).

3.6. Función del módulo INTERFACE_BRIDGE_OUTPUT

Este módulo interconecta la salida de datos del Core MIG hacia la interfaz protocolo para que sean transmitidos vía ethernet. En cuanto al tratamiento de los datos, funciona al contrario del módulo INTERFACE_BRIDGE_INPUT. El bus de entrada de datos procedente del MIG es de 512 bits, y hay que separarlos en grupos de 64 bits como requiere la interface ethernet. Esto crea uno de los principales problemas que se han tenido que solucionar para no colapsar la salida ethernet. El flujo de datos máximo de salida es de 64 bits por ciclo de reloj y de entrada de datos de 512 bits, es decir, entran más datos de los que salen de este módulo. Para ello la FIFO que realiza este trabajo tiene configurada internamente una memoria de una alta capacidad. Pero una vez llega a su límite de capacidad, habilita una señal que indica al módulo INTERFACE_DRAM, que pause el envío mientras se vacía la memoria enviando los datos con las dimensiones de 64 bits vía ethernet. En cuanto a gestión de señales procedentes del Core que acompañan a los datos, solo nos interesa la señal que da validez al dato que llega procedente de la DRAM.



3.7. Función del módulo INTERFACE_DRAM

En este apartado del trabajo se va a hacer un breve resumen sobre aspectos técnicos relacionados con la controladora DRAM. También se va a exponer una descripción detallada del comportamiento de la Interfaz hardware de memoria 7 Series FPGA.

3.7.1. Comportamiento de la interfaz MIG

En cuanto al kit DRAM que ofrece la 7 Series tienen en su interior tiene una memoria dram DDR3 de la empresa Micron Technology (Modelo MT8JTF12864HZ 1G6G1) de un Gigabyte de capacidad como las de la imagen siguiente.

Las principales características que nos interesan saber del funcionamiento del kit son la velocidad de lectura y escritura de datos y su capacidad. En cuanto al bus entrada o salida de datos es de 512 bits. Estos 512 bits a 200 MHz son la misma tasa máxima que soporta el kit, es decir 64 bits por 1,600 MHz hace que podamos suministrar datos para el máximo rendimiento del DRAM.

Como hemos mencionado anteriormente, el kit tiene una memoria de 1 Gigabyte. Este gigabyte se divide en celdas de 64 bits, por lo tanto, 512 bits son 8 celdas, esto hace que, en cada periodo de escritura o lectura, escribiremos o leeremos 8 celdas en cada ciclo de reloj. Esto hace un total de 15,625,000 de posiciones.

La interacción con el MIG se hace a través de comandos que controlan los ciclos de escritura y lectura. Estos comandos entran por sus respectivos puertos. En el interior de estos puertos estos comandos son almacenados en unas FIFO's para su posterior procesado. Esta forma de trabajar tiene la peculiaridad de que, aunque en el ciclo posterior en que las señales de control indican que no se puede interactuar con la dram, puedes dejar comandos

ejecutados, es decir cargado en las FIFO's para que cuando se vuelva a activar el sistema, perder el menor tiempo posible en la reanudación tanto en la escritura como en la lectura.

3.7.2. Interfaz de memoria 7 Series FPGA

En este apartado vamos a explicar las partes que componen la interface que nos ofrece la empresa Xilinx para las tarjetas de la serie 7. El programa INTERFACE_DRAM que se explica en el punto siguiente, está diseñado para interactuar con el bloque que contiene el interfaz usuario como se observa en la imagen siguiente:

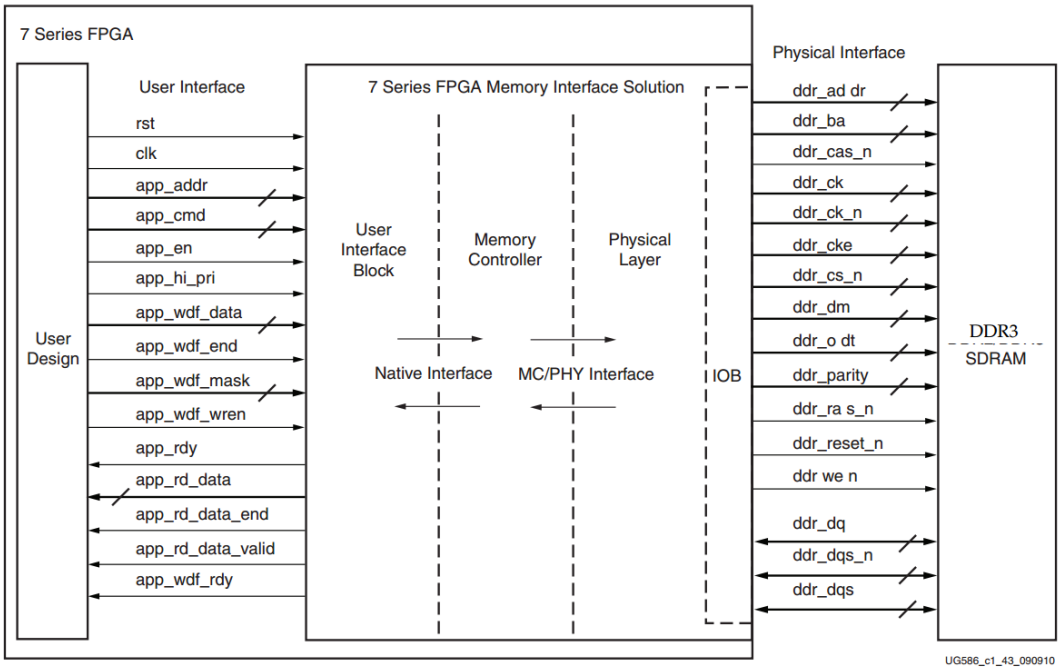
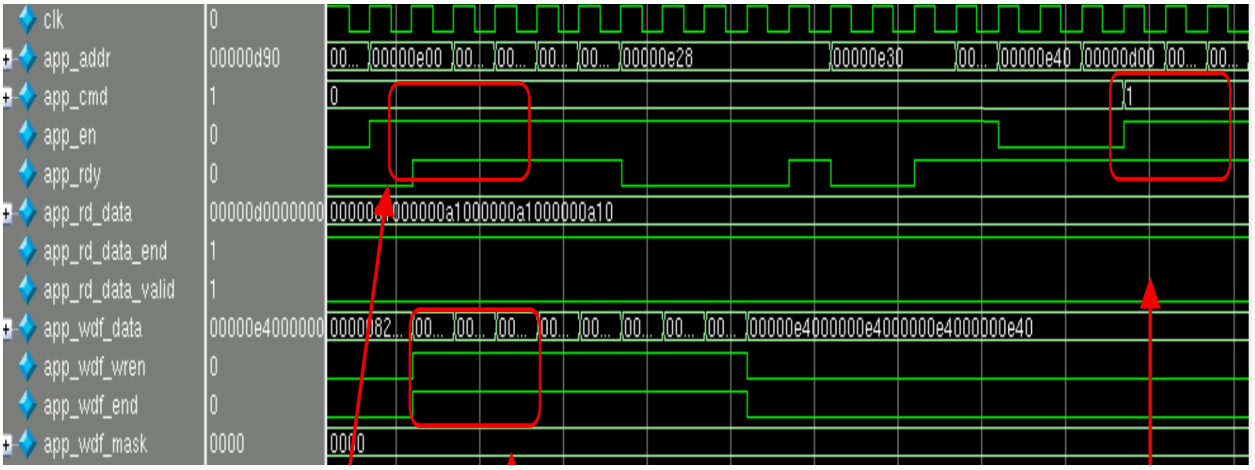


Figure 1-25: **7 Series FPGAs Memory Interface Solution**

En este bloque contiene una serie de bloques internos, que procesan los datos a las velocidades de transferencia requerida por la interfaz física de la DRAM. Una vez los datos están en la interfaz de usuarios, son almacenados momentáneamente por programas que contienen memorias internas, para su posterior procesado. Esto se debe a que la salida o entrada de los pines que conectan con la DRAM (En la imagen la interfaz física que se encuentra a la derecha), la salida o entrada de datos son de 64 bits, pero a 1.6 GHz. Esta frecuencia es 8 veces mayor que la frecuencia de entrada de datos que es de 512 bits a 200 MHz. A los buses de datos les acompaña una serie de buses de control como viene siendo habitual en estos sistemas de transporte y almacenamiento de datos.

