Proyeto Fin de Carrera

Fast Data Transfer based on a USB3.0 Super Speed Device

Autor:
Jesús Benítez Lorente

Directores:
Dipl.-Ing. (FH) Markus Wohlschlager
Prof. Isidro Urriza Parroqué

Universidad de Zaragoza/EINA

2015

RESUMEN

El objetivo de este proyecto es comprobar si el chip FX3 del microcontrolador FX3 Super Speed Device puede realizar una transmisión de datos sin errores, para poder realizar el depurado de las señales del monitor AVALON[1]. Estos datos serán generados a una tasa constante de 16 Mbps.

Para comprobarlo se llevarán a cabo varios experimentos en los que se utilizará una FPGA como generador de señales, el microcontrolador FX3 Super Speed Device como interfaz entre la FPGA y el ordenador y un disco duro externo donde se guardarán los datos enviados desde la FPGA.

En el primero de ellos se implementará un contador de 16 bits en la FPGA y se enviarán los datos al ordenador a través de un USB 3.0 sin ningún tipo de control sobre el buffer del DMA lo que producirá una gran perdida de datos debido a que el buffer será sobreescrito antes de que dé tiempo a mandar los datos al bloque USB. Aunque con este primer experimento no se consigan los objetivos propuestos servirá de base al experimento número dos ya que se aprenderá a programar los diferentes módulos del FX3 Super Speed Device, se comprobará que todos los bloques funcionan correctamente y se desarrollarán los scripts de Matlab que servirán de punto de partida para el segundo experimento.

En el segundo experimento se enviará desde una FPGA al ordenador a través del FX3 Super Speed Device una señal en diente de sierra. Esta vez se inspeccionará el buffer del DMA antes de escribir en él, esto se hará utilizando el flag DMA0_Ready que indicará si el buffer está preparado para recibir información y el flag DMA0_Watermark que avisará con la antelación que quiera el usuario de que el buffer se va a llenar, esto permitirá enviar datos al ordenador sabiendo con certeza de que no se perderán datos en el camino debido al desbordamiento del buffer DMA.

Por último analizando los datos se llegará a la conclusión de que el chip permite llevar a cabo la transmisión con los requerimientos exigidos y se analizará la influencia de la tasa de transmisión sobre la tasa de error recogiendo muestras durante media hora a diferentes frecuencias de reloj.

[1] Monitor médico hacia el que se enfoca este proyecto fin de carrera.
http://www.healthcare.philips.com/main/products/patient_monitoring/products/fetalmaternal_monitors/index.wpd

Indice de contenidos

Lista de imágenes

1 INTRODUCCIÓN

En este primer capítulo se hablará de la compañía en la cual, se ha realizado el proyecto, Koninklijke Philips N.V., así como de los productos que se desarrollan en Boeblingen (Stuttgart, Alemania). Además se hablará de la motivación y objetivos de este proyecto y para terminar se explicará como se ha estructurado la memoria y se hará un pequeño resumen de cada capítulo.

1.1 Koninklijke Philips N.V.

Koninklijke Philips N.V., más conocida como Philips es una empresa de electrónica fundada en el año 1981 en Eindhoven, Paises Bajos por Gerard Philips, su padre Benjamin y su hermano Anton.

En la actualidad Philips tiene su sede en Amsterdam, Paises Bajos. Su director ejecutivo es Frans Van Houten. Da trabajo a 115.000 personas aproximadamente, y reportó unas ventas de 23.300 millones de euros en el año 2014, teniendo al final unos beneficios de 415 millones de euros.[2]

Además Philips es una empresa que cotiza en la bolsa de Nueva York (NYSE:PHG) y en la Euronexst de Amsterdam (AEX:PHI).[3] Su cotización actual en bolsa es de 25.02 euros por acción(06.05.2015, 9:28 horas).[4]

Sus principales líneas de negocio son:

- Electrónica para el hogar, se trata de una división que da trabajo a más de 16.000 personas y desarrolla productos como maquinillas de afeitar o planchas.
- Alumbrado, es la división más grande de Philips con unos 50.000 empleados y desarrolla productos como bombillas o LEDs.
- Sistemas médicos, se trata de una de las divisiones más rentables y da trabajo a más de 35.000 personas. Desarrolla productos como monitores médicos o aparatos de resonancia magnética.
  Será en esta división donde se desarrollará este proyecto fin de carrera.

Para los próximos años se espera una reestructuración de la empresa, en la cual se fusionarán las divisiones de electrónica para el hogar y sistemas médicos pasando a llamarse "HealthTech" y el negocio de iluminación seguirá siendo independiente bajo el nombre de "Lighting Solutions".[5]

[2] http://es.investing.com/equities/philips-kon-income-statement
[3] http://www.philips.es/about/company/companyprofile.page
[4] http://www.invertia.com/mercados/bolsa/empresas/philips/portada-rv024phillip
[5] http://www.newscenter.philips.com/main/standard/news/press/2014/20140923-philips-to-sharpen-strategic-focus-by-establishing-two-market-leading-companies-in-lighting-solutions-and-in-healthtech.wpd

1.2 Philips Healthcare

Philips Healthcare[6] es la división de productos médicos de Philips, tiene sedes a lo largo de todo el mundo en más de 100 paises, ocupando a más de 37.000 empleados, con unos ingresos anuales de 9.6 billones de euros.[7]

Desarrolla productos destinados a la prevención de enfermedades como aparatos de mamografía, productos para el diagnóstico de enfermedades como sistemas de resonancia magnética o tomagrafía computerizada, así como máquinas para el tratamiento de enfermedades como por ejemplo equipos de radiación oncológica y sistemas destinados a la monitorización de pacientes entre los que cabe destacar monitores para el seguimiento de las constantes vitales o aparatos de ultrasonidos para la monitorización de los fetos.

Este proyecto fin de carrera se realizará en esta división, concretamente en la filial situada en Boeblingen, Stuttgart, que se dedica al desarrollo y construcción de sistemas médicos para la monitorización tanto de los fetos como de la madre basados en ultrasonidos.[8]

1.3 Philips Medizin Systeme Boeblingen GmbH

Es una filial de la división Philips Healtcare con sede en Boeblingen, localidad situada a 30 km de Stuttgart, Alemania, y da trabajo a unas 800 personas.

En la fábrica se desarrollan y construyen aparatos médicos, en concreto, monitores. En el departamento OB and Special Measurements se desarrollan los dispositivos para la monitorización de mujeres en estado de gestación y fetos. Todo esto se hará con el monitor AVALON[9], el cual, a través de pulsos de ultrasonidos emitidos por tranductores situados en el vientre materno es capaz de medir el ritmo cardiaco de hasta dos fetos al mismo tiempo (gemelos/mellizos), así como el pulso materno y la presión intrauterina entre otros. Estos parámetros serán visibles en un display táctil, teniendo cada uno un color diferente para poder diferenciarlos con claridad, así por ejemplo el color naranja se utilizará para el pulso de los fetos, el verde para mostrar la presión intrauterina y el azul claro mostrará el pulso marterno, también existe la posibilidad de imprimir los resultados en papel a modo de cardiograma.

Existen tambien versiones sin cables, como por ejemplo el AVALON CL,[10] Este monitor tienen la gran ventaja de permitir a la mujer una gran movilidad, reduciendo el estrés preparto, mientras se toman constantes vitales tanto del feto/s como suyas propias (pulso y contracciones). Además puede ser sumergido durante al menos 5 horas, lo que facilita la vida al paciente que recibe una monitorización continuada. Este tipo de monitorización se utiliza para mujeres que han tenido complicaciones durante el embarazo o que han recibido la epidural, mientras esperan la hora del alumbramiento en el hospital.

[6] http://www.philips.es/healthcare
[7] http://www.philips.es/healthcare-about/philips
[8] http://www.philips.es/healthcare-solutions/mother-and-child-care/fetal-maternal-monitoring
[9] http://www.healthcare.philips.com/main/products/patient_monitoring/products/fetalmaternal_monitors/index.wpd
[10] http://www.healthcare.philips.com/main/products/patient_monitoring/products/Avalon_CL/

Imagen 1: Avalon FM30 con un transductor de ultrasonidos para fetos y un TOCO MP con un conector rojo para medir el pulso materno

1.4 Motivación y objetivos del proyecto

Electronics fetal monitors o Cardio-toco-Graphs se definen como instrumentos para la medida y visualización de más de un parámetro fisiológico, como por ejemplo, el ritmo cardiaco o la actividad intrauterina.

Los párametros vitales tanto del feto como de la madre son recogidos por transductores que se situarán en el vientre materno, estos parámetros serán procesados por los transductores y enviados al monitor que está equipado con un display táctil y una impresora térmica para la representación de los mismos. Los datos serán enviados utilizando un protocolo de comunicación CAN a través de un bus. Este CAN bus trabaja con una velocidad de transmisión de 500KBit/s, lo que permite la transmisión de dos canales de 16 bits a una frecuencia de muestreo de 3 kHz por sensor.

Las nuevas mejoras planeadas para el futuro requieren un incremento drástico en la capacidad de transmisión de datos, siendo el mínimo requerido 1 mega sample con una resolución de 16 bits por segundo, y eso está muy lejos de la capacidad que proporciona el CAN bus.

Por tanto el objetivo de este proyecto es investigar si el chip FX3, desarrollado por "Cypress Semiconductor" se adapta a los requerimientos establecidos. Estos requirimientos consistirán en una comunicación continuada y sin errores a 16 Mbits/s durante varios minutos.

1.5 Esquema de la memoria

En un primer momento se explicarán las diferentes tecnologías utilizadas para la comunicación entre el ordenador y el FX3 Super Speed Device, luego se pasará a presentar tanto el software como el hardware utilizados, y una vez introducidos todos los elementos empleados se dará una descripción detallada de los experimentos realizados para terminar

sacando una serie de conclusiones.

A continuación se procederá a realizar un breve resumen de cada uno de los capítulos:

En el capítulo USB se hablará de la tecnología de comunicación utilizada en este proyecto, USB proporcionando una defición, y hablando tanto de su historia como de sus características.

En el capítulo ELEMENTOS DEL PROYECTO se presentará el hardware utilizado para llevar a cabo este proyecto, estos elementos serán:

- FPGA.
- FX3 Super Speed Device.
- cable USB 3.0.
- Tarjeta PCI USB 3.0.

En el capítulo PROGRAMAS UTILIZADOS se presentarán los diferentes programas empleados tanto para la programación como para el análisis y la captura de los datos. Estos programas son:

- Quartus II web Edition.
- ModelSim-Altera 10.1e.
- EZ USB Suite(eclipse) IDE.
- GPIF II designer.
- USB Control Center.
- Visual Studio 2013.
- Clear Terminal.
- MATLAB.
- CollectData.exe.

El capítulo EXPERIMENTOS es el más importante de todos, ya que es en el cual se explica lo que se ha llevado a cabo. En el primer experimento se implementará un contador de 16 bits en la FPGA y se enviarán los datos al ordenador de forma continuada sin ningún tipo de control. En el segundo experimento se implementará en la FPGA una señal de diente de sierra, esta señal se enviará al ordenador, pero esta vez si que se controlarán los buffers a través de una serie de flags para evitar el desbordamiento de estos con su correspondiente pérdida de datos, además se analizará la influencia de la tasa de transmisión sobre la tasa de error.

En el capítulo CONCLUSIONES se sacarán las conclusiones oportunas a la vista de los datos proporcionados por las diferentes simulaciones que se han llevado a cabo.

2 USB (Universal Serial Bus)

A continuación se presenta la tecnología USB, la cual definiremos además de hablar de su historia así como de sus características técnicas poniendo especial énfasis en su último estándar en el mercado, el USB 3.0.

2.1 Definición

El USB (Universal Serial Bus) es un bus estándar industrial que define los cables , conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica a ordenadores, periféricos y dispositivos electrónicos. Ha llegado a convertirse en el estándar de conexión de periféricos como teclados, impresoras o móviles. [11]

2.2 Historia

En el año 1996 un consorcio de empresas formado entre otras por Microsoft, IBM, Intel o Apple, presentaron la primera especificación del USB, se trataba del USB 1.0. Esta especificación no tenía una gran velocidad, pero esto al principio no era importante ya que se utilizaba con dispositivos HID(Human Interface Device), como el teclado o el ratón.

En 1998 apareció otro nuevo estándar el USB 1.1, que permitía por ejemplo hacer una copia de una película de 4 Gbytes en tan sólo 45 minutos.

Los dispositivos se fueron desarrollando y el usuario demandaba cada vez más velocidad y capacidad de transmisión, por tanto apareció en el año 2004 un nuevo estándar, el USB 2.0, que permitía tener un gran ancho de banda y además era retrocompatible con el estándar USB 1.0. A dia de hoy el USB 2.0 sigue siendo el estándar más utilizado incluso por delante del nuevo estándar el USB 3.0.

En el año 2008 apareció el estándar USB 3.0 que ofrece entre sus muchas novedades una gran capacidad de transmisión y una mayor potencia de alimentación lo que hace que los dispositivos se puedan recargar mucho más rápido.

En el año 2014 fue presentado el nuevo estándar el USB 3.1, aunque todavía no ha salido al mercado, promete una velocidad de transmisión de 10 Gbps y un nuevo tipo de conector el tipo C, además presentará la ventaja de tener puertos reversibles.

[11] http://simson.net/clips/1999/99.Globe.05-20.USB_deserves_more_support+.shtml

2.3 Velocidad

Las velocidad que tienen los diferentes estándares del USB es:

- USB 1.0 $\Longrightarrow$ 1.5 Mbits/seg
- USB 1.1 $\Longrightarrow$ 12 Mbits/seg
- USB 2.0 $\Longrightarrow$ 480 Mbits/seg
- USB 3.0 $\Longrightarrow$ 5 Gbits/seg
- USB 3.1 $\Longrightarrow$ 10 Gbits/seg

2.4 Conectores

Existe tres tipos de conectores USB:

- Tipo A $\Longrightarrow$ Es un conector USB estándar. Se trata de un conector plano que por lo general se conecta al equipo.

Imagen 2: Conector USB tipo A macho y hembra

- Tipo B $\Longrightarrow$ Es un conector cuadrado en la parte inferior y ligeramente inclinado en la parte superior. Se utiliza para grandes dispositivos como por ejemplo impresoras o escáneres.

Imagen 3: Conector USB tipo B macho y hembra

- Tipo C $\Longrightarrow$ Aparecerá para el nuevo estándar el USB 3.1 y tendrá la gran ventaja de ser reversible.

Imagen 4: Conector USB tipo C macho

2.5 estándar USB 3.0

El estándar USB 3.0 ha sido desarrollado por un conglomerado de empresas entre las que destacan Intel, Microsoft o Texas Instrument, fue presentado en el año 2008 y destaca por las siguientes características:

- Permite la transmisión de datos a 5 Gbps, esto es 10 veces más rápido que el antiguo estándar el USB 2.0.
- Proporciona una corriente de hasta 900 mA, en comparación con los 500 mA del antiguo estándar, lo que hace que se pueda cargar los dispositivos mucho más rápido.
- Aunque aporta más energía no consume más, ya que utiliza un protocolo basado en interrupciones a diferencia de su antecesor que consultaba los dispositivos periodicamente.
- Tiene compatibilidad con los estándares USB 2.0 y 1.1.
- Permite el tráfico bidireccional.
- Soporta la transmisión de imágenes y video en HD.
- Se suele distinguir de los otros estándares porque lleva una pestaña azul y muestra el símbolo SS.
- Tiene la desventaja de que al llevar más filamentos, el cable es más grueso y rigido, como el cable Ethernet.

2.6 ¿Por qué USB 3.0?

El objetivo final de este proyecto es hacer una transmisión entre una FPGA y el ordenador con el requerimiento de que se transmitan los datos a una tasa de 16 Mbps, utilizando en este caso la tecnología USB.

Dentro de la tecnología USB hay varios estándares que se han presentado anteriormente, pero entre todos ellos se ha elegido el último estándar, el USB 3.0, ya que además de que el microcontrolador elegido para realizar la transferencia de datos entre la FPGA y el ordenador incluye este estándar, el USB 3.0 puede alcanzar una velocidad de 5 Gbps (625 MBps) en comparación con el estándar USB 2.0 que tiene una tasa de velocidad de 480 Mbps (60 MBps) y además el USB 3.0 incluye un nuevo protocolo basado en interrupciones que permite consumir un 50% menos de energía.

Por tanto se elegirá el estándar USB 3.0 ya que además de que su ancho de banda es mucho mayor y permite transmisiones a muchas más velocidad, los dispositivos que se alimentan a través de esta tecnología consumen mucha menos energía.

3 ELEMENTOS DEL PROYECTO (HARDWARE)

En este capítulo se introducirá al lector en el proyecto fin de carrera presentando el material utilizado durante su realización y su disposición durante los experimentos.

3.1 Esquema

Como se ha comentado anteriormente el objetivo del proyecto es investigar si el chip FX3, desarrollado por "Cypress Semiconductor" se adapta a los requerimientos establecidos para la comunicación entre los sensores y el monitor, para poder capturar esos datos y enviarlos al ordenador para su depuración. Para ello una señal será enviada desde una FPGA hasta el ordenador a través del FX3 Super Speed Device, que estará controlado por el usuario a través de una consola.

Para ello se dispondrá del siguiente esquema:

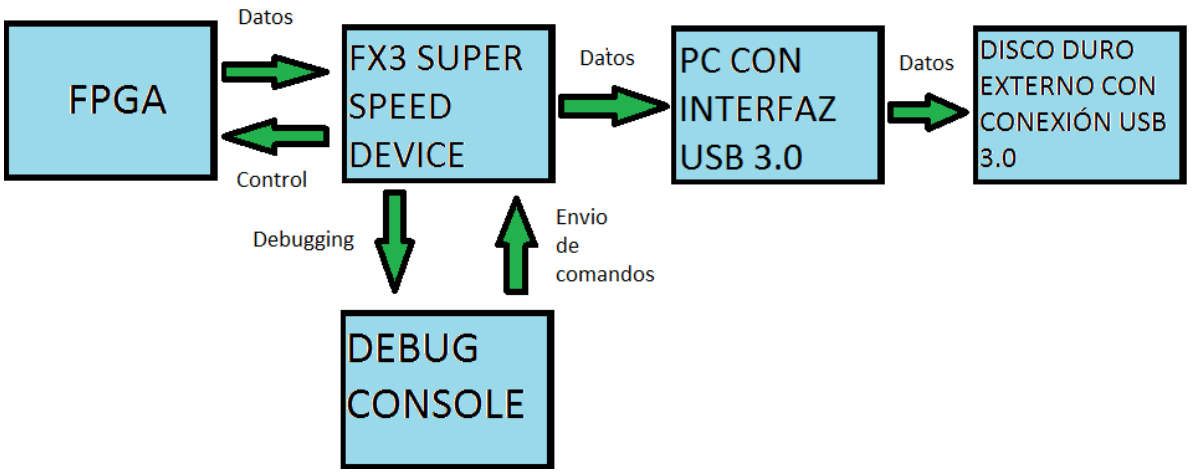

Imagen 5: Esquema que se utilizará durante los experimentos

En primer lugar se dispone de una FPGA(Field Programmable Gate Array), una FPGA es un dispositivo semiconductor que posee bloques lógicos interconectados para que puedan ser programados [12], se utilizará como un generador de datos, se creará una señal que tenga una frecuencia de 1 MHz, esta señal se enviará a través de los pines de propósito general(GPIO).