3.7.3. Módulo INTERFACE_DRAM

El módulo INTERFACE_DRAM consta de cinco módulos internos. Todos ellos a excepción del módulo que genera el valor de la dirección están programados con una máquina de estados, que se puede consultar su diseño en el apartado Anexos: Máquinas de estado. Para explicar detalladamente la función de cada uno de los módulos que conforman la INTERFACE_DRAM, utilizaremos una imagen extraída de la documentación proporcionada por la empresa Xilinx, para ir desglosando el comportamiento de las señales más importantes a lo largo de un ciclo de escritura y lectura.



Modulo GLOBAL

En este módulo consta de una máquina de estados programada la cual, controla el bus de indicación de periodo de escritura o lectura a la controladora 7 Series. Este comando (el bus del comando app_cmd) está recalcado por el recuadro amarillo en la imagen siguiente:

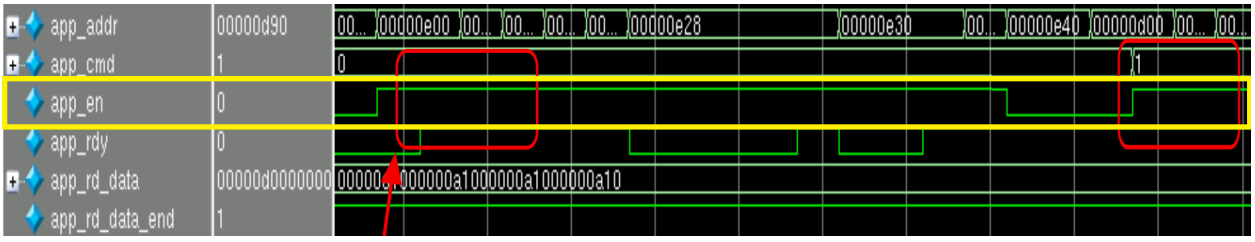
En los dos círculos blancos de la imagen anterior, podemos observar como el comando app_cmd indica los periodos de escritura, indicado con un cero en el círculo blanco de la izquierda y el periodo de lectura, indicado con un 1 en el círculo blanco de la derecha.

A su vez este módulo recibe indicaciones de finalización de procesos tanto exteriores a esta interfaz como internos. Esto se debe a que los contadores de flujo de entrada y salida de datos, están implementados en otras interfaces, aunque sea este el módulo que controle la interfaz series 7.

Módulo READ

Este módulo, por medio de una máquina de estados, gestiona el proceso de lectura de la memoria DRAM. En lectura la señal más importante que tenemos que gestionar es la app_en que es la encargada de validar la posición de petición de lectura que se ha enviado a la interfaz MIG. Aunque este módulo gestiona las señales de lectura, está controlado por el módulo global como todos los módulos que se encuentran dentro de este módulo INTERFACE_DRAM.

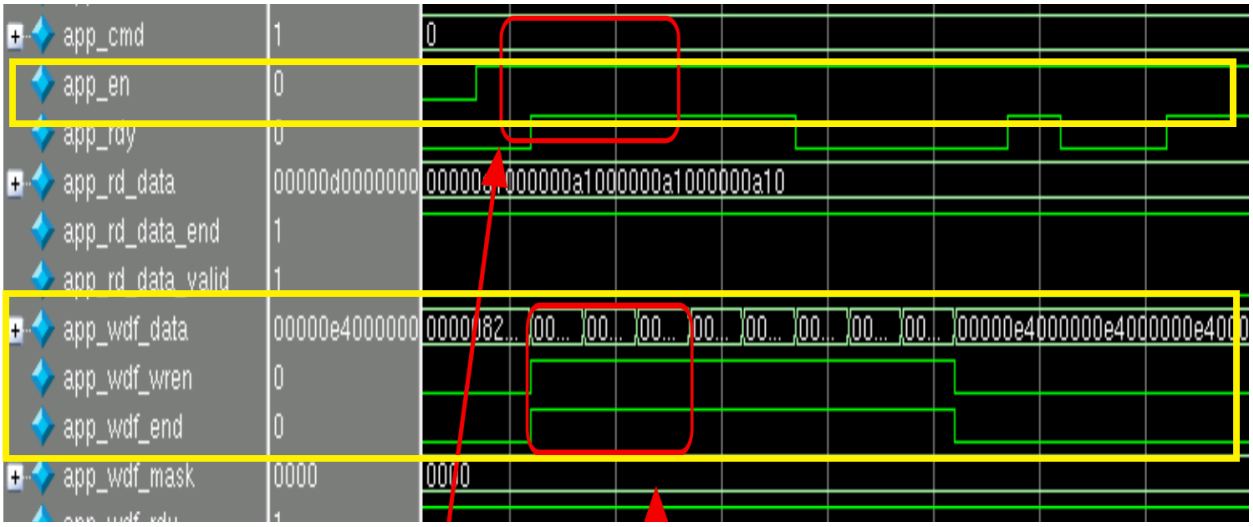
Cabe destacar que hay un periodo de demora entre la validación de la lectura y a entrega del dato requerido como se ve en la imagen siguiente:



En el recuadro de la izquierda observamos la petición de lectura y en el recuadro de la derecha la llegada del dato con sus correspondientes señales de validación. Este periodo de demora hay que tenerlo en cuenta para la recepción de datos en las FIFO's tanto del módulo INTERFACE_BRIDGE_OUTPUT como el módulo INTERFACE_BRIDGE_OUTPUT _DAC. Para ello la señal que indica que la FIFO está llena, se ha configurado para que lo indique con un periodo de antelación, así no se producirá un desbordamiento de la memoria con la pérdida de información que esto conllevaría.

Módulo WRITE

Este módulo es controlado por una máquina de estados que gestiona tanto señales internas propias de esta interfaz, como de módulos exteriores. La principal función de este módulo es, al igual que el módulo READ, controlar la señal de validación de posición de escritura. Esta señal es compartida con el módulo anterior, es decir, en los periodos de escritura será controlada por este módulo y en los periodos de lectura será controlada por el módulo READ.



A su vez, como se observa en la imagen anterior, se observan una serie de señales de validación de escritura. Dos señales acompañan al bus de datos (_wren y _end). En cuanto a la gestión de datos, en periodos de refresco de la memoria, será otro modulo el que gestione estos puntos críticos, como se explica en el punto WRITE_RDY_SHIELD.