En segundo lugar se dispondrá del FX3 Super Speed Explorer Kit que tendrá el chip que se quiere investigar. Este dispositivo recibirá las señales de la FPGA de forma paralela a través de sus pines GPIO, que son pines de propósito general que pueden ser controlados por el usuario en tiempo de ejecución[13] y estos datos serán enviados al bloque USB 3.0 del que dispone la placa.

[12] http://www.alegsa.com.ar/Dic/fpga.php
[13] http://es.wikipedia.org/wiki/GPIO

El microcontrolador estará controlado por el usuario a través de una consola. La consola se comunicará con el microcontrolador a través del puerto serie UART, enviando comandos al procesador del FX3 Super Speed Device, con los cuales se podrá iniciar la transmisión o resetear la FPGA entre otros.

Por último estos datos serán recibidos en el ordenador a través de un interfaz que soporte la tecnología USB 3.0 y enviados a un disco duro externo con conexión USB 3.0 donde serán guardados en un archivo con extensión .bin. Luego estos serán tratados, analizados y representados con MATLAB para descartar la presencia de errores.

3.2 FPGA

Una FPGA (Field Programmable Gate Array), es un dispositivo semiconductor que posee bloques lógicos interconectados para que puedan ser programados.[14] La FPGA que se utilizará durante los experimentos es la 10M08SAE144C8GES fabricada por Altera y perteneciente a la famila MAX10.[15]

Esta FPGA se caracteriza porque:

- Se puede comprar por un precio unitario de 25,56 dolares, aunque se obtiene un descuento al comprar paquetes de 25, 100 o 500 unidades.
- Dispone de 8.000 elementos lógicos.
- Tiene 144 pines.
- de los cuales 101 se consideran pines de proposito general (GPIO).
- El voltaje oscila entre los 2.85 V y los 3.465 V.
- El rango de temperatura de trabajo varía entre los 0 y los 85 grados centígrados.
- Contiene 500 LABs(Logic Array Block)/CLBs(Configuration Logic. Block).

La FPGA se programará utilizando los lenguajes de programación Verilog y VHDL con ayuda del programa Quartus II proporcionado por el fabricante.

3.3 FX3 SuperSpeed Explorer Kit

El FX3 Super Speed Explorer Kit es una plataforma de desarrolo que permite controlar periféricos añadiendo la funcionalidad del USB 3.0 a cualquier sistema.
Para su correcto uso y entendimiento existe un libro llamado SuperSpeed Device by Example[16] escrito por John Hyde, antiguo ingeniero de Intel, que irá guiando al lector a través de todas las etapas del aprendizaje.

[14] http://www.alegsa.com.ar/Dic/fpga.php
[15] http://www.buyaltera.com/scripts/partsearch.dll?Detail&name=544-3037-ND
[16] http://www.cypress.com/?rID=99917

Los elementos más importantes de este microcontrolador serán:

1. El chip FX3 en el cual se basa todo este proyecto.
2. El puerto USB 3.0 a través del cual se mandarán datos a alta velocidad al ordenador.
3. Un puerto mini USB en el que se conectará un ordenador y servirá tanto para realizar el depurado del programa como para enviar comandos desde el ordenador.
4. El GPIO a través del cual se recibirán los datos de la FPGA y serán controlados por el bloque GPIF II.
5. Interruptor con el cual se podrá realizar un reboot del dispositivo.
6. Interruptor de uso general, que puede ser utilizado entre otras cosas para encender el LED o para iniciar una nueva transmisión.
7. LED, durante este proyecto será utilizado para indicar cuando hay una transmisión en curso.

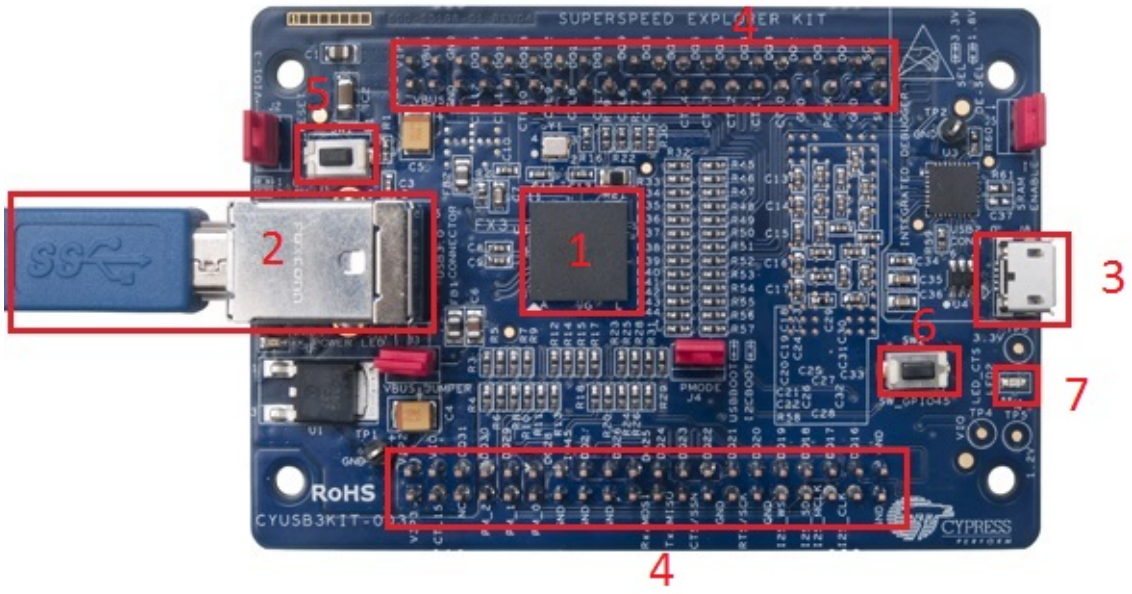

Imagen 6: Plataforma de desarrolo FX3 SuperSpeed Explorer Kit

3.3.1 Diagrama de bloques

El corazón del FX3 se puede considerar la "Distributed DMA Controller" que unirá a los diferentes bloques a través de tuberías (sockets) a una velocidad máxima de 800 MBps.

Los bloques más importantes de los que dispondrá el FX3 y que se explicarán en las siguientes secciones son:

- CPU block.
- GPIF II block.

- Low speed peripherals block.
- USB block.
- Distributed DMA Controller.

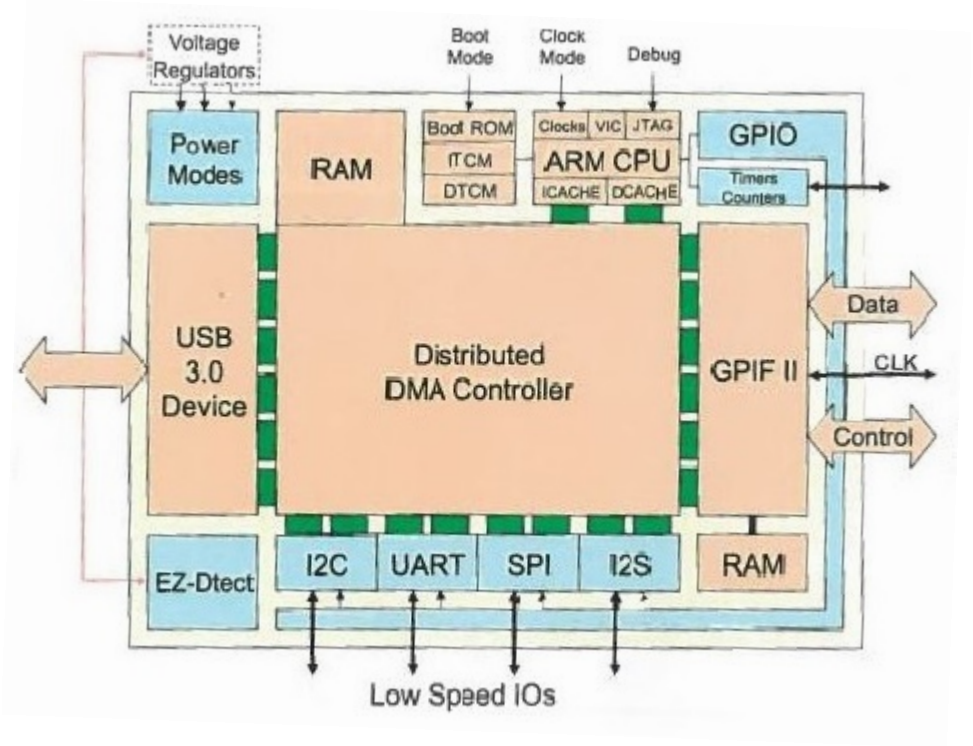

Imagen 7: Diagrama de bloques del FX3 SuperSpeed Explorer Kit

3.3.2 CPU block

Se trata del bloque que contiene el procesador,[17] dispondrá de un procesador ARM9 que trabajará a una frecuencia de 200 MHz y dispondrá de 2 memorias cache de 8 KB cada una, además la señal Clock la podrá recibir internamente a través de un cristal de cuarzo que vibrará a una frecuencia de 19,2 MHz o a través de un reloj externo. También dispone de un JTAG que servirá para la descargar y depuración de los programas y un controlador de interrupciones estándar PL192[18]. Unido a este bloque se tendrán 3 memorias. La primera será una memoria de 32 KB que dispondrá del código de arranque para el dispositivo. Este código se podra descargar en la memoria a través de un USB o a través de una memoria EEPROM conectada en serie. Para la realización del proyecto se cargará el programa a través de un cable USB 3.0. Además dispondrá de una memoria de 16 kB para instrucciones y otra de 8 kB para datos.

El procesador tendrá asignado un VID (Vendor ID) y un PID (Product ID) que serán reconocidos por el driver CyUSB3.sys, y servirán para que el ordenador pueda reconocer el dispositivo cuando se conecta.

[17] SuperSpeed Device by Example by John Hyde ISBN-10: 1500588059
[18] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0273a/DDI0273.pdf

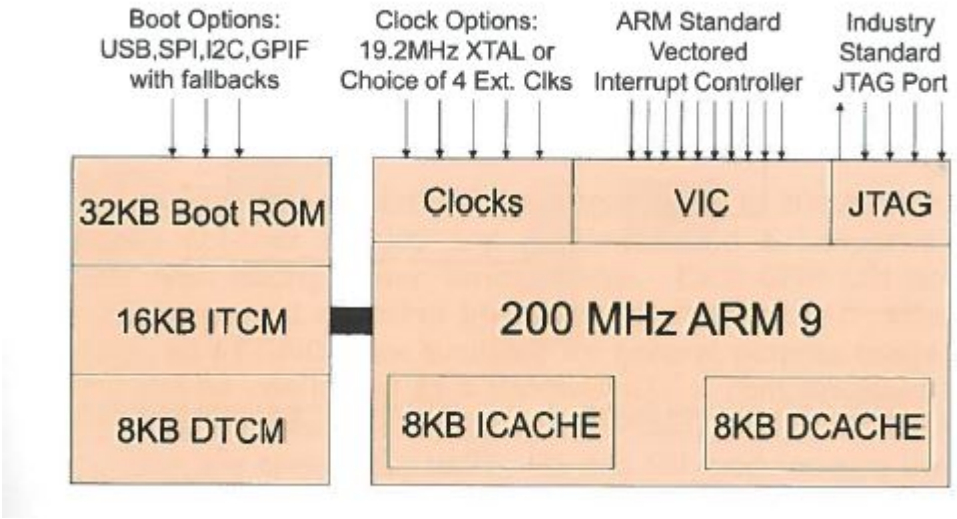

Imagen 8: Diagrama del CPU block

Este bloque será donde se cargará el firmware ya que en él se ejecutará la aplicación. Será el encargado entre otras cosas de configurar e iniciar el resto de los bloques y periféricos además de interpretar los comandos que el usuario mandará desde la consola que llegarán a través del periférico UART y de controlar la tubería que irá a través del DMA y conectará el bloques GPIF II con el bloque USB 3.0 Device.

3.3.3 GPIF II block

Este bloque dispone de una memoria RAM de 8 KB donde se cargará el programa diseñado con el software GPIF II Designer y que controlará el bloque. Este bloque consiste en una serie de vectores de elementos lógicos que tienen que ser programados. Su frecuencia máxima de funcionamiento es de 100 MHz que podrá recibir interna o externamente.

Dispone de 32 líneas de datos y 14 líneas de control además de 32 tuberías(sockets) para comunicarse con otros elementos de la plataforma de desarrollo, que podrán funcionar de una manera independiente unas de otras.

Para programar este bloque se utilizará el programa GPIF II Designer, proporcionado por Cypress. Este bloque usa la filosofía de programación de una FPGA y admite hasta 256 estados lógicos diferentes.

El bloque será uno de los más importantes del FX3 Super Speed Device durante el experimento, ya que será el encargado de la comunicación entre el microcontrolador y la FPGA, por tanto, habrá que prestarle especial atención.

El bloque GPIF (General Programmable Interface) tiene la siguiente forma:[19]

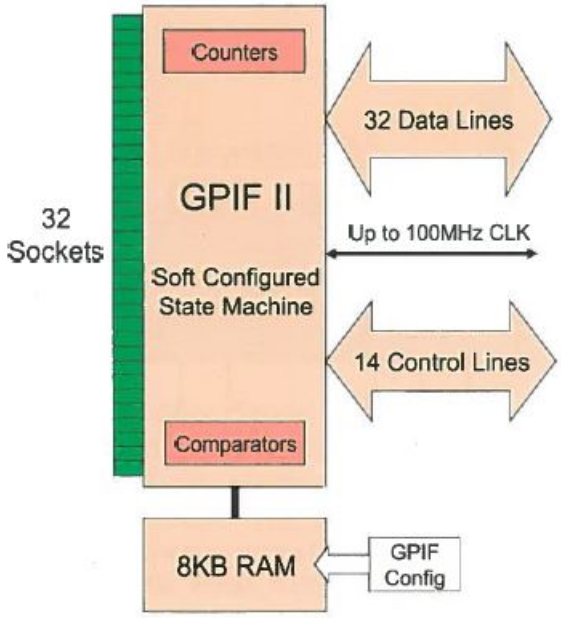

Imagen 9: Diagrama del GPIF II block

3.3.4 Low speed peripheral block

Este bloque contendrá los periféricos necesarios para las comunicaciones más lentas, servirá para conectar dispositivos como una EEPROM o para ser utilizado como "debug console" (consola de depuración), su diagrama será:[20]

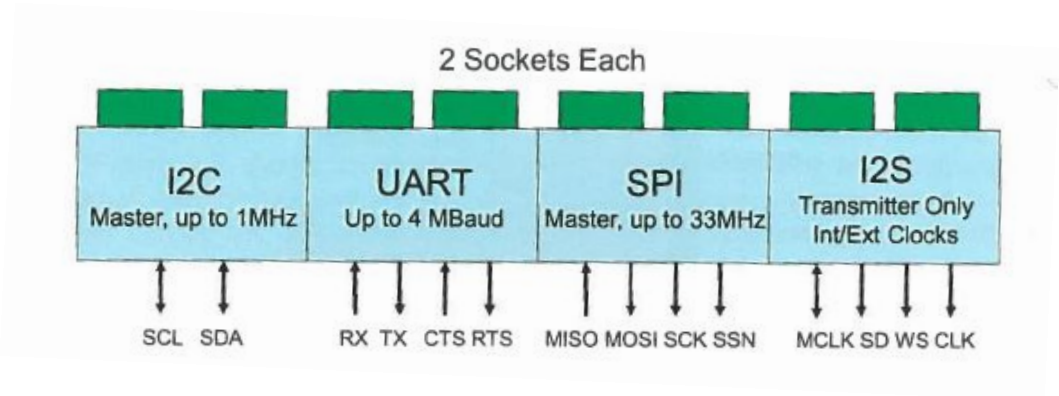

Imagen 10: Diagrama del Low speed peripheral block

De este bloque el periférico más importante será el puerto serie UART con el cual se establecerá la comunicación entre el procesador y la consola para el envío de comandos y mensajes de depuración.

[19] SuperSpeed Device by Example by John Hyde ISBN-10: 1500588059
[20] SuperSpeed Device by Example by John Hyde ISBN-10: 1500588059

3.3.5 Distributed DMA Controller

Este bloque se puede considerar como el corazón del dispositivo ya que a través de él discurrirán todas las tuberías(sockets) que unirán a los diferentes periféricos y que estos utilizarán para enviar información de un punto a otro. En este proyecto se utilizarán dos tuberías una que irá desde el GPIF II block pasando por el Distributed DMA Controller hasta el bloque USB y otra que comunicará el procesador con el periférico UART.

3.3.6 USB block

Este bloque se utilizará para el envío de datos a gran velocidad desde la placa de desarrollo hasta el ordenador, así como de fuente de alimentación.

El bloque implementa 32 endpoints diferentes a los que se podrán conectar cada una de las 32 tuberías disponibles, permitiendo la transmisión de datos de 32 conexiones diferentes al mismo tiempo.

También incluye un bloque denominado EZ-Dtect, que está controlado por el procesador y permite detectar la presencia de una carga a través del USB para alimentar a los periféricos.

Su esquema es:

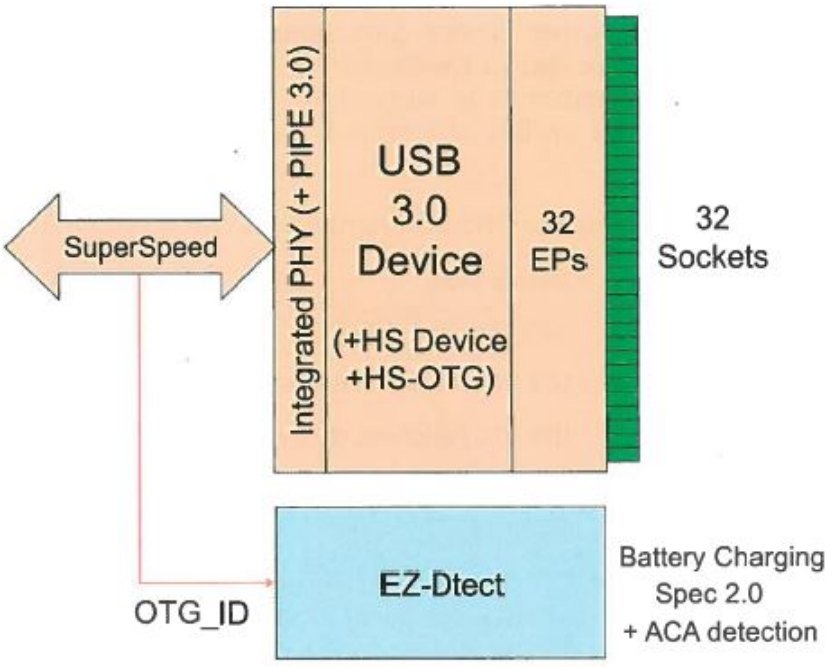

Imagen 11: Diagrama USB block

3.4 Cable USB 3.0

El USB que se conectará al ordenador será de tipo A y el que se conectara al dispositvo será de tipo B, se puede observar el color azul típico del USB 3.0. Estos además se caracterizarán por tener la siguiente forma y patillaje:

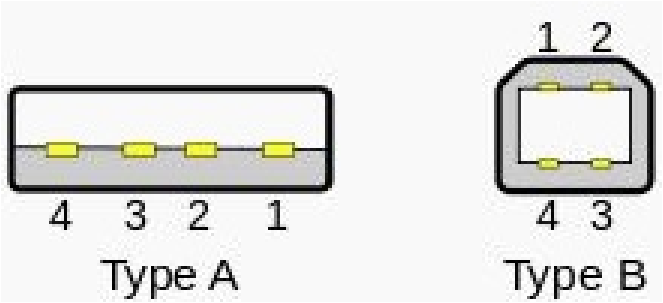

Imagen 12: Patillaje cable USB 3.0 de tipo A y B

Pin	Nombre	Color cable	Descripción
1	VCC	Rojo	+5V
2	D-	Blanco	Data-
3	D+	Verde	Data+
4	GND	Negro	Tierra

3.5 Tarjeta PCI USB 3.0

Se utilizará una tarjeta PCI USB 3.0 para recibir los datos enviados desde el FX3 Super Speed Device en el ordenador. La tarjeta utilizada será la tarjeta PCI de NEC-Chipsatz, la cual proporcionará 4 puertos externos USB 3.0.

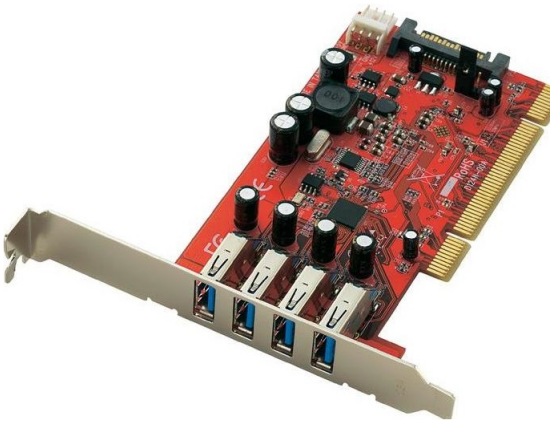

Imagen 13: Tarjeta PCI USB 3.0

Una vez explicado el Hardware utilizado se pasará a introducir los programas (software) empleados.

4 PROGRAMAS UTILIZADOS (SOFTWARE)

En esta sección se van a introducir los programas utilizados durante la realización de este proyecto.

Para programar la FPGA se utilizarán los siguiente programas proporcionados por Altera:[21]

- Quartus II web Edition $\Longrightarrow$ Para escribir el código de los programas en un lenguaje de descripción de hardware como por ejemplo VHDL o Verilog.
- ModelSim-Altera 10.1e $\Longrightarrow$ Para simular el programa escrito anteriormente con un lenguaje de descripción de hardware.

Para la programación y control de la placa de desarrollo FX3 Super Speed Device se utilizarán los siguientes programas:

- EZ USB Suite Files $\Longrightarrow$ Para escribir los programas.
- GPIF II designer $\Longrightarrow$ Para programar el bloque GPIF II.
- USB control center $\Longrightarrow$ Para cargar los programas en el dispositivo.

Estos programas son proporcionados por el fabricante de la placa, Cypress. [22]

Por último en el ordenador se utilizará:

- Visual Studio 2013 $\Longrightarrow$ Con el cual se escribirá un programa para recolectar los datos que van llegando.
- Clear Terminal $\Longrightarrow$ Para mandar los comandos al dispositivo FX3 Super Speed.
- CollectData.exe $\Longrightarrow$ Con el que se guardarán los datos que nos van llegando en un archivo .bin.
- MATLAB $\Longrightarrow$ Para analizar y representar los datos recibidos.

4.1 Quartus II web edition

Quartus II es un software de diseño de dispositivos lógicos programables producido por Altera. Permite el diseño y análisis de circuitos lógicos y diseños en HDL (Hardware Description Language). Quartus II incluye además una implementación de VHDL y Verilog para la descripción de hardware, así como la edición de circuitos lógicos y un simulador de formas de onda.

La versión web que será la utilizada es una versión gratuita proporcionada por Altera, con ella, se puede trabajar con algunas familias de dispositivos como la familia Cyclone o la familia MAX, cabe destacar la MAX 10[23] que se utilizará durante los experimentos.

[21] https://www.altera.com/
[22] http://www.cypress.com/
[23] https://www.altera.com/products/fpga/max-series/max-10/overview.html

4.2 ModelSim-Altera 10.1e

Programa que sirve para la simulación de lenguajes de descripción de hardware como por ejemplo VHDL o Verilog. Se puede utilizar de forma independiente o junto con el programa Quartus II web Edition.

4.3 EZ USB Suite(eclipse) IDE

EZ USB Suite es un programa con el cual se pueden importar proyectos ya realizados, editarlos y compilarlos, así como realizar proyectos propios. El Software trabaja bajo un entorno de tiempo de ejecución de Java (JRE), el cual permite la ejecución de programas en Java.

4.4 GPIF II designer

Se trata de un programa gráfico que permite la configuración de la interfaz GPIF II del controlador EZ-USB FX3 USB 3.0.
El programa permite la creación de diferentes estados lógicos, la configuración de los saltos entre estados así como la asignación de los pines del dispositivo. Además permite la simulación del diagrama de estados diseñado especificando las diferentes transiciones y tiempos.

4.5 USB Control Center

También es un programa administrado por Cypress que sirve entre otras cosas para:

- Cargar el programa ya compilado en la memoria RAM del dispositivo FX3 Super Speed Explorer Kit.
- Ver la configuración del dispositivo, entre otras cosas se podrá saber los puntos de acceso del USB o el tipo de comunicación (Stream, bulk...).
- Transferencia y recepción de datos.

4.6 Visual Studio 2013

Es un entorno de programación para sistemas operativos de Windows. Permite a los desarroladores la creación de aplicaciones y sitios web. Soporta entre otros lenguajes C++, C#, Visual Basic .NET, Java, Phyton, PHP...
Será utilizado para la modificación de ejecutables .exe que son proporcionados por Cypress.

4.7 Clear Terminal

Se trata de una aplicación gratuita desarrollada por ClearConnex[24] que corre sobre Windows y permite entre otras cosas una comunicación serie entre dispositivos(UART).
Será utilizado para la recepción de mensajes de depuración y para el envío de comandos al dispositivo.

4.8 CollectData.exe

Ejecutable proporcionado por Cypress con el que se irán guardando los datos que van llegando al ordenador en un archivo .bin.

4.9 MATLAB

Matlab (MATrix LABoratory) es una herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio.
Se caracteriza por la manipulación de matrices, la representación y el análisis de datos, la implementación de algoritmos y la comunicación de programas en otro lenguaje de programación.[25]
Será utilizado para la manipulación y el análisis de datos que serán recibidos en el ordenador a través del USB 3.0.

[24] http://www.clearconnex.com/
[25] http://de.wikipedia.org/wiki/Matlab

5 EXPERIMENTOS

En esta sección se explicarán los dos experimentos que se han llevado a cabo. El primero será un contador de 16 bits en el cual se enviarán los datos desde la FPGA hasta el ordenador a través del USB 3.0 y servirá para verificar que todos los elementos funcionan correctamente y para analizar las dificultades que se nos pueden plantear durante la transmisión. En el segundo experimento se enviará una señal en diente de sierra. Para que no halla perdida de datos se llevará a cabo control de flujo a través de dos flags de control que serán DMA0_Ready y DMA0_Watermark.

5.1 Experimento 1

Este primer experimento se usará para empezar a entender las diferentes partes de la programación tanto del dispositivo FX3 Super Speed Device como de la FPGA, así como para comprobar que todos los bloques del microcontrolador funcionan correctamente y ver los posibles problemas con los que nos podremos encontar durante la transmisión. Además se desarrollarán los scripts de MATLAB para el análisis de datos que servirán de base para el próximo experimento.

El experimento consistirá en un contador de 16 bits que se implementará en la FPGA. En este experimento la FPGA ejercerá el rol de master y el FX3 Super Speed Device el de slave, por tanto, además de propocionar los datos, la FPGA enviará una señal denominada WR que le indicará al dispositivo cuando estarán los datos preparados para su lectura en los pines GPIO.

El bloque GPIF II (slave) del microcontrolador recibirá los datos y los enviará al bloque USB 3.0 a través de una tubería que irá por el bloque "Distributed DMA Controller", además también se encargará de enviarle a la FPGA una señal denominada Reset para indicarle que tiene que reiniciar el contador, mientras esta señal este activa (Reset=1) el contador no funcionará. Esta señal estará controlada por el usuario y para cambiar su valor actual será necesario enviar el comando Reset a través de la consola al FX3 Super Speed Device.

El bloque USB 3.0 enviará los datos al ordenador, donde tras poner en funcionamiento el ejecutable CollectData.exe proporcionado por Cypress los datos serán recibidos en la tarjeta PCI USB 3.0 y guardados en un archivo .bin en un disco duro externo con conexión USB 3.0.

Este archivo .bin donde se encuentran los datos se analizará con MATLAB, que proporcionará el número de datos leídos, el número de errores y la proporción de estos con respecto al número de datos leídos.

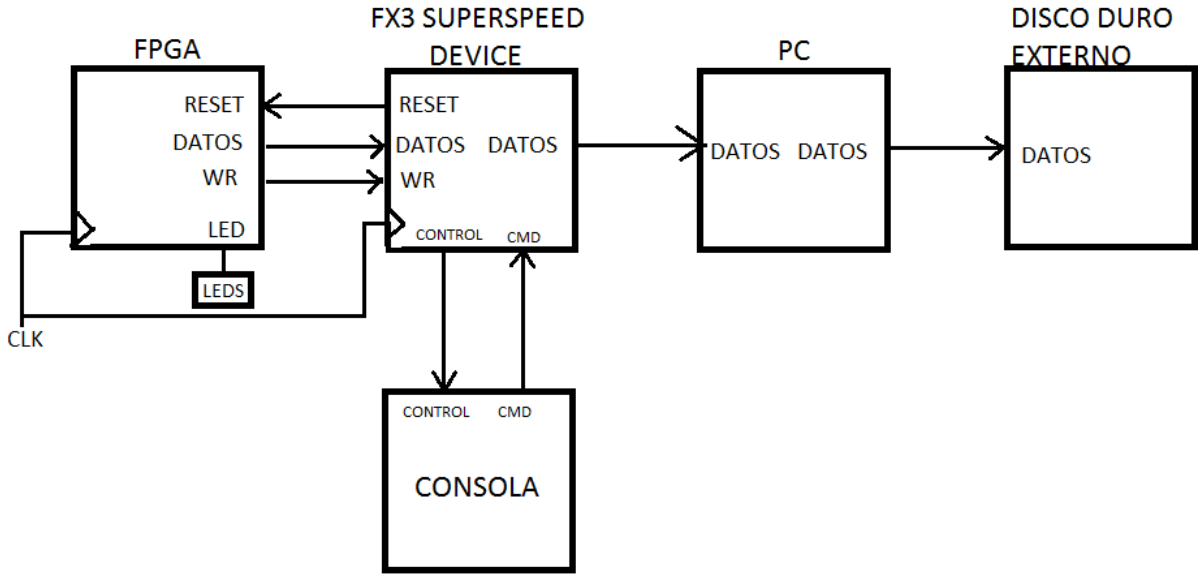

Imagen 14: Configuración del primer experimento

Durante el experimento se programarán las siguientes partes:

- La FPGA para lo cual se utilizará:
  - Quartus II web Edition para escribir el código.
  - ModelSim-Altera 10.1e para simular el programa.
- El FX3 Super Speed Device para lo cual se utilizará:
  - GPIF II designer para programar el bloque GPIF II.
  - EZ USB Suite(eclipse) para escribir los programas que controlarán el chip FX3.
  - USB Control Center para cargar los programas en la RAM del dispositivo.
- MATLAB para el análisis de datos.

5.1.1 FPGA

Lo primero que se hará es el diseño del contador de 16 bits que correrá en la FPGA diferenciando las señales de entrada y salida que pertencerán al bloque de control y al contador.

Se tendrán dos entradas, la pertenciente al reloj que hará que la FPGA funcione, esta fuente de reloj será proporcionada por el FX3 Super Speed Device y una señal Reset que indicará cuando se cuenta o cuando se tienen que reiniciar las variables.
Se dispondrá además de 3 salidas, WR con la cual indicaré al microcontrolador cuando se pueden leer los datos que se envían desde la FPGA, LED que servirá para que el usuario sepa en todo momento en que estado del programa se encuentra la FPGA dependiendo de los LEDs que se enciendan y DATOS[0:15] que contendrá al contador.

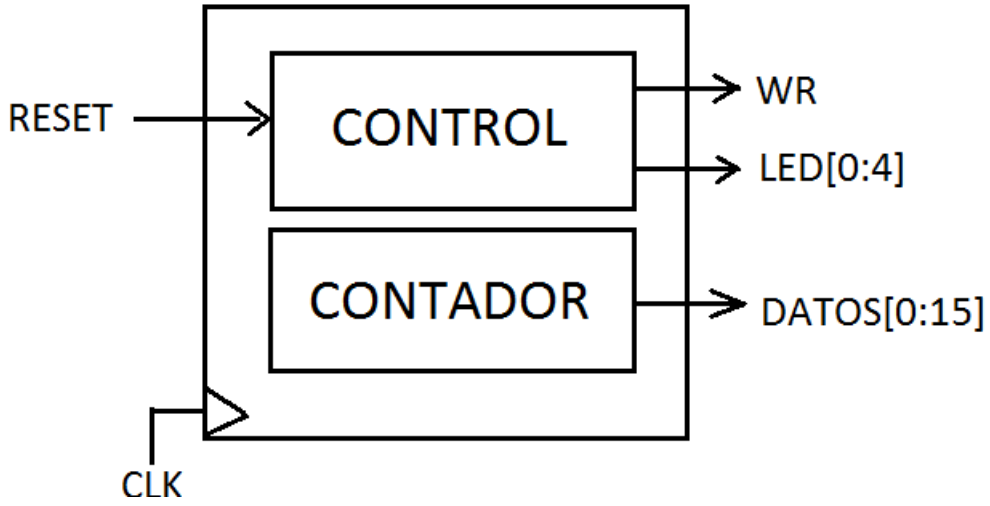

Imagen 15: Diagrama de bloques de la FPGA en el experimento 1

El bloque de control tendrá la siguiente máquina de estados que establecerá como va a funcionar el programa:

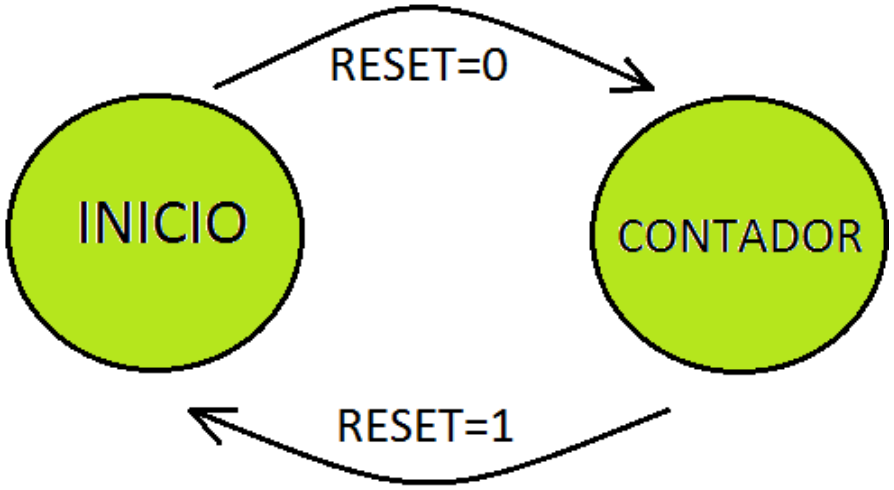

Imagen 16: Máquina de estados del experimento 1

En un primer momento se estará en el estado de INICIO, en este estado el contador será reseteado y a la FPGA se estarán enviando ceros que servirán para saturar los buffers del ordenador por tanto la señal WR_n=0 y LED[0:4]="11111" por lo que todos los leds se encenderán.

Cuando el usuario mande el comando Reset a través de la consola se pasará al estado CONTADOR, en este estado la FPGA estará contando y enviando esos datos al micro-controlador por tanto la señal WR_n=0 y LED[0:4]="10000" por lo que sólo se encenderá el primer led que indicará al usuario que hay una transmisión el la cual se está enviando el contador.

Una vez que se ha diseñado el contador habrá que programarlo, para ello se utilizará el programa Quartus II web Edition. El lenguaje de programación de hardware elegido para este experimento es Verilog.