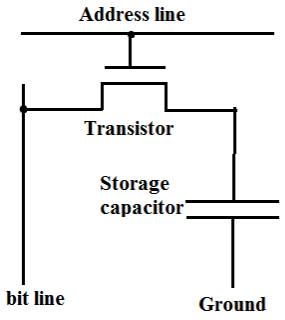
Módulo ADDRESS

Este módulo se encarga de generar la posición donde se almacenará el dato de entrada que llegue por el bus de datos como se observa en la imagen siguiente:

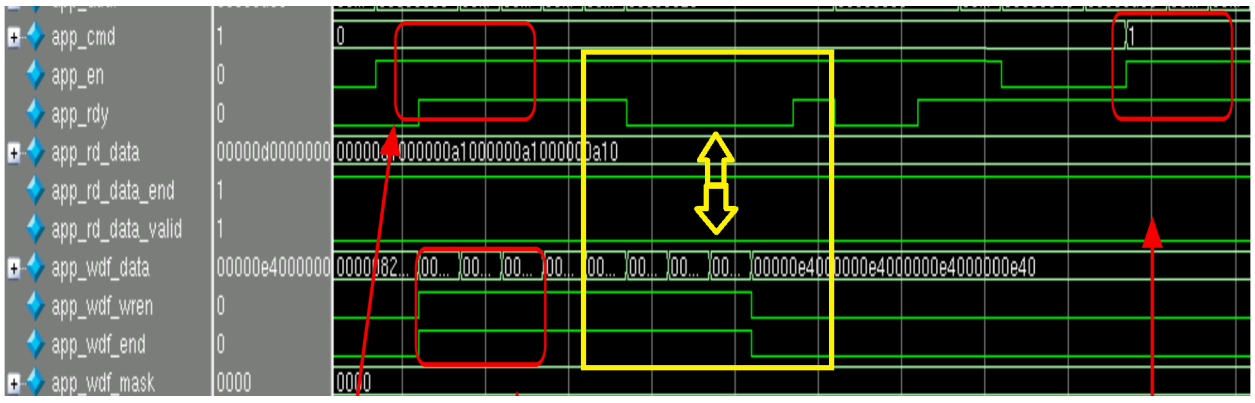
Existen dentro de este módulo dos módulos internos que generan un valor de posición de memoria que será transmitido por el bus de 28 bits asignado. El módulo que actúa en el periodo de escritura, siempre parte de la posición inicial de cero. Sin embargo, el de lectura parte de la posición inicial que le llegue desde el programa software. Esto posibilita leer direcciones concretas desde el Matlab y poder transmitir vía DAC, diferentes tipos de mensaje almacenados en la memoria DRAM. Todo este módulo ha sido programado con lógica secuencial y combinacional, como se puede comprobar en el anexo de código.

Módulo WRITE_RDY_SHIELD_512 y WRITE_RDY_SHIELD

Estos módulos se encargan de proteger en los ciclos de escritura los datos y las señales de control que se ejecutan en el ciclo de escritura de los periodos de refresco de la DRAM. La diferencia entre ambos módulos es la dimensión de la señal que se quiere guardar. Físicamente, una memoria DRAM donde se almacenan los datos, son condensadores como se ve en la imagen siguiente:



Cada un periodo de tiempo, estos condensadores si están cargados, es decir, si significa que están almacenando un bit igual a 1 en su interior, tienen que volverse a cargar hasta el nivel óptimo de carga para mantener la información almacenada. Estos periodos de refresco, no son notificados con anterioridad por ninguna señal de control internas. Esto acarrea que cuando se producen, puedes estar enviando a escribir datos y estos serán desechados o guardados en un orden diferente. En cuanto a las señales de control que transmitimos a la interfaz MIG, como se ha explicado en el apartado anterior Comportamiento de la interfaz MIG, pueden que sean almacenadas en las FIFO's de comandos y que produzcan un descuadre en la escritura, provocando el mal funcionamiento de la interfaz controladora de la DRAM. En la imagen siguiente podemos observar cómo la señal que indica el periodo de refresco (app_rdy) está indicando que está en periodo de refresco y cómo se continúa enviando comandos de escritura (señales indicadas con el recuadro amarillo). Esto es lo que queremos evitar con estos dos módulos.



3.8. Función del módulo INTERFACE_BRIDGE_OUTPUT_DAC

El módulo INTERFACE_BRIDGE_OUTPUT_DAC es el que se encarga de adaptar los datos de salida de la dram hacia los pines que conectan con el DAC. Las dos principales funcionalidades de este módulo es adaptar el tamaño de trama e interconectar con la interfaz de salida de datos que trabaja a diferente velocidad de reloj. Para ello hay conectada una FIFO asíncrona que une las dos zonas de dominio de reloj diferentes. Esto puede acarrear problemas a la hora de trasmitir datos. Como se ha explicado en puntos anteriores tendremos una tasa de llegada de datos bruta a la FIFO de 102.4 Gbits/s. A esta tasa, como hemos mencionado en otros apartados hay que descontarle los periodos de refresco de la memoria RAM. La tasa de salida de datos, al ser ininterrumpida será de 80 Gbits/s. Aunque la tasa de llegada de datos es mucho mayor que la de salida, la llegada de datos no es continua, y esto puede acarrear problemas de abastecimiento de datos, es decir, en el arranque de la transmisión. Para ello se ha implementado una demora en la señal de activación de salida de datos, para que la FIFO esté llena antes de trasmitir, así nos aseguramos que el flujo de salida de datos desde el principio de la transmisión es constante. En cuanto a la interacción con la DRAM, este módulo controla la salida de datos mediante contadores, una vez han llegado estos contadores al valor que ha sido indicado por el usuario, envían la señal de fin de lectura tanto al módulo que controla la DRAM como al módulo protocolo que es el que controla los ciclos de lectura del sistema.

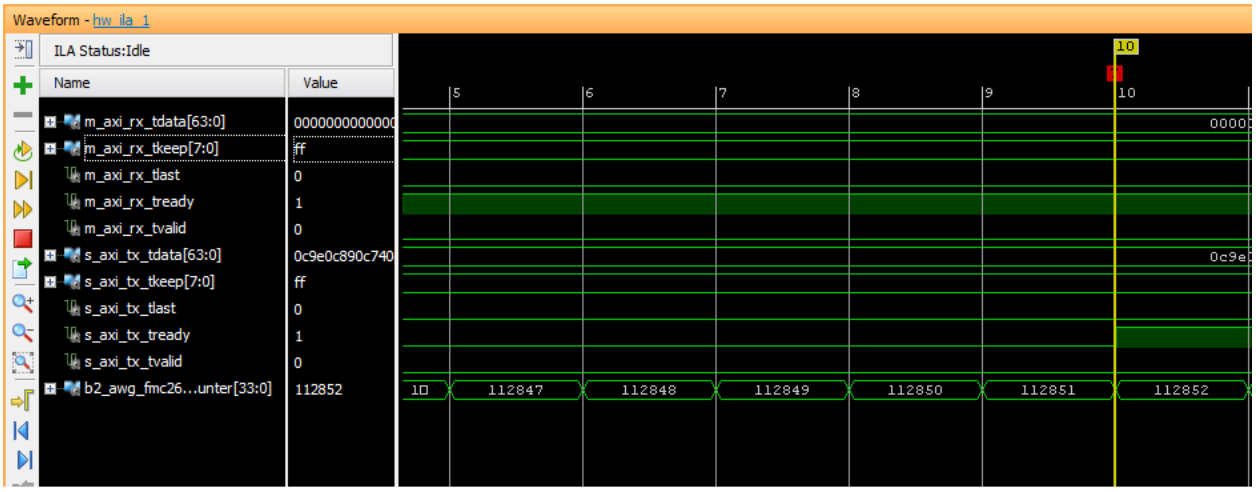
Capítulo 4

Rendimiento del sistema Hardware

En este capítulo se va a explicar los resultados del funcionamiento del sistema implementado en este trabajo. Las prestaciones que nos interesa monitorizar son las de los programas Hardware. En estos programas tenemos dos puntos para analizar que son las prestaciones del Core ethernet y las prestaciones de lectura del Core de la memoria DRAM.