Una vez escrito el programa, este deberá de ser compilado para comprobar que no se han cometido errores en la sintaxis. Si la compilación no reporta ningún error el siguiente paso en el proceso será el de simular el programa, ya que aunque no se halla cometido ningun error de sintaxis es posible que el programa no haga exactamente lo que se busca. Para simular el programa se utilizará el programa ModelSim y este es el resultado:

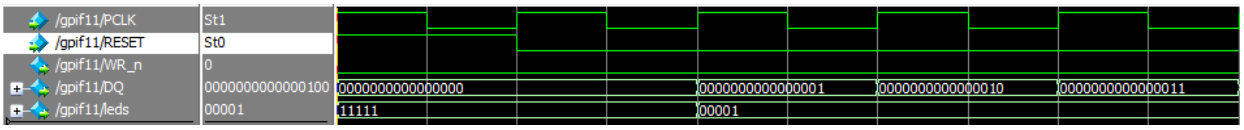

Imagen 17: Simulación de la FPGA en el experimento 1

Como se puede observar en un primer momento la señal reset esta activa por tanto, se estará en el estado inicio y sólo se mandaran ceros por lo tanto WR_n=0 y todos los leds estarán encendidos, cuando se reciba la señal reset=0 se pasará al estado contador por lo que la FPGA activará el contador,WR_n=0 y sólo se encederá el primer led.

Escrito y simulado el código el siguiente paso es realizar la asignación de pines de la FPGA, en la cual se tendrán 16 pines que se asignarán a la salida del contador, 1 pin para la señal de reloj y 7 pines para el control del programa que corresponderán a la señal WR_n, al Reset y a los leds. Esta asignación se hará con la tool "Pin-Planner".

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
DQ[15]	Output	PIN_52	3	B3_N0	PIN_52	2.5 V (default)
DQ[14]	Output	PIN_48	3	B3_N0	PIN_48	2.5 V (default)
DQ[13]	Output	PIN_47	3	B3_N0	PIN_47	2.5 V (default)
DQ[12]	Output	PIN_46	3	B3_N0	PIN_46	2.5 V (default)
DQ[11]	Output	PIN_44	3	B3_N0	PIN_44	2.5 V (default)
DQ[10]	Output	PIN_43	3	B3_N0	PIN_43	2.5 V (default)
DQ[9]	Output	PIN_41	3	B3_N0	PIN_41	2.5 V (default)
DQ[8]	Output	PIN_39	3	B3_N0	PIN_39	2.5 V (default)
DQ[7]	Output	PIN_70	4	B4_N0	PIN_70	2.5 V (default)
DQ[6]	Output	PIN_69	4	B4_N0	PIN_69	2.5 V (default)
DQ[5]	Output	PIN_65	4	B4_N0	PIN_65	2.5 V (default)
DQ[4]	Output	PIN_62	4	B4_N0	PIN_62	2.5 V (default)
DQ[3]	Output	PIN_61	4	B4_N0	PIN_61	2.5 V (default)
DQ[2]	Output	PIN_60	3	B3_N0	PIN_60	2.5 V (default)
DQ[1]	Output	PIN_57	3	B3_N0	PIN_57	2.5 V (default)
DQ[0]	Output	PIN_55	3	B3_N0	PIN_55	2.5 V (default)
PCLK	Input	PIN_38	3	B3_N0	PIN_38	2.5 V (default)
RESET	Input	PIN_66	4	B4_N0	PIN_66	2.5 V (default)
WR_n	Output	PIN_64	4	B4_N0	PIN_64	2.5 V (default)
leds[4]	Output	PIN_14	1A	B1_N0	PIN_14	2.5 V (default)
leds[3]	Output	PIN_13	1A	B1_N0	PIN_13	2.5 V (default)
leds[2]	Output	PIN_12	1A	B1_N0	PIN_12	2.5 V (default)
leds[1]	Output	PIN_11	1A	B1_N0	PIN_11	2.5 V (default)
leds[0]	Output	PIN_10	1A	B1_N0	PIN_10	2.5 V (default)

Imagen 18: Asignación de pines en la FPGA

Por último antes de cargar el programa en la FPGA, este se volverá a compilar y se creará un archivo .sof que es el que se tendrá que cargar en la FPGA a través del USB-Blaster.

5.1.2 FX3 Super Speed device

A la hora de programar el dispositivo se tendrá que configurar y controlar todos los bloques y periféricos de los cuales dispone este microcontrolador, para ello, el fabricante Cypress proporciona un RTOS (Real Time Operating System), un RTOS es Sistema manejador de recursos que permite una distribución controlada y ordenada del procesador, memoria, E/S, entre los diversos programas que compiten por ellos.[26] El RTOS que se utilizará durante este proyecto fin de carrera es el Express Logic's ThreadX[27] (version 5.1).

A continuación se presenta una vista de los diferentes partes a programar en la plataforma de la familia FX3.

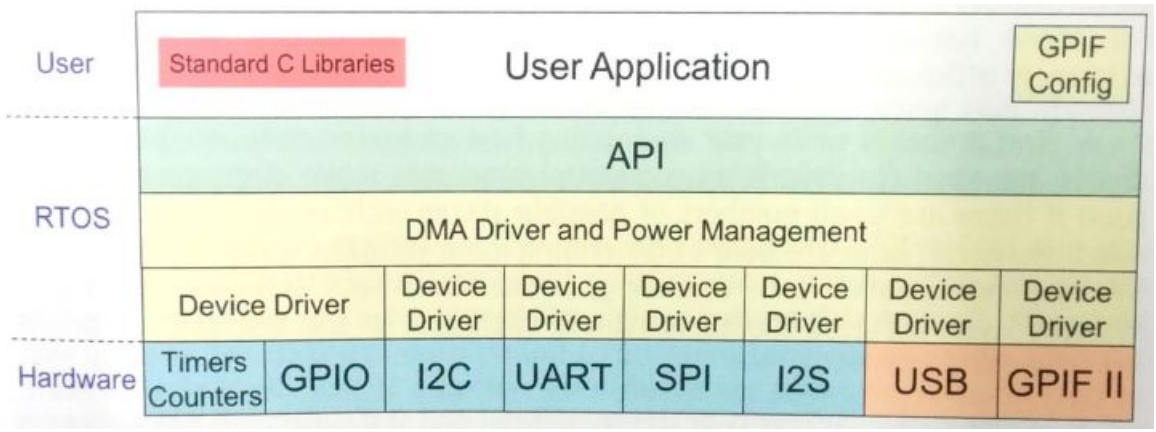

Imagen 19: Vista de la programación de los dispositivos de la familia FX3

Como se puede observar se tendrán tres niveles diferentes, el primero es el nivel de usuario, en el cual se podrán utilizar las librerías estándar de C, en este nivel se programará el GPIF y las diferentes funciones que el usuario quiera para su programa. Luego en el segundo nivel se encontrará el sistema operativo de tiempo real (RTOS), en el cual a través de las funciones API proporcionadas por Cypress se activarán los drivers que servirán para controlar los bloques y periféricos situados en el bloque tres, el correspondiente al Hardware.

En primer lugar se procederá a la programación del bloque GPIF II, este bloque será el que se encargue de la comunicación entre el microcontrolador y la FPGA para ello se utilizará el programa GPIF II Designer, que proporcionará una interfaz gráfica para la programación de este bloque.

[26] http://www.ing.unlp.edu.ar/electrotecnia/procesos/transparencia/SOTR_1.pdf
[27] http://rtos.com/products/threadx/

Se configurará la relación del GPIF con la FPGA.

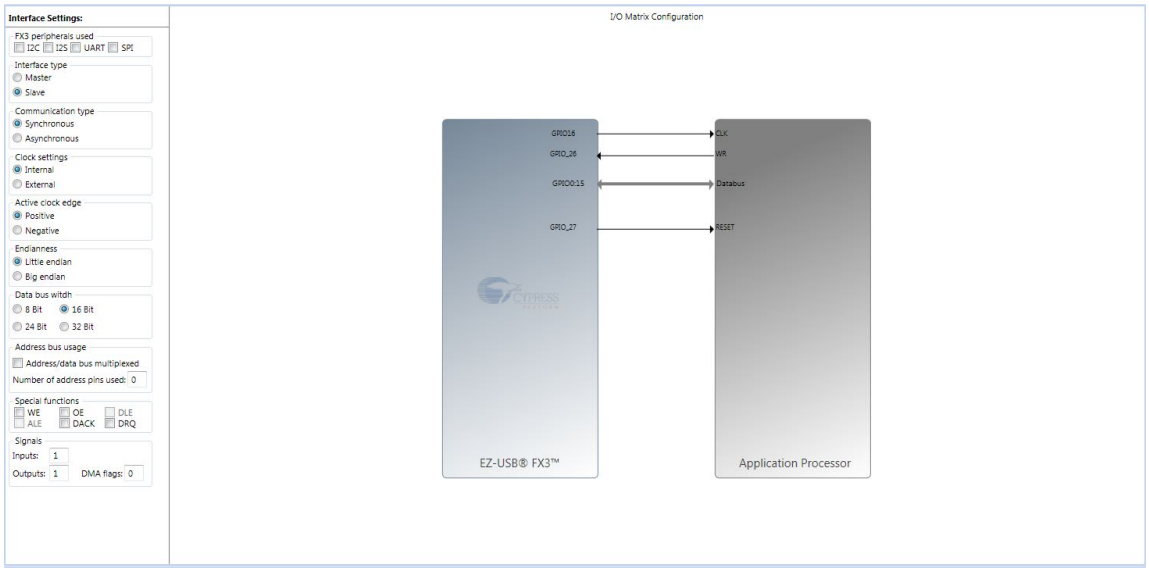

Imagen 20: Interacción del bloque GPIF II con el exterior

Se tendrá que seleccionar el tipo de rol que el dispositivo tomará durante la comunicación con el exterior, en este caso, el FX3 Super Speed Device será el slave, tendrá una comunicación síncrona, la fuente del reloj será interna, se utilizará un flanco de reloj ascendente y el número de bits de datos que serán enviados será 16. Además también habrá que seleccionar el número de entradas y salidas de control, en este caso serán tres. Dos salidas pertencientes a las señales CLK (reloj) y RESET y una entrada WR. Por último se tendrán que seleccionar los pines del GPIO a los cuales corresponderá cada señal.

señal	Pin GPIO	Descripción
CLK	GPIO 16	Se corresponde con la señal clock que alimentará a la FPGA
WR	GPIO 26	Señal que le dirá al GPIF cuando puede leer
RESET	GPIO 27	Señal que dirá a la FPGA cuando reiniciarse
Datos	GPIO 0:15	Bits pertenecientes a los datos

Una vez configurado habrá que programar su relación con el exterior, esto se realizará a través de un diagrama de estados.

Una vez que se inicie el programa se pasará del primer estado START al estado WAIT de forma automática. En el estado WAIT se esperará hasta que llegué la señal WR=0, en este momento se pasará al estado SAVE donde se leerán los datos que vayan llegando desde la FPGA, esto es posible gracias a la operación IN_DATA que permite la lectura de datos de los pines GPIO. Se permanecerá en el estado SAVE mientras que la señal WR=0 cuando se reciba WR=1 se volverá al estado WAIT y se seguirá en este bucle mientras el usuario lo desee.

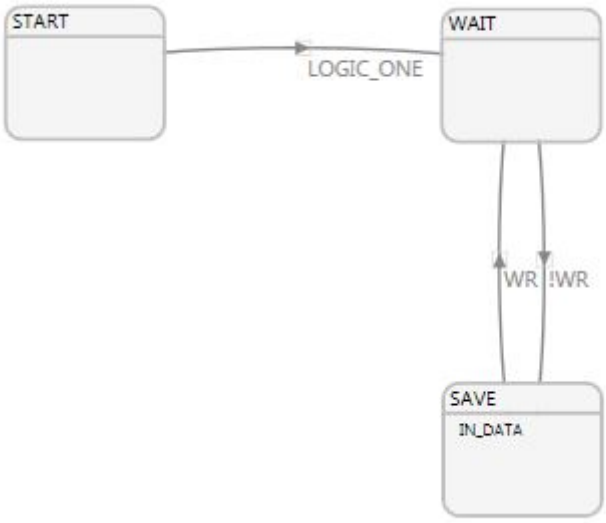

Imagen 21: Diagrama de estados GPIF experimento 1

Una vez que se halla programado todo habrá que compilar el programa para verificar que no hay errores. Si no ha habido errores se creará un archivo .h(SupplyGpifData.h) que se compilará más adelante con el resto de archivos del firmware.

Una vez programado el GPIF II se pasará a configurar y programar todos los periféricos del dispositivo así como los diferentes bloques que aún quedan. Para esto se utilizarán tanto las librerías estándar de C como las funciones API y los drivers proporcionados Cypress, así como el programa EZ USB Suite(eclipse), el cual tiene la siguiente apariencia:

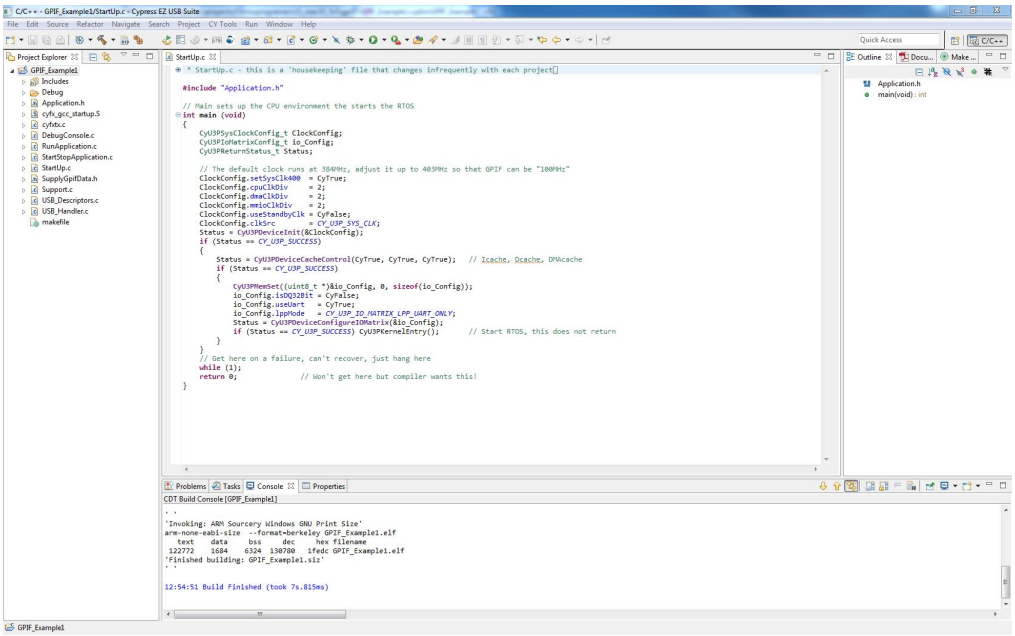

Imagen 22: Apariencia del programa EZ USB Suite(eclipse)

Los scripts que se tendrán que programar serán:

- Startup.c $\Longrightarrow$ Donde se inicializarán los parámetros de la CPU y la matriz de entrada y salidas.
- RunApplication.c $\Longrightarrow$ En el cual se mandará inicializar la consola, la FPGA y el USB además se configurarán los threads y estará la aplicación que correrá sobre la CPU.
- DebugConsole.c $\Longrightarrow$ Iniciará el periférico UART, creará la tubería entre el UART y la CPU además de programar el funcionamiento de la consola con los comandos incluidos.
- StartStopApplication.c $\Longrightarrow$ Configurará e iniciará el bloque GPIF II así como los puntos de acceso a los bloques GPIF II y USB y las tubería necesaria para llevar los datos desde el bloque GPIF II al bloque USB 3.0. También servirá para que la aplicación destruya las tuberías y los puntos de acceso así como para limpiar la memoria.
- USB_Handler.c $\Longrightarrow$ Script con el cual se iniciará y configurará el bloque USB.
- USB_Descriptors.c $\Longrightarrow$ Donde se encontrarán los descriptores necesarios para la configuración del bloque USB.
- SupplyGpifData.h $\Longrightarrow$ Este archivo contendrá la información necesaria para iniciar el bloque GPIF II ya que es el archivo que proporciona el programa GPIF II Designer después de la compilación.
- Support.c $\Longrightarrow$ Contendrá algunas rutinas de ayuda como checkstatus().
- Application.h $\Longrightarrow$ Dispondrá de las contantes utilizadas durante la programación.
- cyfx_gcc_startup.S $\Longrightarrow$ Utilizado para el arranque del dispositivo y proporcionado por Cypress.
- cyfxtx.c $\Longrightarrow$ Utilizado para el arranque del dispositivo y proporcionado por Cypress.

Lo primero que se hará es configurar los relojes por defecto a una frecuencia de 403 MHz para que el bloque GPIF II pueda funcionar a una frecuencia máxima de 100 MHz, cuando se halla configurado se iniciarán con la función *CyU3PDeviceInit()* , situado en el archivo Startup.c.

Ahora se pasará a activar las memorias cache del dispositivo que serán la Icache, para instrucciones, la Dcache, para datos y la DMAcache, que guardará información sobre las tuberías. Estas memorias se activarán con la función *CyU3PDeviceCacheControl()* situada también en el archivo StartUp.c.

El último punto antes de iniciar el RTOS será configurar la matriz de entrada y salida. Esta matriz se configurará para enviar 16 bits de datos y para utilizar el periférico UART. Para iniciar la matriz se utilizará la función *CyU3PDeviceConfigureIOMatrix()* situada en Startup.c.

Y ahora se iniciará el sistema operativo de tiempo real (RTOS) que servirá para administrar las diferentes tareas del microcontrolador. Este sistema operativo se activará mediante la función *CyU3PKernelEntry()*. A partir de aquí se entrará en un bucle infinito en el cual estará corriendo el sistema operativo.

El siguiente punto será inicializar la consola, esto se hará con la función *InitializeDebugConsole()*. El procesador irá avisando al usuario a través de la consola cuando se van iniciando los diferentes bloques del microcontrolador. Servirá también para controlar el FX3 Super Speed Device ya que a través de esta consola el usuario podrá enviar comandos. Estos comandos serán:

- pclk $\Longrightarrow$ Con el cual se permitirá al usuario cambiar la frecuencia de reloj, esto será muy útil para ver los diferentes efectos que produce en la adquisición de datos el cambio en la frecuencia de reloj.
- threads $\Longrightarrow$ Esta opción permitirá ver al usuario las diferentes threads que en ese momento se encuentren activas.
- reset $\Longrightarrow$ Con el cual se resetearán todos los parámetros de la CPU del dispositivo FX3 Super Speed.
- fpga $\Longrightarrow$ Servirá para cambiar el valor de la señal Reset que se enviará a la FPGA.
- gpif $\Longrightarrow$ Este comando devolverá en que estado del diagrama del GPIF II se encuentra el programa.

Para configurar la consola y que se pueda comunicar con el procesador, lo primero que se hará es activar el driver del periférico UART con la función *CyU3PUartInit()*, luego se configurará la comunicación UART con una tasa de 115.200 baudios, que es la recomendada por el fabricante, y se activará tanto la recepción como la transmisión de datos. Toda esta configuración del periférico se hará con la función *CyU3PUartSetConfig()*. Luego se conectarán los drivers del debugging del sistema al periférico para que todos los mensajes de depuración sean enviados a la consola, esto último se hará con la función *CyU3PDebugInit()* y por último se creará una tubería que vaya desde el periférico UART hasta el procesador pasando por el DMA con la función *CyU3PDMAChannelCreate()*. A través de esta tubería llegaran los comandos enviados por el usuario al procesador y este enviará los mensajes de depuración a la consola.

A continuación se inicializará la FPGA llamando a la función *InitializeFPGA()*. En esta función se configurarán los relojes de los pines GPIO y se iniciarán con la función *CyU3PGpioInit()*. También se activará el pin GPIO27 para que se pueda sobreescribir con la función *CyU3PDeviceGpioOverride()*. Este pin corresponderá a la señal Reset y se inicializará con el valor 1 para que cuando se inicie el dispositivo la FPGA esté en el estado INICIO.

Ahora se procederá a arrancar la aplicación pero antes habrá que asignarle una posición en memoria y un nombre. La aplicación se llamará ApplicationThread y se encontrará en el archivo RunApplication.c.

Lo próximo será iniciar el USB, para ello se llamará a la función *InitializeUSB()* desde el archivo RunApplication.c, pero la función se encontrará en USB_Handler.c. En esta función se inicializará el driver del USB con la función *CyU3PUsbStart()*. Además se configurarán los diferentes callbacks con los que se encontrará el periférico durante su funcionamiento. Estos callbacks describirán el funcionamiento del USB y serán:

- SetupCallback $\Longrightarrow$ La cual establecerá el comportamiento de la conexión USB a través del Class and Vendor que se reciba. Para este experimento se busca una aplicación tipo streaming por lo que el VendorID será "04B4h" el ProductID "00F1h" y la clase "00h".
- EventCallback $\Longrightarrow$ Se utilizará para notificar estados importantes en las transiciones del USB, por ejemplo, cuando se inicia o se para la aplicación.
- LPMRequestCallback $\Longrightarrow$ Para controlar la cantidad de energía que se le pasará al módulo USB que vendrá dado en función de la velocidad de transmisión con la que se quiere trabajar.

Una vez configurados los callbacks, se pasará a la configuración de los descriptores del USB, esto se hará con la función *SetUSBDescriptors()* que se llamará desde el archivo USB_Handler pero se encontrará en el archivo USB_Descriptors.c. Una vez configurado se conectarán los pines y se activará todo el bloque con la función *CyU3PConnectState()*.

Una vez configurado el USB se pasará a la función *StartApplication()*. En esta función lo primero que se hará es iniciar y configurar los relojes del bloque GPIF II, estos relojes utilizarán una fuente interna de reloj y tendrá una frecuencia que será el doble de lo que se envie a la FPGA. Estos relojes se inicializarán con la función *CyU3PPibInit()*.

Ahora se configurará el endpoint que será el consumidor de datos de la tubería que irá desde el bloque GPIF II hasta el Bloque USB 3.0 Device. Esta tubería se creará con la función *CyU3PDmaChannelCreate()*. La tubería se configurará para que el productor de datos sea el bloque GPIF II, el consumidor el bloque USB 3.0 Device y para que esta sea manejada en todo momento por el procesador configurándola en modo AUTO.

Por último se inicializará el bloque GPIF II con la función *StartGPIF()* situada en el código StartStopApplication.c y se activará la variable gliApplicationactive que permitirá entrar en el bucle infinito en el cual está la aplicación desarrollada por el usuario. Esta aplicación estará esperando indefinidamente a que halla algun evento, en cuyo caso se escribirá por la consola.

Para el caso en el cual halla que parar la aplicación, como por ejemplo cuando se cambie la frecuencia de reloj existente, existe la función *StopApplication()* situada en el archivo

StartStopApplication.c. Esta función lo primero que hará es detener el bloque GPIF II con la función *CyU3PGpifDiseable()*, luego se destruirán las tuberías existentes en el bloque Distributed DMA controller con la función *Cyu3PDMAChannelDestroy()*, se desactivarán los puntos de acceso al USB con *CyU3PSetEpConfig()* y por último se desactivará la variable gliApplicationactive para que la función no se pueda ejecutar.

5.1.3 MATLAB

Unos vez realizada la transmisión de datos estos se guardarán en un archivo .bin, situado en un disco duro externo con conectividad USB 3.0. Lo que se hará con MATLAB es coger ese archivo y analizarlo. Para ello se ha creado el script analisis_final_gpif1.

En este script lo primero que hará es buscar el principio de la secuencia, para ello se leerá una cadena de datos y se guardará en el vector num. Se sabe que mientras el dispositivo está en el estado INICIO manda sólo ceros, por tanto, se buscará el primer dato que no sea cero, este primer dato tendrá que ser el número uno, por tanto se comprobará y de no ser así se aumentará la variable error en una unidad.

Ahora se entrará en un bucle en el cual se terminarán de comprobar los datos que aún quedan en el vector num, pero a partir de aqui se tendrán que ir transformando los datos que se tienen en el archivo .bin ya que al archivo llegarán los datos en formato hexadecimal en 4 paquetes de 4 bits cada uno y lo que se hará es, transformar estos paquetes en un número decimal de 16 bits, este número decimal será comparado con un contador, si estos dos números no son iguales se aumentará en una unidad la variable error.

Al terminar este búcle se entrará en otro nuevo del cual sólo se saldrá cuando se halla llegado al final del fichero. En este nuevo bucle se irán leyendo paquetes de 40.000 bits y se comprobarán. Una vez que se ha llegado al final del fichero se comunicarán los datos leídos, el número de errores y el porcentaje de error.

5.1.4 Funcionamiento

Una vez que se han programado todos los códigos, se ha cargado el programa en la FPGA y se ha conectado esta a la placa de desarrollo y el FX3 Super Speed Device al ordenador se iniciará el experimento.

Lo primero que se tiene que hacer es abrir el programa ClearTerminal, con el cual la consola se conectará con el procesador del FX3 Super Speed Device a través del puerto serie.

Ahora se cargará el programa .img en el FX3 e irán apareciendo unos mensajes en el

ClearTerminal a modo de debugging comunicando cuando los bloques se han inicializado correctamente y dirá que estará funcionando para siempre. En este punto el programa se quedará esperando a que se vayan introduciendo los comandos previamente establecidos:

- pclk
- threads
- reset
- fpga
- gpif

Una vez que el programa este inicializado y se halla comprobado que el programa funciona correctamente, se procederá a la lectura de los datos desde el ordenador con el ejecutable CollectData.exe proporcionado por Cypress.

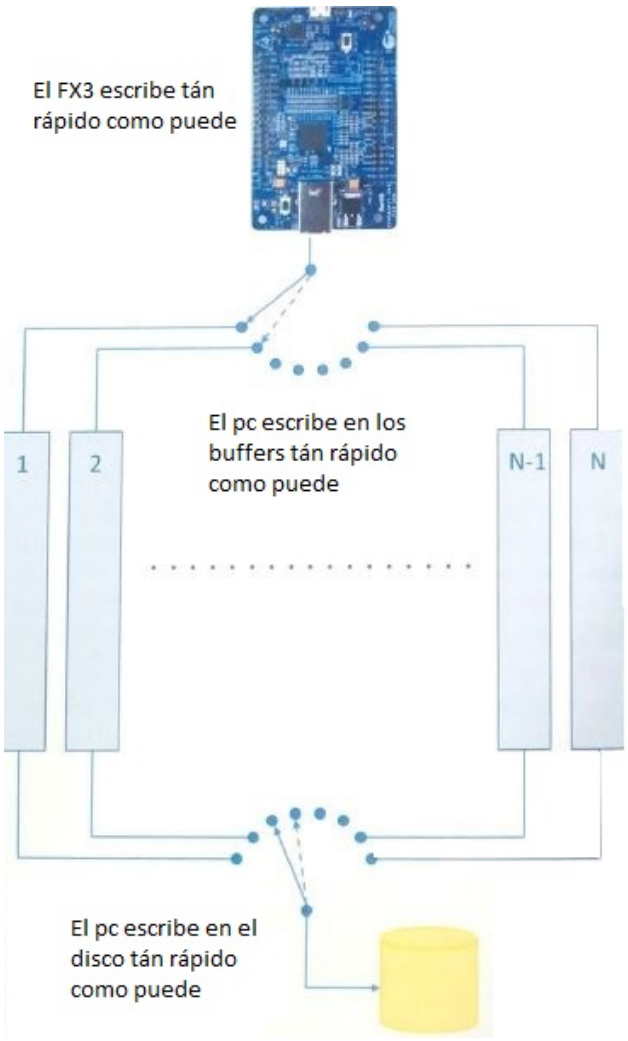

Imagen 23: Esquema de recolección de datos por parte del ejecutable CollectData.exe"

Lo que hará el ejecutable es coger los datos que le irá proporcionando el microcontrolador y guardalos tan rápido como le sea posible en los buffers. Cuando empiece a haber datos en los buffers se irán recogiendo y se escribirán tan rápido como sea posible en el disco duro externo.

Se ha decidido guardar el archivo en un disco duro externo con conectividad USB 3.0 y no en la memoria interna del ordenador, ya que los archivos que se van a tener que guardar son tan grandes que la memoria del ordenador se saturaría y este dejaría de trabajar de forma correcta.

Se llevarán a cabo varios experimentos en los que se irá variando la frecuencia inicial del clock que será de 80 MHz para ver el efecto que hay en la transmisión de datos al utilizar diferentes frecuencias de reloj, cabe esperar que cuanto mayor sea la frecuencia mayor sea el número de errores ya que los datos se enviarán a una mayor velocidad.

El primer experimento se realizará a la frecuencia inicial de 80 MHz, lo primero que se tiene que hacer es llenar los buffers de llegada con ceros para que luego se pueda reconocer bien en el archivo el inicio de la secuencia, por tanto estando la FPGA en el estado RESET se abrirá el ejecutable CollectData.exe se elegirá la opción Receive and Discard Data from the Device con la cual se llenarán los buffers del ordenador de ceros pero no se guardarán los datos en el archivo .bin y se le dará al botón Start Data Transfer.

Ahora que ya se han inundado los buffers de recepción de ceros se procederá a realizar la transmisión de datos para ello en el ejecutable CollectData se elegirá donde se guardará el archivo (rojo) y el tiempo de transmisión de datos (azul) y el programa nos proporcionará la velocidad de transmisión (negro)

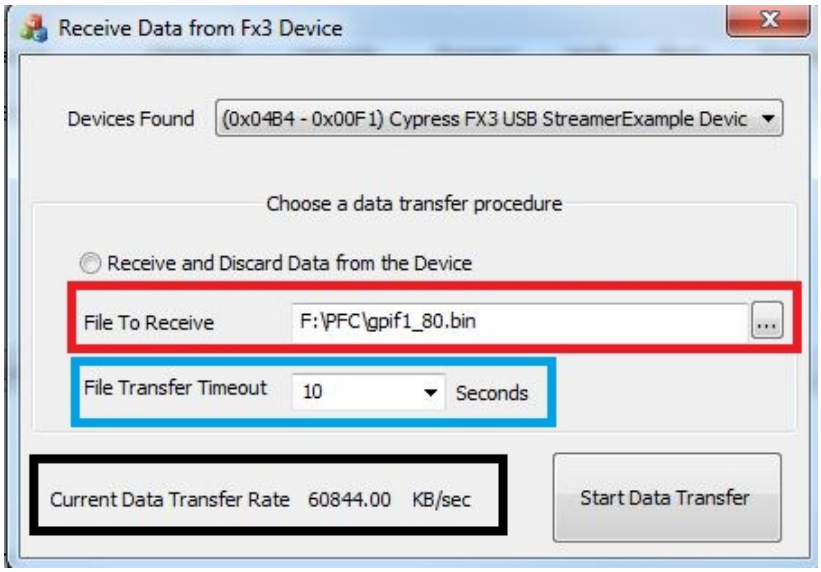

Imagen 24: Apariencia del ejecutable CollectData cuando se realiza la transmisión a 80 MHz

Se cogerá el archivo gpif_80.bin y se analizará con MATLAB obteniendo los siguientes resultados.

```
>> analisis_final_gpif1
inicio busqueda del principio de secuencia
principio secuencia encontrado
final de fichero
datos comprobados = 14999
numero de errores = 14926
porcentaje error = 99.513301 >>
```

Imagen 25: Resultados del analisis del contador a 80 MHz

Como se puede observar el porcentaje de error es grandísimo, haciendo un análisis más exaustivo de los datos de forma visual se puede observar que lo que pasa es que están llegando bloques del contador de forma desordenada, además dentro de esos bloques tambien habrá algunos bits de error lo que provocará un porcentaje de error aún mayor.

Ahora se llevará a cabo una nueva simulación esta vez a una frecuencia de 1 MHz esperando mejorar el porcentaje de error.

```
>> analisis_final_gpif1
inicio busqueda del principio de secuencia
principio secuencia encontrado
final de fichero
datos comprobados = 24707
numero de errores = 24584
porcentaje error = 99.502165 >>
```

Imagen 26: Resultados analisis del contador a 4 MHz

Como se puede volver a observar el porcentaje de error sigue siendo muy grande, inspeccionando de forma visual el archivo se puede observar que es debido esta vez a los saltos que va pegando el contador, ya que los bloques de números que van llegando lo hacen de forma ordenada, por tanto los únicos no fallos se encontrarán al principio de la secuencia.

5.1.5 Conclusiones

Después de llevar a cabo los experimentos se ha comprobado que se pueden enviar datos desde la FPGA hasta el ordenador a través del FX3 Super Speed Device, por lo que todos los bloques del dispositivo funcionan correctamente, pero a la hora de analizar los datos se ha comprobrado que la tasa de error es muy alto y que cuanto mayor es la tasa de transmisión mayor es la tasa de error como cabía esperar.

Haciendo un análisis en profundidad de los datos recibidos se llega a la conclusión de que la tasa de error es tan alta ya que aunque los datos se envíen sin errores estos sobreescriben el buffer del DMA antes de que de tiempo a que este envie los datos por la tubería al bloque USB 3.0, por tanto, lo que hace falta es llevar a cabo algún tipo de control de flujo para no sobreescribir el buffer. Ese control de flujo se llevará a cabo con dos flags que serán

proporcionados por el propio bloque. Esos flags serán DMA0_Ready y DMA0_Watermark que avisarán respectivamente de cuando el buffer está listo para recibir datos y cuando se va a llenar.

5.2 Experimento 2

En este segundo experimento se realizará la transmisión de una señal en diente de sierra de 16 bits que será implementado en una FPGA. Esta señal se enviará al ordenador a través del periférico USB 3.0 que proporcionará el FX3 Super Speed Device.

Para que no se sobreescriba el buffer del DMA habrá control de flujo, para ello se utilizarán dos flags que serán proporcionados por el bloque Distributed DMA Controlled. Esos flags serán DMA0_Ready que comunicará cuando el buffer está listos para recibir datos y el flag DMA0_Watermark que indicará cuando el buffer está casi lleno.

El envío de la señal estará controlado por un interruptor que tendrá que ser activado por el usuario para iniciar la transmisión de datos, además se mandará una señal de control al FX3 Super Speed para encender un LED durante el envío. Estos datos serán recibidos por el FX3 que en la comunicación con la FPGA será el slave, esto quiere decir que el bloque GPIF II sólo leerá los datos cuando reciba la señal WR, además en este experimento se utilizarán los flags DMA que permitirán escribir en el buffer del DMA sólo cuando este tenga espacio libre con lo que se conseguirá no perder datos, estos flags serán enviados a la FPGA que los utilizará para cambiar de estado en función de si se puede escribir o no.

Cuando halla sitio en el buffer del DMA, el bloque GPIF II leerá y transferirá los datos al bloque USB 3.0 Device a través de una tubería por el Distributed DMA Controller. El bloque del USB será el encargado de enviar la información al ordenador.

Esta información será recolectada y guardada en un archivo .bin con la ayuda del ejecutable CollectData.exe, este ejectuable tendrá que ser modificado para permitir la lectura de datos durante horas. El fichero creado será analizado con MATLAB proporcionando el número de datos leídos, el número de errores y su proporción respecto a la cantidad de datos y por último a la vista de los datos proporcionados se extraerán las conclusiones. Además se realizarán pruebas a varias frecuencias para ver la influencia que tiene la tasa de transmisión sobre la probabilidad de error.

Durante el experimento se tendrán que programar las siguientes partes:

- La FPGA para lo cual se utilizará:
  - Quartus II web Edition para escribir el código.
  - ModelSim-Altera 10.1e para simular el programa.
- El FX3 SuperSpeed device para lo cual se dispondrá de:
  - GPIF II designer para programar el bloque GPIF II.
  - EZ USB Suite(eclipse) para escribir los programas que controlarán el chip FX3.
  - USB Control Center para cargar los programas en la RAM del dispositivo.

- El ordenador donde se utilizarán los siguientes programas para la transmisión:
  - Visual Studio 2013 para modificar el ejecutable CollectData.exe.
  - MATLAB para el análisis de datos.
  - CollectData.exe para recolectar los datos.

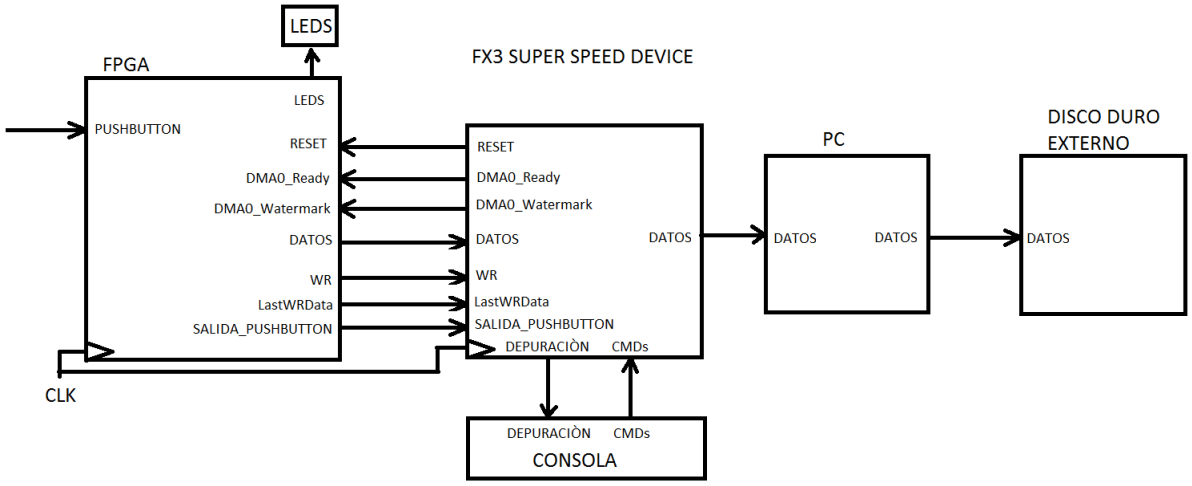

Imagen 27: Configuración del segundo experimento

5.2.1 FPGA

En la FPGA se implementará un programa que permitirá el envio de una señal en diente de sierra con 16 bits. Durante el envio habrá control de flujo para no sobreescribir el buffer del DMA. Se diseñará el siguiente diagrama de bloques en la FPGA para hacer posible el programa.