4.1. Prestaciones del Core ethernet

El rendimiento del Core ethernet se va a calcular con la tasa de envío de datos. No se hace con la de llegada, ya que el periodo entre ciclos de llegada va a estar supeditada a la velocidad de procesamiento de datos del dispositivo transmisor. En la siguiente captura podemos observar el número de ciclos que tarda desde que se ha enviado la última trama de datos de 64 bits hasta que el canal vuelve a estar habilitado para la transmisión.

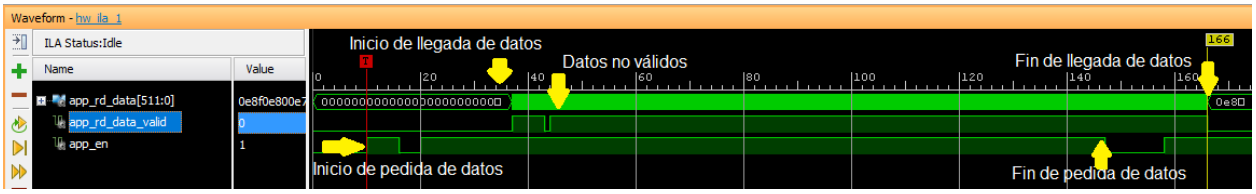


Los ciclos hábiles por segundo los calculamos de esta forma. Para transmitir 65,024 bytes que es el tamaño máximo del buffer de ethernet, el Core ethernet necesita 8,128 ciclos de lectura de datos y 112,852 ciclos de reloj para transmitirlos, que es la cifra que observamos en la captura anterior. Esto está medido con un periodo de reloj de 200 MHz, por lo tanto, tendremos que un ciclo de transmisión de datos vía ethernet se compone de 120980 ciclos o lo que es lo mismo 604.9 microsegundos. Esto nos da una tasa de envió neta de 107.5 Mbyte/s.

4.2. Prestaciones del Core DRAM

En cuanto al Core DRAM, tenemos dos tasas que monitorizar para ver el rendimiento, la tasa de escritura y la tasa de lectura de datos. En la tasa de escritura tenemos periodos de refresco explicados en el apartado de la memoria DRAM. Pero ya que la llegada de datos se produce con una velocidad muy inferior a la máxima que nos puede dar en escritura la DRAM, solo nos vamos a centrar en la tasa de lectura de datos.

La tasa en bruto de llegada de datos seria de 200 MHz por 512 bits de datos que hacen un total de 102.4 GBits/s. En la lectura de datos tenemos una demora de tiempo entre el transcurso de la demanda de datos y la entrega de estos mismos. Esto hace que tengamos una tasa en bruto muy diferente a la tasa en neto o real como se puede observar en la captura siguiente:



De la captura anterior podemos calcular las tasas de lectura de datos del Core DRAM. Recordamos que como en el caso anterior las capturas se realizan con un reloj de 200 MHz y que en cada ciclo se entregan 512 bits. De un ciclo de pedida y entrega de datos tenemos que desde la pedida hasta la llegada hay 36 ciclos de reloj. Una vez llegan los datos tenemos que de 130 ciclos solo están hábiles 129 ciclos. Por lo tanto, de un ciclo total de lectura de 166 ciclos tenemos 129 ciclos efectivos de entrega de datos. Esto nos lleva a tener una tasa de lectura neta de datos de 79.58 GBits/s.

Capítulo 5

Prueba del generador de formas de onda arbitrario

En este capítulo se va a probar el sistema conjuntamente a un montaje experimental que se ha realizado dentro de un laboratorio de la UPV.