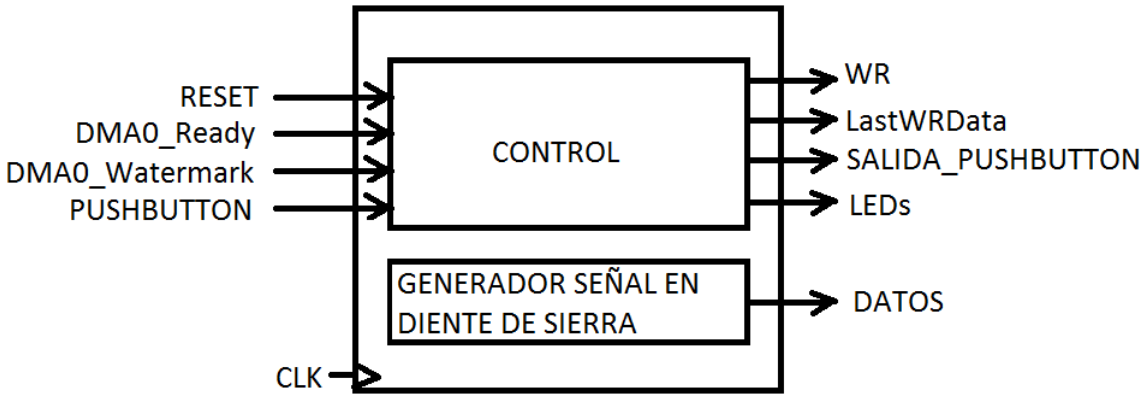

Imagen 28: Diagrama de bloques de la FPGA en el experimento 2

Se tendrán cinco entradas:

1. DMA0_Ready $\Longrightarrow$ Avisará cuando el buffer del DMA esté listo para recibir datos.
2. DMA0_Watermark $\Longrightarrow$ Indicará cuando el buffer del DMA esté a punto de llenarse.

3. CLK $\Longrightarrow$ Señal de reloj.
4. Pushbutton $\Longrightarrow$ Para iniciar o parar la transmisión de datos.
5. reset $\Longrightarrow$ Servirá para reinicializar la FPGA

y cinco salidas:

1. DQ[15:0] $\Longrightarrow$ Señal en diente de sierra
2. LastWRData $\Longrightarrow$ Indicará que se va a enviar el último dato
3. salida_pushbutton $\Longrightarrow$ Controlaá los leds situados en el FX3 Super Speed Device.
4. WR $\Longrightarrow$ Controlará la lectura de datos en el microcontrolador.
5. LED $\Longrightarrow$ Servirá para indicar al usuario en que estado se encuentra la FPGA.

Ahora se pasará a diseñar el bloque de control que tendrá la siguiente máquina de estados:

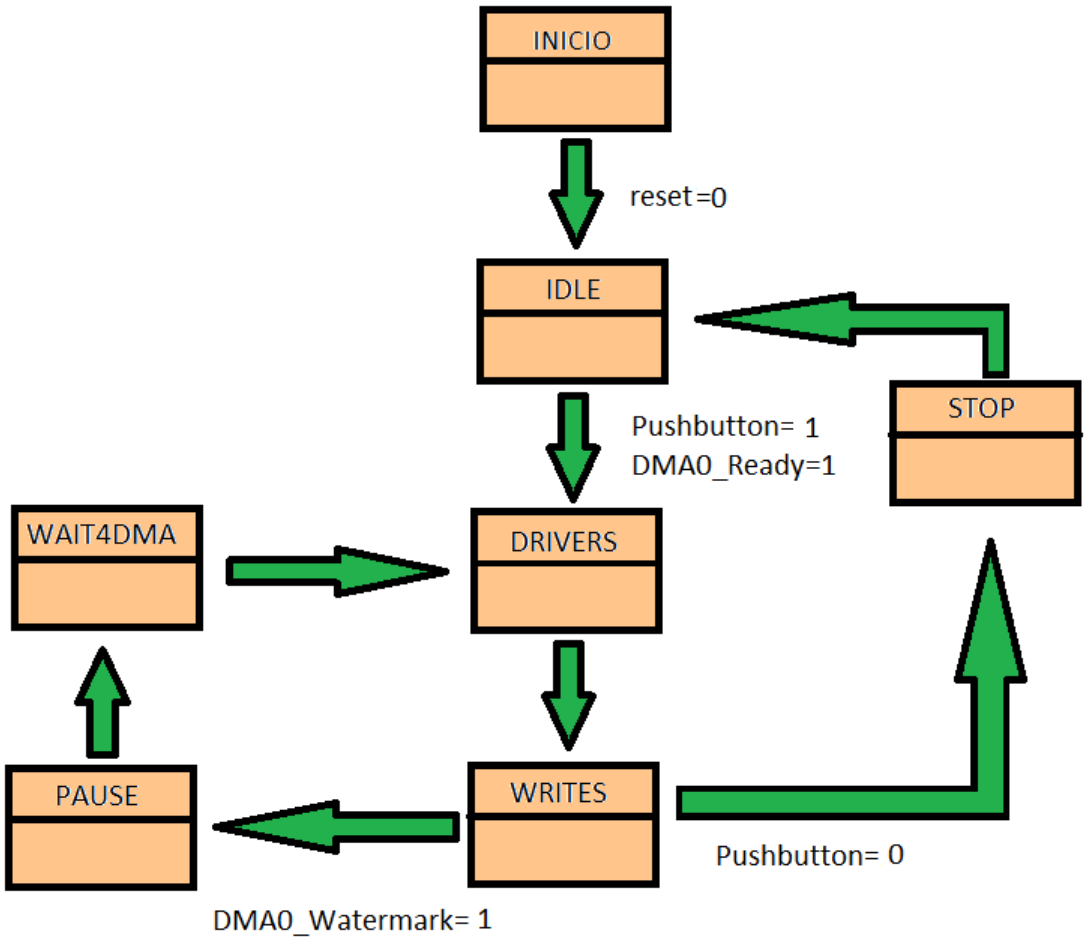

Imagen 29: Máquina de estados del experimento 2

Al encender la FPGA el primer estado al cual se irá será el estado INICIO, en este estado no se enviarán datos, por tanto WR=0. Se pasará al siguiente estado el IDLE cuando reciba la señal reset=0. Para que el usuario sepa que se encuentra en este estado se encendán todos los LEDs.

Al estado IDLE se pasará desde INICIO cuando llegue la señal reset=0. En este estado ya si que se escribirá por tanto WR=1, esto inundará los buffers del ordenador inicialmente de ceros lo que facilitará posteriormente en el análisis de datos pudiendo encontrar con más facilidad el inicio de la frecuencia. Además este estado se utilizará también para reiniciar todas la variables. Para pasar al siguiente estado se tienen que cumplir dos condiciones, la primera es que el usuario encienda el interruptor para que se pueda iniciar la transmisión de datos Pushbutton=1, y la segunda es que DMA0_ready=1, este flag cuando esta activo indicará que hay sitio en el buffer del DMA y por lo tanto se puede escribir. Para saber que nos encontramos es este estado se encenderá el primer LED.

Al estado DRIVERS se podrá llegar desde el estado IDLE o WAIT4DMA, en este estado se permanecerá un ciclo antes de pasar al estado WRITES y servirá para activar la escritura WR=1. En este estado se encenderá el LED número dos.

El estado WRITES será el más importante en este diagrama ya que será en donde se creará la señal en diente de sierra, además se enviará la señal al FX3 Super Speed Device por lo que WR=1, pero no se puede escribir indefinidamente porque sino se sobreescribiría el buffer del DMA, por tanto se utilizará la señal DMA0_Watermark que indicará cuando el buffer está a punto de llenarse, entonces cuando esta señal sea uno se irá al estado PAUSE. También existe la posibilidad de que sea el usuario el que quiera dejar de mandar la señal apagando el interruptor, en este caso se irá al estado STOP. En este estado se encenderá el LED número tres.

En el estado PAUSE se esperarán 3 ciclos de reloj que es lo que le costará al flag DMA0_Ready reaccionar y cambiar su valor para indicar que el buffer no se encuentra en ese momento preparado para recibir datos, después de esos ciclos se pasará al estado WAIT4DMA.

En WAIT4DMA se esperará a que el buffer se vacie y se pueda volver a escribir, cuando el buffer este listos el flag DMA0_Ready se activará, pero para estar seguros de que se puede acceder al buffer con seguridad y sin perder datos, se esperarán 27 ciclos y luego se pasará al estado DRIVERS. Ese número de ciclos se han sacado de las observaciones de las señales con el osciloscopio durante los experimentos.

Al estado STOP se llegará desde el estado WRITES cuando se desactive el interruptor, en este estado se escribirá un último dato y posteriormente se desactivará la escritura antes de pasar al estado IDLE donde se esperará a que el usuario vuelva a activar el interruptor para iniciar la transmisión.

Una vez que se ha escrito el código con el programa Quartus II web Edition, este se tiene que depurar para verificar que no se han cometido errores en la sintaxis del programa y habra que simularlo para comprobar que el programa escrito en VHDL se comporta exactamente como se ha diseñado. Para llevar a cabo la simulación se utilizará el programa ModelSim.

La primera simulación que se llevará a cabo es un proceso normal de escritura, se seguirá la siguiente secuencia de estados: RESET-IDLE-DRIVERS-WRITES-PAUSE.

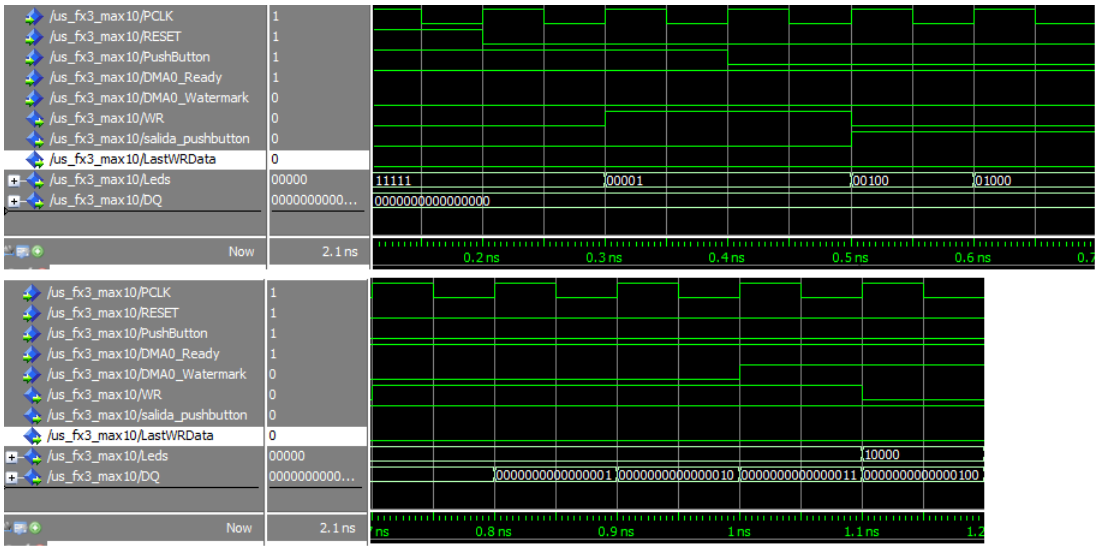

Imagen 30: Primera simulación experimento 2

En la siguiente simulación se podra observar que pasa al desconectar el interruptor. Se seguirá la siguiente secuencia de estados: RESET-IDLE-DRIVERS-WRITES-STOP-IDLE.

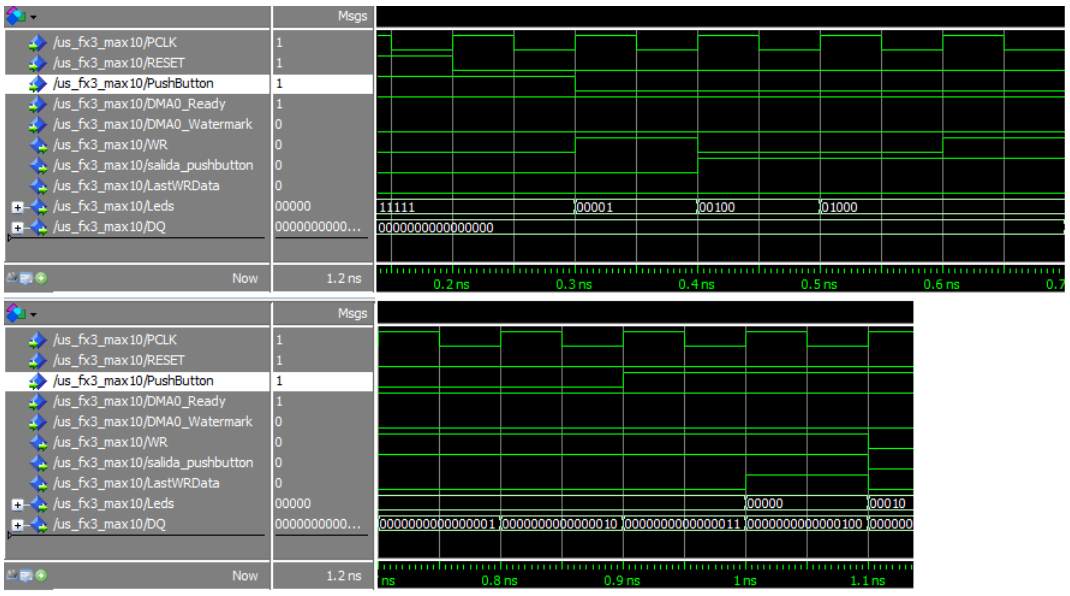

Imagen 31: Segunda simulación experimento 2

Una vez que se ha simulado y escrito el código habra que realizar la asignacion de pines:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
DMA0_Ready	Input	PIN_50	3	B3_N0	PIN_50	2.5 V (default)
DMA0_Watermark	Input	PIN_54	3	B3_N0	PIN_54	2.5 V (default)
DQ[15]	Output	PIN_52	3	B3_N0	PIN_52	2.5 V (default)
DQ[14]	Output	PIN_48	3	B3_N0	PIN_48	2.5 V (default)
DQ[13]	Output	PIN_47	3	B3_N0	PIN_47	2.5 V (default)
DQ[12]	Output	PIN_46	3	B3_N0	PIN_46	2.5 V (default)
DQ[11]	Output	PIN_44	3	B3_N0	PIN_44	2.5 V (default)
DQ[10]	Output	PIN_43	3	B3_N0	PIN_43	2.5 V (default)
DQ[9]	Output	PIN_41	3	B3_N0	PIN_41	2.5 V (default)
DQ[8]	Output	PIN_39	3	B3_N0	PIN_39	2.5 V (default)
DQ[7]	Output	PIN_70	4	B4_N0	PIN_70	2.5 V (default)
DQ[6]	Output	PIN_69	4	B4_N0	PIN_69	2.5 V (default)
DQ[5]	Output	PIN_65	4	B4_N0	PIN_65	2.5 V (default)
DQ[4]	Output	PIN_62	4	B4_N0	PIN_62	2.5 V (default)
DQ[3]	Output	PIN_61	4	B4_N0	PIN_61	2.5 V (default)
DQ[2]	Output	PIN_60	3	B3_N0	PIN_60	2.5 V (default)
DQ[1]	Output	PIN_57	3	B3_N0	PIN_57	2.5 V (default)
DQ[0]	Output	PIN_55	3	B3_N0	PIN_55	2.5 V (default)
LastWRData	Output	PIN_58	3	B3_N0	PIN_58	2.5 V (default)
Leds[4]	Output	PIN_14	1A	B1_N0	PIN_14	2.5 V (default)
Leds[3]	Output	PIN_13	1A	B1_N0	PIN_13	2.5 V (default)
Leds[2]	Output	PIN_12	1A	B1_N0	PIN_12	2.5 V (default)
Leds[1]	Output	PIN_11	1A	B1_N0	PIN_11	2.5 V (default)
Leds[0]	Output	PIN_10	1A	B1_N0	PIN_10	2.5 V (default)
PCLK	Input	PIN_38	3	B3_N0	PIN_38	2.5 V (default)
PushButton	Input	PIN_84	5	B5_N0	PIN_84	2.5 V (default)
RESET	Input	PIN_66	4	B4_N0	PIN_66	2.5 V (default)
salida_pushbutton	Output	PIN_64	4	B4_N0	PIN_64	2.5 V (default)
WR	Output	PIN_45	3	B3_N0	PIN_45	2.5 V (default)

Imagen 32: Asignación de pines en el experimento 2

Por último el programa se volverá a compilar y se creará un archivo .sof que contendrá el programa que se cargará en la FPGA.

5.2.2 FX3 Super Speed device

En esta sección se procederá a la programación y configuración de todos los periféricos del FX3 Super Speed Device. Recordemos que para la configuración y programación se utilizarán dos programas diferentes, para la programación del bloque GPIF II que se encargará de la comunicación con la FPGA se utilizará el programa GPIF II Designer y para la programación y compilación del firmware se utilizará el programa Cypress EZ USB Suite ambos proporcionados por Cypress. Además una vez que se tenga el programa compilado y listo en un archivo .img este se cargará en el microcontrolador con la ayuda del programa USB Control Center.

Lo primero que se programará será el bloque GPIF II del FX3 Super Speed Device, este bloque como ya se ha comentado se encarga de la relación entre el dispositivo y la FPGA.

Se configurará el interface con las siguientes características:

- Tipo de interface $\Longrightarrow$ Slave.
- Tipo de comunicación $\Longrightarrow$ Sincrona.
- Fuente del Reloj $\Longrightarrow$ Interna.
- Flanco de Reloj $\Longrightarrow$ Ascendente.
- Codificación $\Longrightarrow$ Little endian.
- Anchura del bus de datos $\Longrightarrow$ 16 Bits.

Una vez que se han configurado los ajustes de la interfaz se tendrá que indicar el número de entradas, salidas y flags que se utilizarán durante la transmisión.

- Entradas
  - WR $\Longrightarrow$ Esta señal indicará al bloque GPIF II cuando tiene permiso de escritura. Será enviada desde la FPGA.
  - LastWRData $\Longrightarrow$ La señal será enviada desde la FPGA para indicar que sólo queda un dato por escribir. Esta señal será enviada desde el estado WRITES por la FPGA antes de pasar al estado STOP.
  - Datos $\Longrightarrow$ Se tratará de una señal de 16 bits, estos bits se enviarán de forma paralela y contendrán la información de la señal en diente de sierra que se manda.
- Salidas
  - CLK $\Longrightarrow$ Señal de reloj que alimentará a la FPGA.
  - Reset $\Longrightarrow$ Se trata de una señal que se enviará a la FPGA cuando se quiere reinicializar todas la variables.
- Flags DMA
  - DMA0_Ready $\Longrightarrow$ Flag que será enviado a la FPGA e indicará si esta activo que el buffer del DMA está para recibir datos.
  - DMA0_Watermark $\Longrightarrow$ Flag que indicará a la FPGA que el buffer del DMA está a punto de llenarse para que cambie de estado y vaya a PAUSE.

Por último se tendrá que asignar a cada señal un pin de propósito general (GPIO):

señal	Pin GPIO	Descripción
DataBus	GPIO[0:15]	Datos que contendrán la información de la señal en diente de sierra
CLK	GPIO 16	Señal de reloj que alimentará a la FPGA
WR	GPIO 17	Señal que indicará cuando se puede escribir
LastWRData	GPIO 19	Indicará cuando se esta llegando al final de la secuencia
DMA0_Ready	GPIO 21	Dirá a la FPGA cuando se puede escribir en el buffer del DMA
DMA0_Watermark	GPIO 22	Indicará a la FPGA que el buffer del DMA está a punto de llenarse

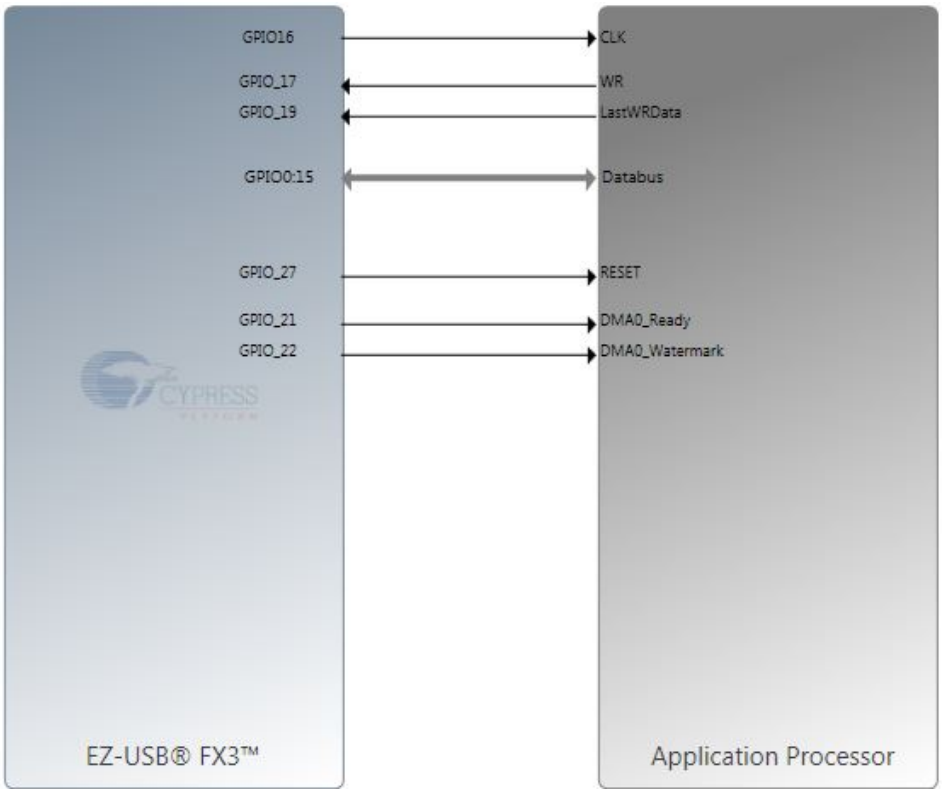

Imagen 33: Configuración de la matriz IO del GPIF II experimento 2

Una vez que se ha configurado el bloque se pasará a su programación a través de un diagrama de estados. Cabe recordar que este bloque interactúa con la FPGA por lo que será un espejo de lo que pasa allí.

Inicialmente se estará en el estado START ya que se trata del estado de inicio del programa que es requerido siempre por el bloque GPIF II, acto seguido se pasará al estado WAIT4WR ya que la señal LOGIC_ONE indica que se pase de forma incondicional de un estado a otro.

En el estado WAIT4DMA se esperará indefinidamente hasta que la FPGA envie la señal de WR, esto significará que ya se puede empezar a leer los datos que están llegando a los pines GPIO del dispositivo, por tanto se pasará al estado READ.

En el estado READ se leerán los datos que van llegando y se enviarán los flags DMA0_Ready y DMA0_Watermark a la FPGA, el flag DMA0_Ready se enviará activo a la FPGA cuando halla sitio para escribir en el buffer del DMA y el flag DMA0_Watermark se enviará unas ciclos antes de que el buffer del DMA se llene, ese número de ciclos será elegido por el usuario. Todo esto será posible gracias a la acción IN_DATA.

La señal WR enviada desde la FPGA se encargará de controlar la lectura de los datos por parte del dispositivo, por tanto cuando llegue la señal WR=0 habrá que dejar de leer para esto se pasará a un estado de pausa denominado WAIT2 o si llega la señal LastWRData indicando que se va a terminar la escritura se pasará al estado DONE.

En el estado WAIT2 se esperará bien a que vuelva a llegar la señal WR activa indicando que se puede volver a leer por lo que se irá otra vez al estado READ o a que llegue la señal LastWRData indicando que se está al final de la transmisión y que hay que dejar de leer por lo que se pasará al estado DONE.

En el estado DONE se utilizará la acción COMMIT esta acción servirá para enviar toda la información que se tenga almacenada en los buffers del bloque GPIF II al bloque USB 3.0, ya con los buffers liberados se pasará al estado WAIT4DMA donde se esperará hasta que empiece una nueva comunicación.

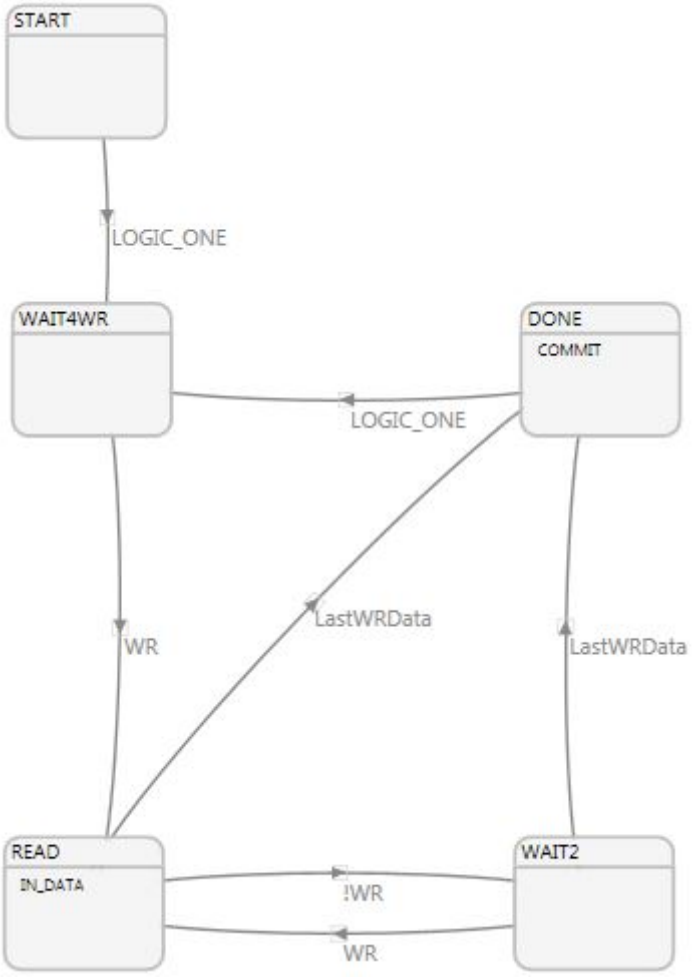

Imagen 34: Diagrama de estados del GPIF II experimento 2

Una vez diseñado el programa, éste se compilará y si no ha habido ningún error en compilación se creará un archivo .h (gpif4.h) que tendrá la configuración y programación del bloque GPIF II que se tendrá que volver a compilar más tarde con todos los archivos del firmware.

Una vez configurado y programado el bloque GPIF II, es decir, la relación entre el FX3 Super Speed Device y la FPGA, se pasará a la configuración de los demás bloques y a la programación del firmware para lo cual se utilizarán tanto las funciones API proporcionadas por Cypress como las librerías estándar de C y el programa EZ USB Suite(eclipse) IDE.

Los scripts que se tendrán que programar son:

- Startup.c $\Longrightarrow$ Donde se inicializarán los parámetros de la CPU y la matriz de entrada y salida.
- RunApplication.c $\Longrightarrow$ En la cual se mandará inicializar la consola, la FPGA y el USB además se configurarán los threads y estará la aplicación que correrá sobre la CPU.
- DebugConsole.c $\Longrightarrow$ Iniciará el periférico UART, creará la tubería entre el UART y la CPU además de programar el funcionamiento de la consola con los comandos incluidos.
- StartStopApplication.c $\Longrightarrow$ Configurará e iniciará el bloque GPIF II así como los puntos de acceso a los bloques GPIF II y USB y las tuberías necesarias para llevar los datos de un bloque a otro. También servirá para que la aplicación elimine las tuberías y los puntos de acceso así como para limpiar la memoria.
- USB_Handler.c $\Longrightarrow$ Script con el cual se iniciará y configurará el bloque USB.
- USB_Descriptors.c $\Longrightarrow$ Donde se encontrarán los descriptores necesarios para la configuración del bloque USB.
- gpif4.h $\Longrightarrow$ Este archivo contendrá la información necesaria para iniciar el bloque GPIF II ya que es el archivo que proporciona el programa GPIF II Designer después de la compilación.
- Support.c $\Longrightarrow$ Contendrá algunas rutinas de ayuda como checkstatus().
- Application.h $\Longrightarrow$ Dispondrá de las constantes utilizadas durante la programación.
- cyfx_gcc_startup.S $\Longrightarrow$ Utilizado para el arranque del dispositivo y proporcionado por Cypress.
- cyfxtx.c $\Longrightarrow$ Utilizado para el arranque del dispositivo y proporcionado por el Cypress.

Lo primero que se hará es configurar los parámetros de la CPU en el archivo Startup.c. Se configurará el reloj para utilizar una fuente interna y una frecuencia de 400 MHz que permitirá utilizar el bloque GPIF II a una frecuencia máxima de 100 MHz. El reloj se iniciará con la función *CyU3PDeviceInit()*.

una vez configurado el reloj se iniciarán la memoria cache del dispositivo, estas memorias son Icache para instrucciones, Dcache para datos y DMAcache para guardar información sobre el bloque Distributed DMA controller. Estas memorias cache se iniciarán con la función *CyU3PDeviceCacheControl()*.

A continuación se configurará la matrix de entrada y salida para enviar datos de 16 bits indicando que se va a utilizar el periférico UART, se configurará el pin GPIO26 perteneciente al reset y el pin GPIO45 que se corresponderá con el interruptor conectado a la FPGA, esta matriz se configurará con la instrucción *CyU3PDeviceConfigureIOMatrix()*. Por último se iniciará el sistema operativo de tiempo real con la función *CyU3PKernelEntry()* situada en el archivo Startup.c.

El siguiente punto será inicializar la consola, esto se hará con la función *InitializeDebugConsole()*. El procesador irá avisando al usuario a través de la consola cuando se van iniciando los diferentes bloques del microcontrolador. Servirá también para controlar el FX3 Super Speed Device ya que a través de esta consola el usuario podrá enviar comandos. Estos comandos serán:

- pclk $\Longrightarrow$ Con el cual se permitirá al usuario cambiar la frecuencia de reloj, esto será muy útil para ver los diferentes efectos que produce en la adquisición de datos el cambio en la frecuencia de reloj.
- threads $\Longrightarrow$ Esta opción permitirá ver al usuario las diferentes threads que en ese momento se encuentren activas.
- reset $\Longrightarrow$ Con el cual se resetearán todos los parámetros de la CPU del dispositivo FX3 Super Speed.
- fpga $\Longrightarrow$ Servirá para cambiar el valor de la señal Reset que se enviará a la FPGA.
- gpif $\Longrightarrow$ Este comando devolverá en que estado del diagrama del GPIF II se encuentra el programa.

Para configurar la consola y que se pueda comunicar con el procesador, lo primero que se hará es activar el driver del periférico UART con la función *CyU3PUartInit()*, luego se configurará la comunicación UART con una tasa de 115.200 baudios, que es la recomendada por el fabricante, y se activará tanto la recepción como la transmisión de datos. Toda esta configuración del periférico se hará con la función *CyU3PUartSetConfig()*.

Luego se conectarán los drivers del debugging del sistema al periférico para que todos los mensajes de depuración sean enviados a la consola, esto último se hará con la función *CyU3PDebugInit()* y por último se creará una tubería que vaya desde el periférico UART

hasta el procesador pasando por el DMA con la función *CyU3PDMAChannelCreate()*. A través de esta tubería llegaran los comandos enviados por el usuario al procesador y este enviará los mensajes de depuración a la consola.

Lo siguiente será inicializar los pines que tendrán comunicación directa con la FPGA y que no estarán controlados por el bloque GPIF II sino que serán leídos y escritos desde el procesador, para ello se ejecutará la función *InitializeFPGA()* del archivo Runapplication.c. Lo primero que se hará en la función es inicializar los relojes del GPIO, se utilizará una fuente de reloj interna y se inicializará con la función *CyU3PGpioInit()*.

Ahora se configurarán los pines. El primer pin en ser configurado será el pin GPIO27 perteneciente al reset y el GPIO54 perteneciente al LED, para que los pines se puedan sobreescribir se utilizará la función *CyU3PDeviceGpioOverride()* y se le dará el valor 1 al reset para que la FPGA se inicie en el estado RESET, esto se hará con la función *CyU3PGpioSetSimpleConfig()* y luego se configurará el pin GPIO26 para que pueda leer los datos que le llegan con la función *CyU3PIOSetSimpleConfig()* .

Por último antes de entrar en el bucle infinito perteneciente a la aplicación habrá que configurar ésta asignándole un nombre, una posición en memoria e indicando al sistema operativo que se quiere que la aplicación se inicie de forma inmediata.

Ahora se pasará a la aplicación que estará en la función *ApplicationThread()* en el archivo RunApplication.c. Lo primero que se hará es inicializar el USB invocando a la función *InitializeUSB()*, esta función se encontrará en el archivo USB_Handler.c. Lo primero que se hará es inicializar el driver del módulo USB, esto se hará con la función *CyUSBStart()*. Una vez iniciado se configurarán los callbacks con las funciones *CyU3PUsbRegisterSetUpCallback()*, *CyU3PUsbRegisterEventCallback()* y *CyU3PUsbRegisterLPMRequestCallback()*. Los callbacks serán:

- SetupCallback $\Longrightarrow$ La cual establecerá el comportamiento de la conexión USB a través del Class and Vendor que se reciba. Para este experimento se busca una aplicación tipo streaming por lo que el VendorID será "04B4h" el ProductID "00F1h" y la clase "00h".
- EventCallback $\Longrightarrow$ Se utilizará para notificar estados importantes en las transiciones del USB, por ejemplo, cuando se inicia o se para la aplicación.
- LPMRequestCallback $\Longrightarrow$ Para controlar la cantidad de energía que se le pasará al módulo USB que vendrá dado en función de la velocidad de transmisión con la que se quiere trabajar.

A continuación se configurará el USB con sus descriptores con la función *SetUSBDescriptors()* invocada desde el archivo USB_Handler.c y contenida en USB _Descriptors.c. Por último se activará la conexión USB con la función *CyU3PConnectState()* y se irá a la función *StartApplication()*.

La función StartApplication estará en el archivo StartStopApplication.c. En esta función lo primero que se hará es preguntar por la velocidad del USB con *CyU3PGetSpeed()*, con esto se sabrá con que tipo de USB se va a transmitir, en este caso será un USB 3.0. Luego se pasará a iniciar los relojes del GPIF II, que irán al doble de frecuencia que la FPGA. Estos se iniciarán con la función *CyU3PPibInit()*. A continuación se configurarán los endpoints con *CyU3PSetEpConfig()* y la tubería que irá desde el bloque GPIF II hasta el USB a través del Distributed DMA Controller con *CyU3PDMAChannelCreate()*. La tubería se configurará para que el productor sea el bloque GPIF II, el consumidor el bloque USB y quien la controle sea el procesador, por tanto estará en modo AUTO. Por último se ejecutará la función *StartGPIF()* donde se cargará el programa de este bloque con *CyU3PGpifLoad()* y se configurará el flag DMA0_Watermark para que se active dos ciclos antes de que el buffer se llene con *CyU3PGpifSocketConfigure()* y se activará la variable glIsApplicationActive para que se puede ejecutar la aplicación del usuario.

En esta aplicación situada en RunApplication.c se comprobará en todo momento el estado del GPIO45 y se encederá o apagará el LED en función de su estado. Cuando se encienda el LED significará que hay una transmisión en marcha.

Una vez que se han escrito todos los códigos estos se compilarán y se creará un archivo .img que se cargará en el FX3 Super Speed Device a través del USB 3.0 con el programa USB Control Center.

5.2.3 Visual Studio 2013

Para recolectar los datos que van llegando al ordenador se utilizará el ejecutable CollectData.exe, este ejecutable es proporcionado por Cypress pero tiene una limitación en tiempo de 50 segundos y lo que se busca es una transmisión de datos a alta velocidad y durante mucho tiempo, por ello se utilizará este programa para modificar el ejecutable e introducir los siguientes tiempos 60, 120, 180, 240, 300, 600, 900, 1.800, 2.700, 3.600 y 432.000 segundos.