5.1. Características del montaje

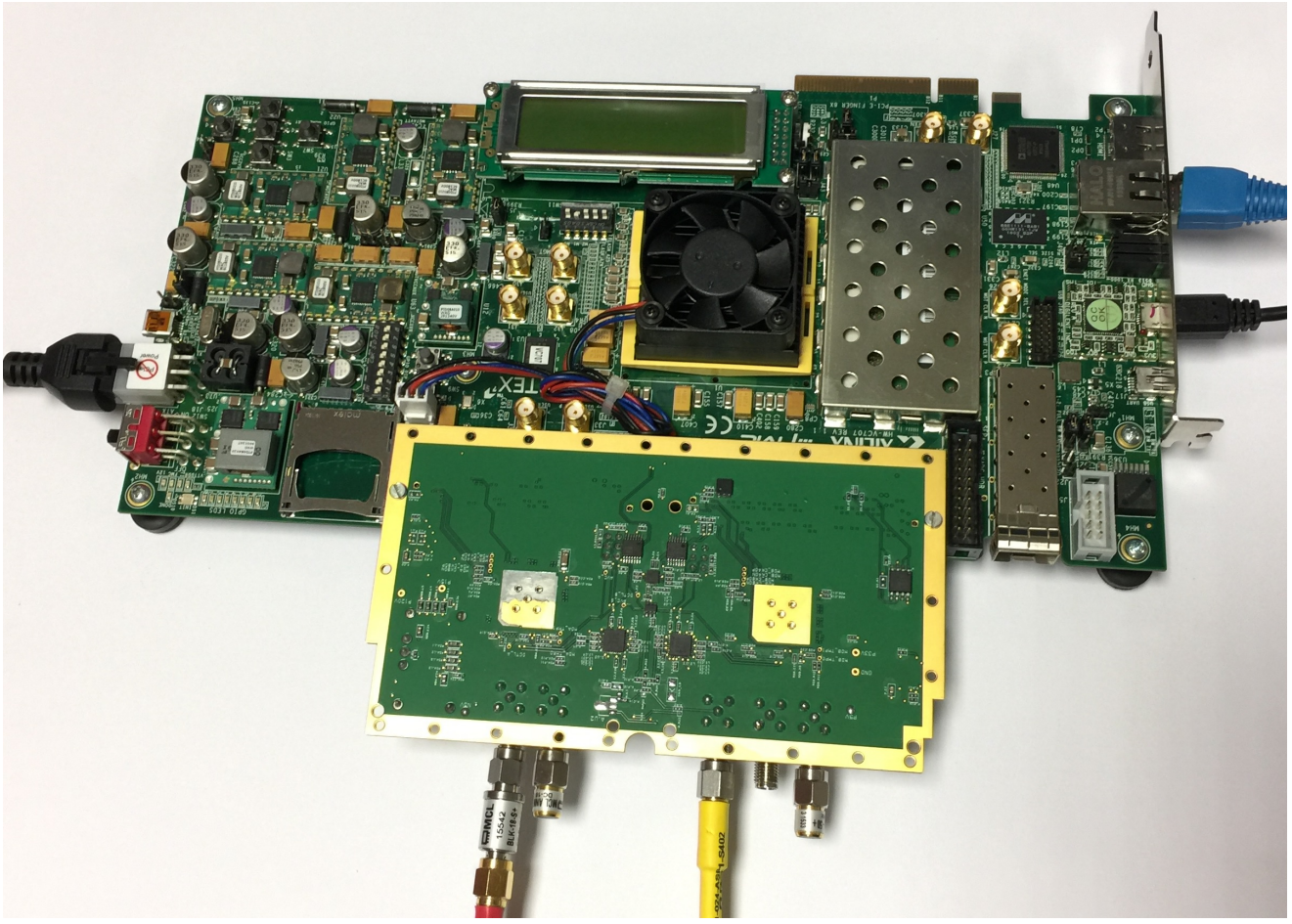


Foto de la FPGA Virtex con el conversor EURVIS

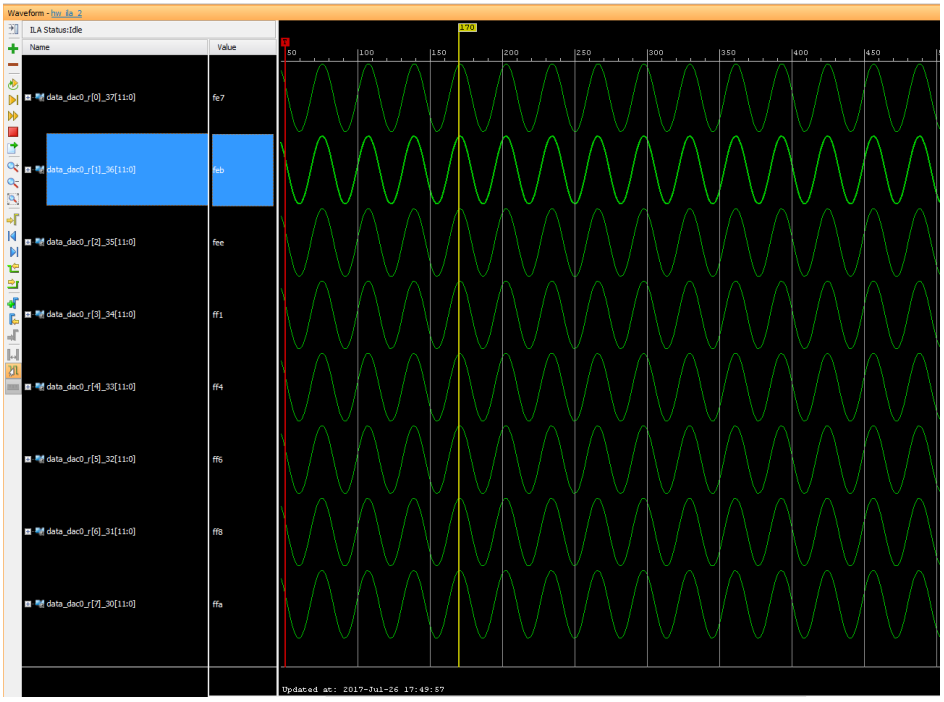
El montaje consta como podemos observar en la imagen superior es de un sistema conformado por una FPGA Xilinx del Modelo Virtex Serie 7 y un DAC (Digital Analogic Converter) marca EUVIS y modelo MD657B.

La característica principal del DAC que nos interesa saber es cuantas muestras por segundo necesita para abastecer el sistema de transmisión. Este DAC necesita 12 bits por canal con un total de 16 canales. Esto hace que en cada ciclo de reloj tengamos que entregarle continuamente y sin interrupción un total de 192 bits a una frecuencia de reloj de 312.5 MHz. Esto hace que el DAC necesite una tasa neta de entrega de datos de 60 GBits/s. Esta tasa está por debajo de la tasa neta que entrega la memoria DRAM que recordamos es de 79.58 GBits/s. Capturas a la entrada del DAC.

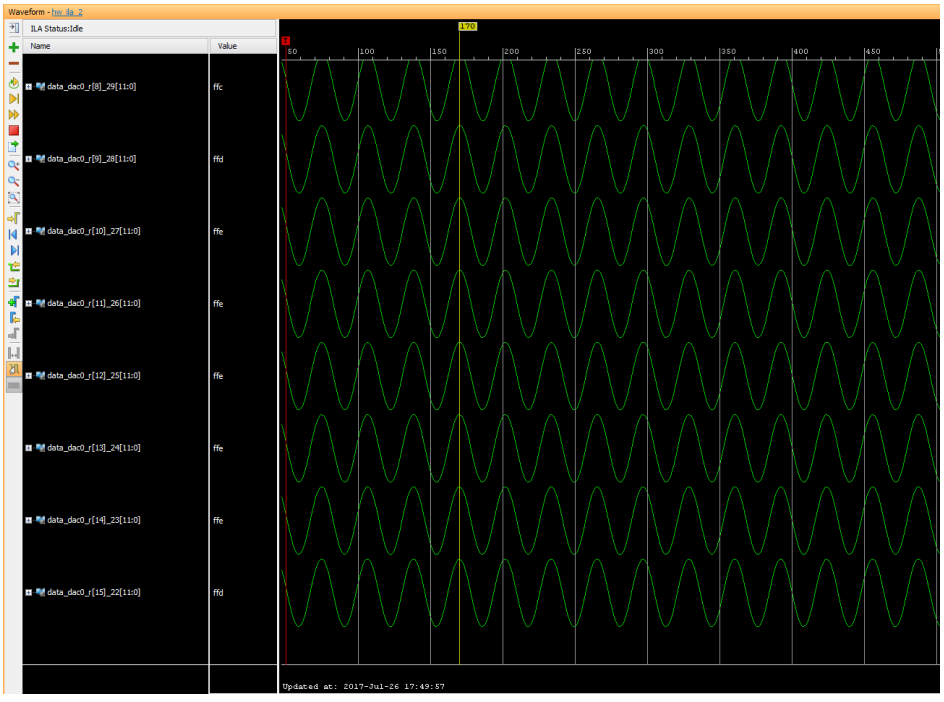
Las capturas que vamos a enseñar a continuación están realizadas a la salida de la FIFO que interconecta las dos regiones de reloj como se ha explicado en el apartado donde se explica el módulo INTERFACE_BRIDGE_OUTPUT_DAC. A la salida de esta FIFO

se asigna el contenido de los datos que se transmitirán por cada uno de los canales. Para poder visualizar la prueba, se ha programado una función que consta de una onda senoidal con una frecuencia 16 veces menor que la tasa de muestreo del DAC que es de 5 Gsps. Esto hará que tengamos una onda senoidal perfectamente definida en cada uno de los canales.

Captura del DAC 0 donde vemos los 16 canales:

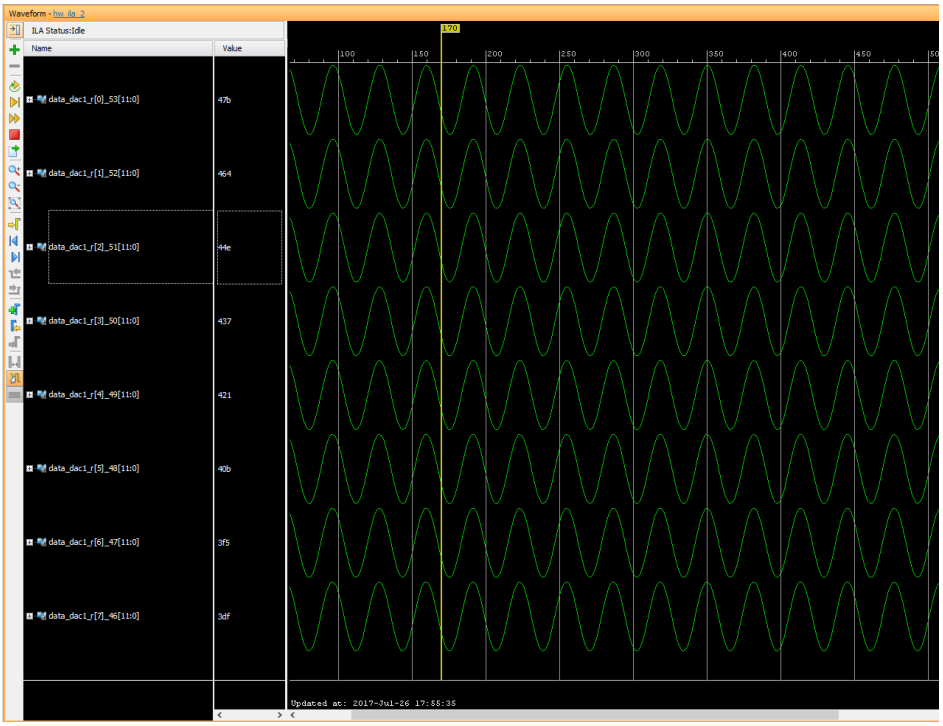


DAC 0 canales del 0 al 7

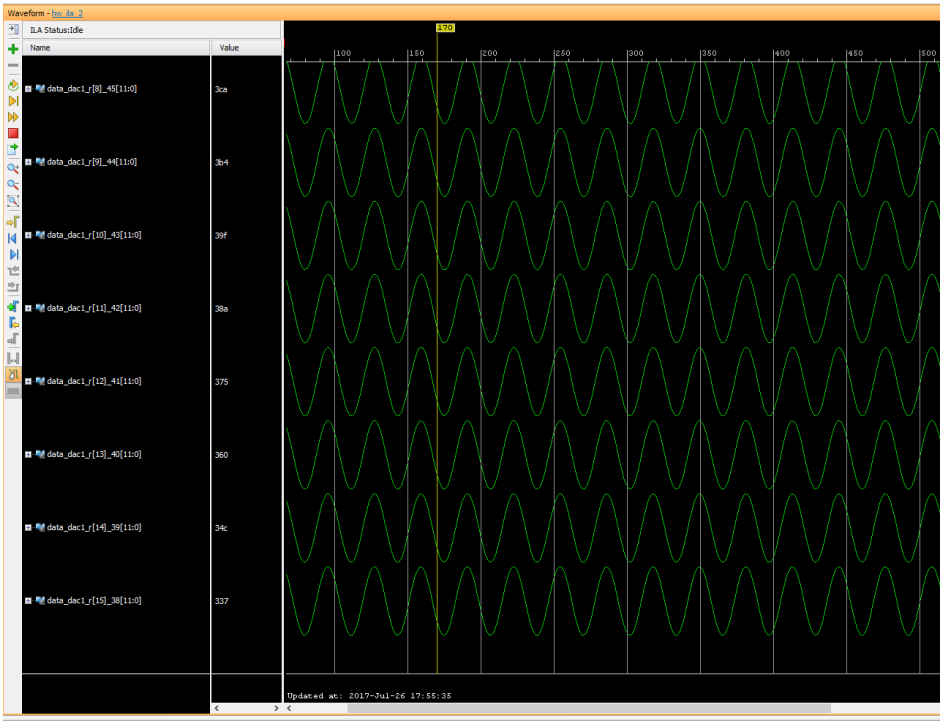


DAC 0 canales del 8 al 15

Captura del DAC 1 donde vemos los 16 canales:



DAC 1 canales del 0 al 7



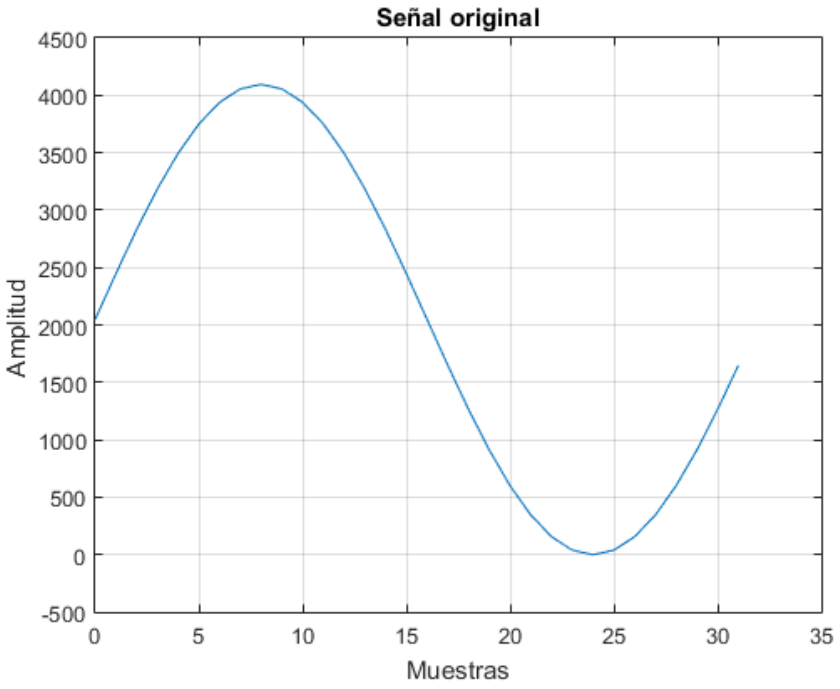
DAC 1 canales del 8 al 15

5.2. Capturas con instrumental de laboratorio

En esta sección vamos a explicar las capturas realizadas tanto en el osciloscopio como con el analizador de espectros. Para realizar estas capturas se ha transmitido una onda senoidal de 32 muestras de 12 bits cada una. Esto hace que al ser muestreadas a 5 GHz hace que en frecuencia real tengamos una señal senoidal de 156.25 MHz.

5.2.1. Señal original

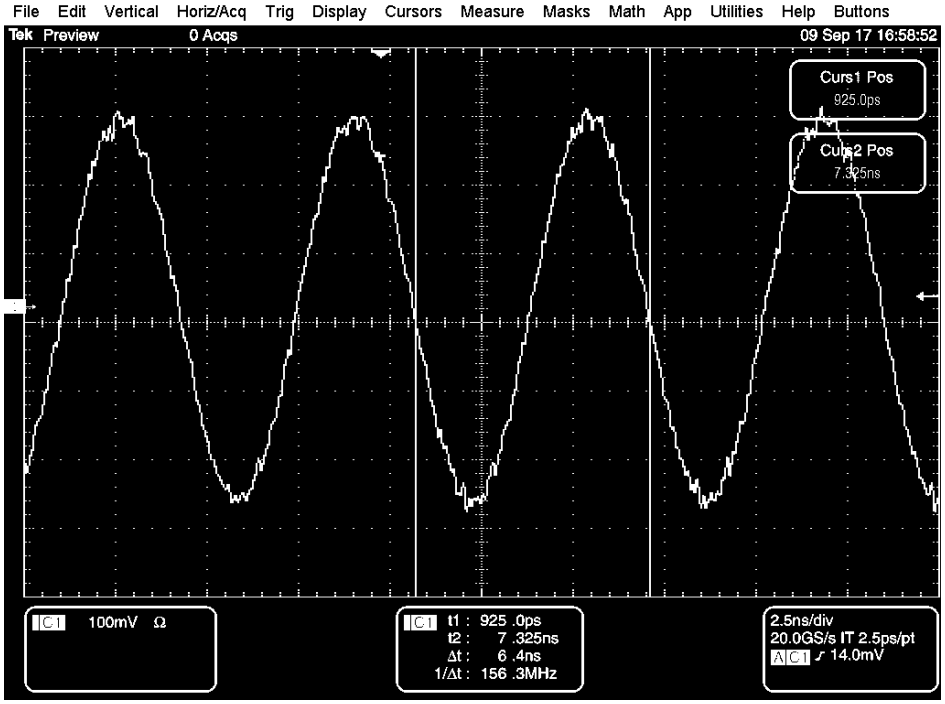
La señal original se va a programar con el programa Matlab. Es una señal senoidal, de 32 muestras y con una amplitud que va desde 0 hasta $2^{12} - 1$. La amplitud de 12 bits hace que aprovechemos todo el rango de bits disponibles por el conversor DAC.