5.2.4 MATLAB

Una vez que se halla finalizado la transmisión de datos se dispondrá de un archivo .bin en el cual se tendrán todos los datos que se han enviado, pero no se sabe si la transmisión ha estado libre de errores y para ello se utilizará MATLAB, se dispondrá de dos scripts uno llamado analisis_final_gpif2.m y otro llamado representar_datos.m

El primer script analisis_final_gpif2.m servirá para analizar los datos y ver si se tienen errores, para ello nos ayudaremos de una máscara que se irá comparando con la datos que se han recibido para ver si ha habido errores en la transmisión.

Lo primero que se hará es abrir el archivo donde se encontrarán los datos y se buscará el principio de la secuencia, para ello se irán leyendo y guardando los datos en un

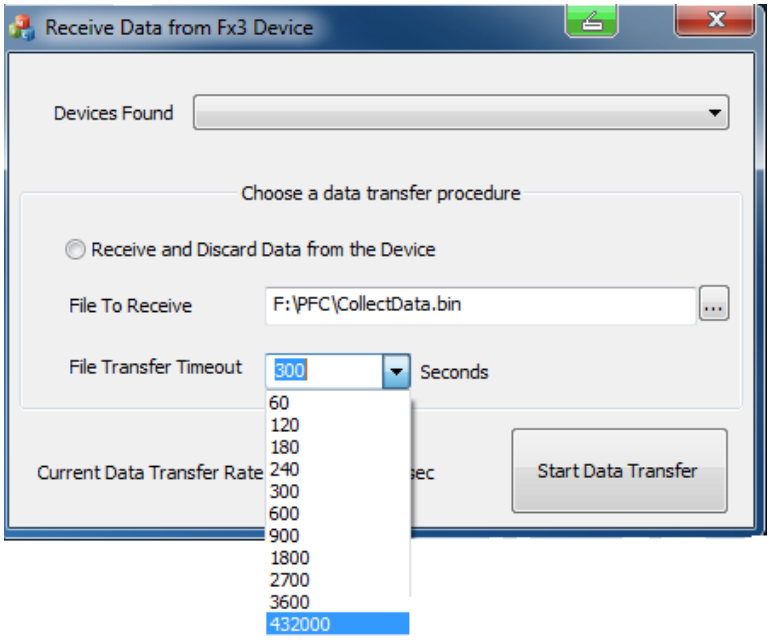

Imagen 35: Apariencia del ejecutable CollectData.exe después de su modificación

vector de 10.000 posiciones, por tanto una vez encontrado el principio de la secuencia se pasará a otro bucle en el cual se terminará de comprobar el conjunto de datos ya leídos, en el caso de que se detecte un error la variable error se incrementará en una unidad.

Luego se entrará en un bucle infinito del cual sólo se saldrá con la sentencia break en el caso que se halla llegado al final de fichero, esto se indicará con la variable final_fichero. En este bucle lo que se hará es rellenar el vector de 10.000 posiciones y compararlo con la máscara de la que se dispone, en el caso de que halla un error se sumará una unidad a la variable error.

Por último una vez que se halla salido del búcle el programa devolverá los siguientes parámetros: número de datos que se han leido, número de errores y relación errores/número de datos leídos.

En el segundo script representar_datos.m se representarán los datos de forma gráfica. En este script lo que se hará es abrir el archivo, luego se encontrará el principio de la secuencia y se irán leyendo bloques de datos que se guardarán en un vector de 1000 posiciones que serán transformados a un entero primero a través de la función lectura_2bytes y representados seguidamente a través de la instrucción plot.

5.2.5 Funcionamiento

Una vez que se tienen todos los códigos programados y compilados además de todas las partes del experimento en funcionamiento, se comprobará en primer lugar que la transmisión se hace de manera correcta para ello se analizarán las señales en el osciloscopio para luego pasar a realizar diferentes pruebas, estas pruebas consistirán en primer lugar

en una transmisión de datos a una frecuencia de 1 MHz, velocidad de transmisión en torno a 2 MBps, para ver que a esa frecuencia se puede establecer una comunicación sin errores y por tanto se cumplirán los requerimientos exigidos para el proyecto, una vez hecho se hará un análisis de la relación entre la tasa de transmisión y la tasa de error.

Se empezará por el análisis de la señales en el osciloscopio para ello primero se recordará como funciona el programa. En un primer momento se estará en un estado de stand by en el cual lo único que se hará es escribir ceros para llenar los buffers del ordenador, esto se hace para facilitar el encontrar el principio de la secuencia, y se estará así hasta que el usuario encienda el interruptor (pushbutton=1), en este momento se iniciará la escritura cuando el buffer del DMA esté listo (DMA0_Ready=1). Se estará escribiendo hasta que se reciba el flag DMA0_Watermark. Mientras se realiza la escritura la señal WR=1, el flag DMA0_Ready=1 y el otro flag el DMA0_Watermark=0.

Dos ciclos antes de que el buffer del DMA se halla llenado y no acepte más datos el flag DMA0_Watermark se activará, en ese momento se dejará de escribir (WR=0) y se iniciará un contador para dejar tiempo a que se vacíen el buffer del DMA, al terminar el contador si el flag DMA0_Ready=1 se volverá a escribir hasta que casi se vuelvan a llenar los buffers y se vuelva a recibir el flag DMA0_Watermark y así se seguirá hasta que se apague el interruptor y se reciba la señal pushbutton=0.

Durante la transmisión se tendrán las siguientes señales en el osciloscopio en la relación FPGA-GPIF II:

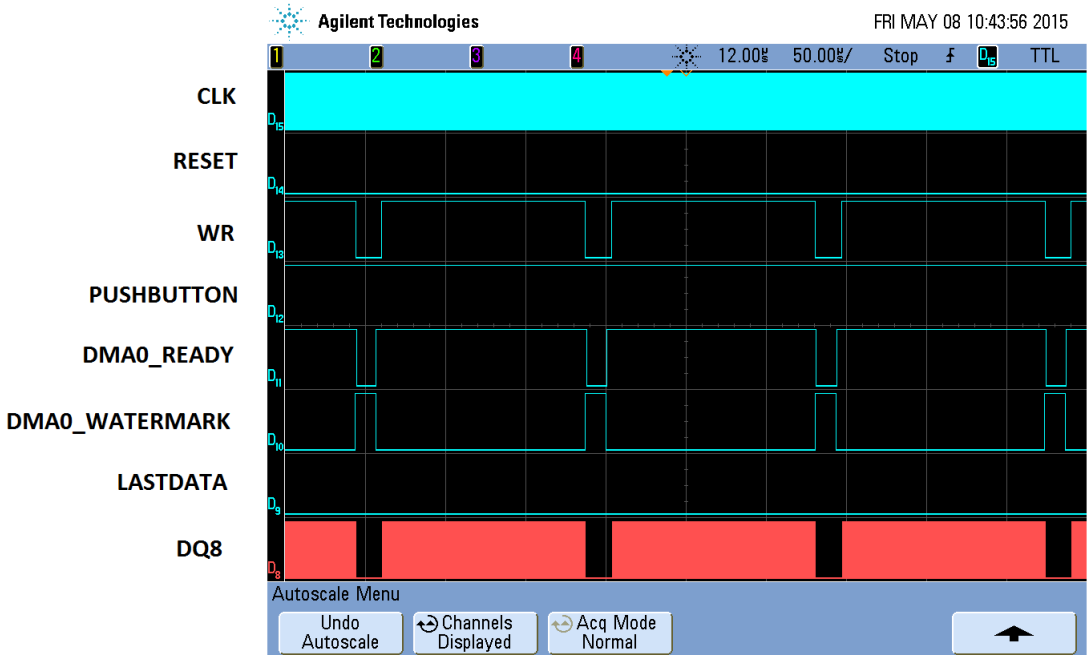

Imagen 36: Señales del experimento 2 I

Haciendo un zoom se pueden observar mejor las transiciones:

Como se puede observar mientras se está realiza la escritura el flag DMA0_Watermark se activará indicando que el buffer se llenará en dos ciclos de reloj, en eso momento se dejará de escribir, el flag DMA0_Ready permanecerá 4 ciclos más activo, que se corresponderán con dos ciclos que aún falta para que se llenen el buffer más los ciclos que le cueste reaccionar que serán dos de espera reaccionando al tercero.

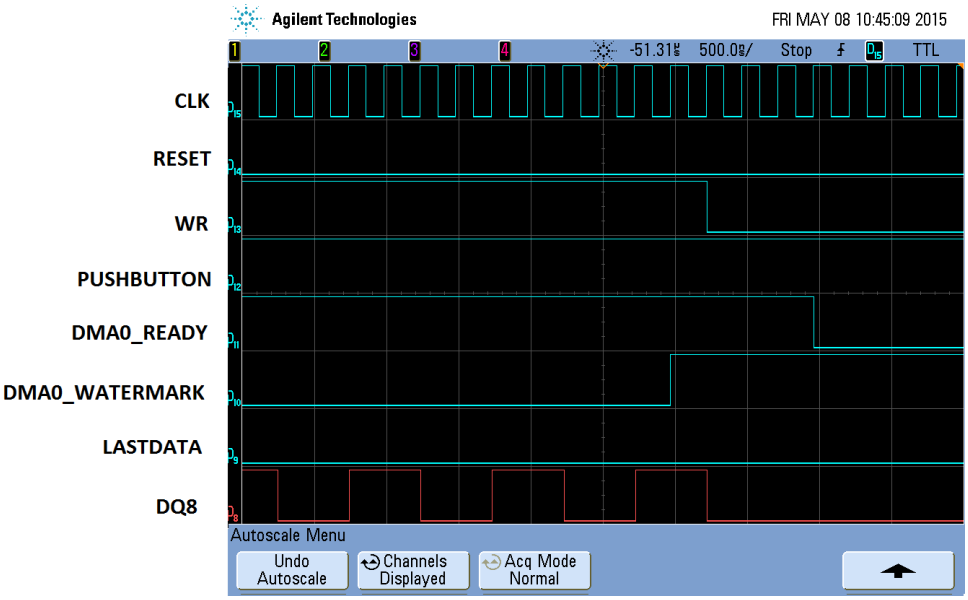

Imagen 37: Señales del experimento 2 II

En el caso contrario que és en el que se está esperando a que se vacíe el buffer del DMA para poder escribir, se esperará hasta el final del contador, en ese momento se verificarán los flags DMA0_Ready y DMA0_Watermark y si DMA0_Ready=1 y DMA0_Watermark=0 se volverá a escribir por lo que WR=1 y se estarán escribiendo datos hasta que se vuelva a recibir otra vez el flag DMA0_Watermark activo.

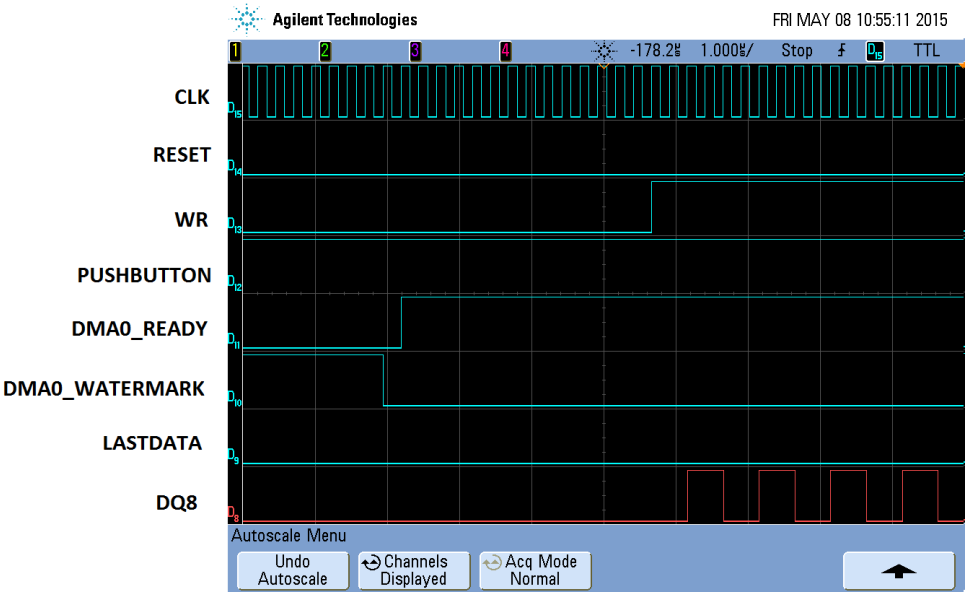

Imagen 38: Señales del experimento 2 III

Una vez se han analizado las señales en el osciloscopio se pasará a un analisis dependiente de la frecuencia en la cual se proporcionará la velocidad de transmisión, el número de datos leídos, el número de errores y la proporción número de errores/número de datos leídos.

- Frecuencia = 1 MHz

A esta frecuencia se tendrá una velocidad de transmisión que será proporcionada por el ejecutable CollectData.exe de:

Imagen 39: Velocidad de transmisión a 1 MHz

Al analizar los datos que han sido enviados durante una hora entre el dispositivo FX3 Super Speed y el ordenador con MATLAB se tendrán los siguientes resultados:

```
>> analisis_final_gpif2
inicio busqueda del principio de secuencia
principio secuencia encontrado
datos comprobados = 4437335989
numero de errores = 0
porcentaje error = 0.000000 >>
```

Imagen 40: Estadísticas de la transmisión a 1 MHz

Si se representara la señal esta tendría la siguiente forma:

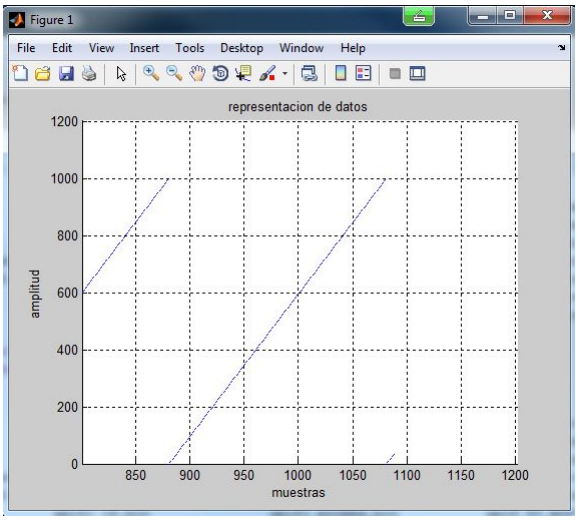

Imagen 41: Representación señal a 1 MHz

Como se puede observar a vista de los resultados se ha llevado a cabo la transmisión sin errores cumpliendo con los requerimientos establecidos para este proyecto, lo que se hará ahora es ver cual es la influencia de la tasa de transmisión sobre la tasa de error analizando archivos de datos que han sido enviados durante 30 minutos.

Tasa de Transmisión (Mbps)	Probabilidad de error (%)
16	0
32	0.001775
64	0.005507
96	0.008015
128	0.008176

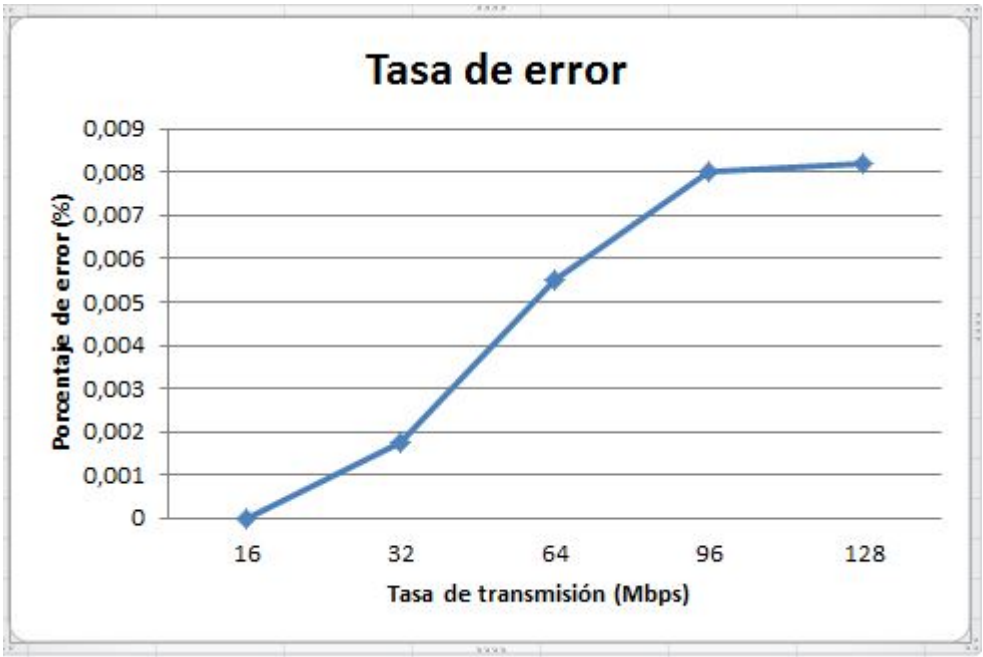

Imagen 42: Representación de la tasa de error respecto a la tasa de transmisión

En el gráfico se puede ver que cuanto mayor sea la tasa de transmisión y en consecuencia la frecuencia de reloj mayor será la probabilidad de error, además cabe destacar los porcentajes de error tán pequeños que se tiene pero intolerables para la aplicación que se busca.

6 CONCLUSIONES

Cuando se inició este proyecto se perseguía el objetivo de transferir las senãles que llegan al monitor AVALON a una tasa de 16 Mbps al ordenador a través del FX3 Super Speed Device y de analizar estas señales para demostrar que la transferencia se produce sin errores.

El objetivo se ha cumplido sin problemas ya que con una frecuencia de 1MHz se ha logrado llevar a cabo una transmisión de datos durante 1 hora transmitiendo 4.437.335.989 Bytes sin errores. Esto ha sido posible gracias a la utilización de los flags DMA0_Ready y DMA0_Watermark que han ido señalando cuando el buffer del DMA estaba disponible para recibir datos y cuando éste se iba a llenar.

Una vez comprobados los dos objetivos principales del proyecto se han llevado a cabo una serie de simulaciones para ver como afecta la tasa de transmisión a la tasa de error y se ha llegado a la conclusión de que cuanto mayor es la tasa de transmisión mayor es el número de datos que se envían en un mismo periodo temporal y a su vez es también mayor la tasa de error.

Se llega a la conclusión, de que el chip FX3 incoporado en el FX3 Super Speed Device es capaz de transportar datos entre una FPGA y el ordenador a través de un USB 3.0 sin errores, por tanto podrá ser utilizado para realizar el depurado de las señales del monitor AVALON en el ordenador, siempre y cuando se utilicen una serie de buffers entre el monitor y el ordenador debido al cuello de botella que se genera en el buffer del DMA.