Dibujo del seno

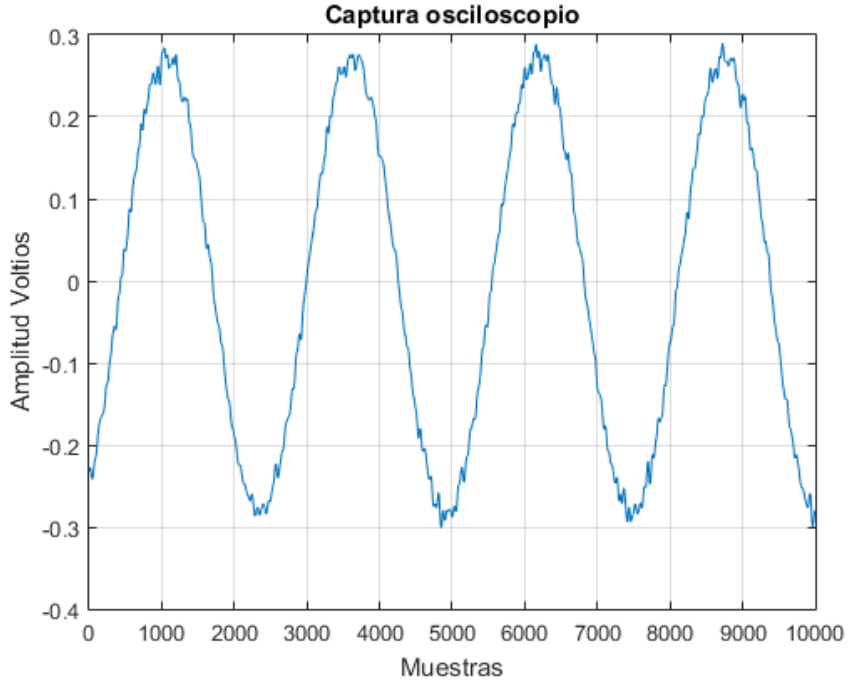
5.2.2. Captura con un osciloscopio

La captura de la onda se va a hacer con un osciloscopio digital de la marca Tektronix del modelo TDS7154B. Este osciloscopio tiene una frecuencia de muestreo de hasta 20 Gsps y con un ancho de banda de 500 MHz. La siguiente captura está realizada en el dominio temporal donde podemos ver el seno de 156.25 MHz.



Captura de pantalla del osciloscopio

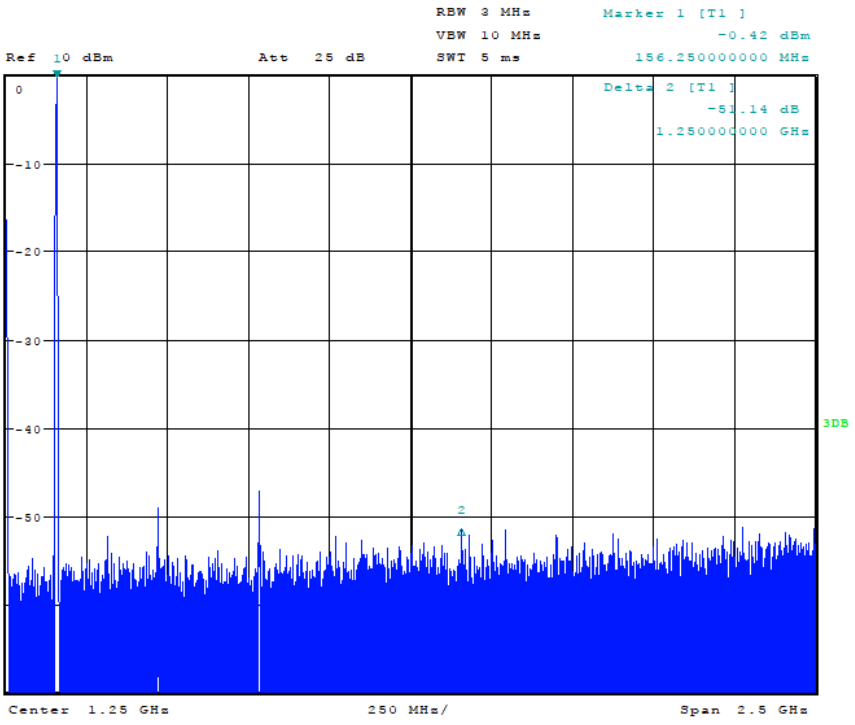
A continuación podemos ver el seno muestreado por el osciloscopio y representado en el programa Matlab.



Representación datos capturados con el osciloscopio

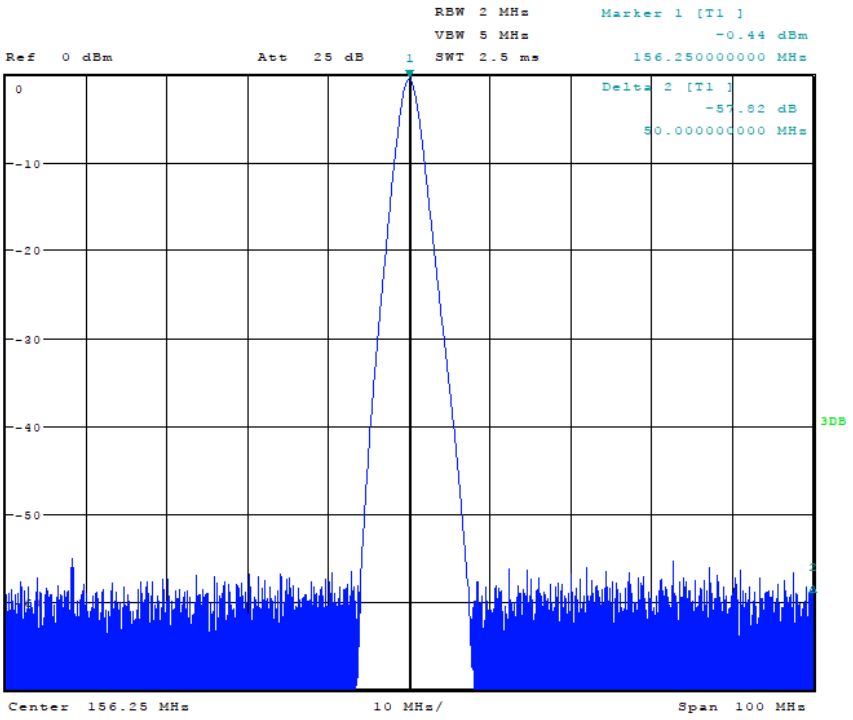
5.2.3. Captura con un analizador de espectros

La captura se va a realizar con un analizador de espectros de la marca Rohde & Schwarz, modelo FSQ. Este analizador de espectros captura señales desde 20Hz hasta 26.5 GHz. A continuación, podemos ver la captura en el dominio frecuencial donde observamos la señal senoidal en una captura de 2.5 GHz de ancho de banda y centrada en 1.25 GHz.



Captura analizadora de espectros de 2.5 GHz de ancho de banda

Por último, vemos la misma captura que anteriormente pero con un hancho de banda de la captura de 100 MHz.



Captura analizadora de espectros de 100 MHz de ancho de banda

Capítulo 6

Conclusiones

Finalmente en este último capítulo vamos a exponer los resultados relacionados con los objetivos del trabajo. También se va a hacer un análisis de la metodología y proceso de aprendizaje. Por último, se elaborará las conclusiones personales del trabajo fin de grado.

6.1. Análisis de objetivos

En primera instancia vamos a refrescar los objetivos marcados en el primer capítulo. Como mencionamos en este apratado, dividimos los objetivos en dos apartados, uno Software y otro Hardware. Los objetivos dentro de la implementación Software que se van a valorar en las conclusiones son:

- Configuración automatizada del sistema operativo Windows.
- Configuración automatizada de los parámetros de transmisión vía ethernet en la tarjeta de red.
- Elección y control de los ciclos de escritura y lectura de los programas Hardware implementados en la FPGA.

En cuanto a los objetivos dentro de la parte implementada en la FPGA son:

- Los programas deben tener una probabilidad de error igual a cero.
- La escritura y lectura de la DRAM deben ser configurable en cuanto al tamaño de las tramas de datos.
- Los programas deben trabajar sincronizados entre ellos mismos y sin ningún tipo de problema de propagación tanto de las señales de control como las señales de transporte de datos.
- El conjunto de programas Hardware deben cumplir las tasas de datos que se desea para abastecer el conversor de 5 Gsps.

El programa diseñado e implementado en Matlab es muy sencillo de utilizar, ya que se ha intentado cumplir los tres puntos anteriores. En cuanto a la configuración del protocolo TCP/IPv4, está totalmente automatizada por medio de la ejecución de los comandos. Estos comandos que habitualmente se introducirían en la consola del sistema operativo han sido programados por medio de funciones para su completa automatización. Una vez el usuario haya introducido los datos en el formato indicado, solo tiene que ejecutar el programa y seleccionar el modo escritura, que es guardar los datos en la memoria DRAM. Una vez guardados tendrá dos opciones, leerlos de vuelta vía ethernet o enviarlos cíclicamente por el transmisor DAC.

En cuanto a la implementación Hardware, el problema principal ha sido crear un protocolo propio de transmisión de comandos para configurar todas las variables necesarias del protocolo TCP/IPv4. Este protocolo a su vez tenía que transportar los comandos necesarios para controlar tanto la escritura y la lectura de la memoria DRAM como el control de la transmisión de los datos vía DAC. Por último, en cuanto a las tasas, se ha tenido que ajustar el sistema para que desde la primera muestra que se enviara hasta la orden de parada de envío, no hubiera ningún desabastecimiento de datos desde la memoria DRAM hacia el conversor DAC.

6.2. Planificación y metodología del TFG

En esta sección vamos a explicar el proceso de aprendizaje, diseño y ejecución del trabajo final de grado.

6.2.1. Proceso de aprendizaje

El proceso de aprendizaje consta de introducir al alumno en la utilización del programa Vivado de la empresa Xilinx, que será la plataforma donde se realizará la implementación del diseño hardware. Una vez el alumno reprodujo ejemplos de pequeños programas diseñados e implementados en asignaturas relacionadas con la temática del trabajo, se le introdujo al uso de las IP's (intellectual properties) de la propia empresa. Esto son recursos de programas complejos que ya tiene implementado el propio compilador. En concretos se mandó al alumno a utilizar una FIFO (FIFO Generator en las IP's) tanto en un sistema de un único reloj como de dos relojes. En este mismo paso también se enseñó al alumno a la búsqueda utilización de la aplicación que contiene la documentación técnica de esta empresa (Videos y PDF's) como a su comprensión.

6.2.2. Diseño y ejecución del trabajo final de grado

El proceso de diseño consta en un primer paso a la utilización de un programa ejemplo donde se realizaba la escritura y lectura de un grupo de datos en la memoria. Con la utilización de otras IP's el alumno podía ver capturas en el tiempo de la ejecución de los programas hardware y ver su funcionamiento. Una vez el alumno tuvo un programa sencillo pero funcional de la controladora DRAM, se le explicó que tenía que conectar la DRAM por el puerto ethernet utilizando un programa IP ya desarrollada. La finalidad era poder conectar la memoria DRAM para su llenado de datos desde el programa Matlab. Una vez realizada esta conexión, se propuso hacer un pequeño protocolo implementado con comandos para poder automatizar totalmente la escritura y lectura de datos desde Matlab. Por último, se explicó al alumno que el trabajo iba a ser utilizado en otro proyecto y que el proyecto necesitaba, a parte de la salida de datos por el puerto ethernet, una salida adicional para conectar otro dispositivo con unas condiciones específicas de tratamiento de datos y que se tenía que poder controlar ambas salidas por comandos desde el programa implementado en Matlab.

6.3. Futuras ampliaciones

En este apartado vamos a comentar unas ideas sobre futuras ampliaciones. En realidad, teniendo un sistema Hardware fiable y sin fallos, las ampliaciones se centrarían en crear una interfaz gráfica para el usuario. Esta interfaz gráfica se puede crear con cualquier tipo de compilador, siempre y cuando se respeten los comandos y la estructura de la transmisión de estos mismos. Esta interfaz ayudaría a los futuros usuarios a utilizar el proyecto más

cómodamente que por una interfaz de comandos y funciones como está actualmente implementada. A continuación, vamos a enumerar una serie de ideas que podría tener esta interfaz:

- Selección del tipo de onda que se quiere transmitir. Por ejemplo, un seno, una rampa, una onda cuadrada, etc.
- Modificación del sistema para poder amplificar la señal hasta la amplitud que se quiere.
- Interfaz gráfica para trabajar en el dominio de las muestras, en el dominio frecuencial o en el dominio temporal.

6.4. Conclusiones personales

En este último apartado de la memoria del trabajo final de grado voy a realizar una valoración personal del transcurso de todo el proyecto de final de grado. En general el proyecto ha sido una experiencia positiva ya que se han puesto en práctica conocimientos adquiridos a través de los años en el Grado. Cabe destacar algunas dificultades que se han sufrido a lo largo del diseño del sistema, la implementación tanto Hardware cono Software y la redacción de memoria.

En primer lugar, tenemos el diseño del sistema donde la dificultad más notable encontrada ha sido la lectura e interpretación de los Datasheets relacionados con las diferentes partes del proyecto. En estos se encontraba toda la información necesaria para el desarrollo del diseño de la arquitectura Hardware. Pero para poder interpretar esta información, me ha hecho falta aprender a buscarla y saber interpretarla. Es un punto que tengo que mejorar y que me será de gran utilidad en el futuro.

En segundo lugar, encontramos la implementación donde el punto más complicado ha sido aprender a comprender la herramienta de diseño Vivado. Con la herramienta Matlab no hubo problemas ya que se desarrolla un buen aprendizaje a través de todo el Grado.

En último lugar, pero no menos importante, encontramos la confección y redacción de esta trabajo. Empujado por la complejidad de los dos puntos anteriores, me ha llevado a elaborar una memoria totalmente diferente a lo que estaba acostumbrado a hacer. El intentar redactar y describir una serie de bloques programados y que se entienda la funcionalidad sin perderte por el camino describiendo pequeños detalles ha sido un apartado bastante complejo.

Pero no todo son experiencias negativas, ya que me llevo el recuerdo de haber empezado un aprendizaje partiendo desde cero hasta tener un buen dominio tanto de las herramientas de diseño y programación como unos conocimientos adquiridos y puestos en práctica que de seguro en un futuro no muy lejano me servirán para desarrollar una carrera profesional como ingeniero de telecomunicaciones.

Anexos 7

Documentación técnica

7.1. Documentación FPGA Virtex y Kintex Serie 7

Toda la documentación relacionada con las FPGA's que se va a nombrar a continuación están todas dentro de la aplicación DocNav que tiene la propia empresa Xilinx.

- Documentación Xilinx FIFO's Generator: **PG 057**
- Documentacion Xilinx FPGA Virtex Serie 7: **DS 183**
- Documentacion Xilinx FPGA Kintex Serie 7: **DS 182**
- Documentacion Xilinx DRAM FPGA core MIG Serie 7: **UG 586**
- Documentación Xilinx FPGA Clocking Wizard Serie 7: **PG 065**

7.2. Documentación Conversor EURVIS 5 Gsps

Enlace a la página web de la empresa Euvis donde se encuentra la documentación del conversor DAC utilizado en este proyecto.

- `http://www.euvis.com/products/mod/fmc/fmc2657.html`

7.3. Documentación Matlab

- `http://www.mathworks.com`