Capítulo 4

Pruebas de funcionamiento y desempeño del sistema desarrol

para que no te divida el texto en estos casos se puede usar:

Capitulo~3

Unir parrafos

El sistema de comunicación desarrollado fue sometido a una prueba de desempeño a fin de verificar que sea capaz de enviar y recibir datos en forma efectiva y que, a su vez, cumpla con el objetivo de establecer un enlace capaz de enviar datos a una tasa de?

La prueba realizada consistió en el envío de un conjunto de datos desde una PC, que fue almacenado en el FPGA para luego ser retransmitidos hacia la PC, la misma fue repetida en forma automática durante 24 horas.

dicha

Para realizar la prueba fue necesario elaborar un sistema en FPGA, dotado con la capacidad de memoria necesaria para recibir los datos que luego enviará, la MEF presentada en el Capitulo 3 que provee lo necesario para la comunicación con el controlador FX2LP y una señal de reloj que permite sincronizar la MEF y la memoria implementada.

El FPGA fue conectado con la interfaz FX2LP a través de la placa de interconexión de

Esto lo sacaria, si decis que vas a probar el sistema que hiciste se entiende que lo que vas a probar es lo que presentaste antes.

configuración elaborada en el Capítulo 2.

Además, se realizó un programa de computadora, escrito en lenguaje C++, que permite enviar una trama de datos aleatorios hacia el dispositivo desarrollado en el presente trabajo, a través del protocolo USB.

A lo largo de este Capítulo, se describiran los componentes desarrollados para realizar las pruebas de desempeño de la interfaz de sus componentes y el programa de PC elaborado, y se expondrán los resultados obtenidos de las pruebas y las conclusiones del trabajo realizado.

Yo pondria las conclusiones en un capitulo separado

4.1. Sistema de pruebas implementado en FPGA

El sistema desarrollado en el FPGA para realizar las pruebas de desempeño implementa la MEF desarrollada en el Capítulo 3, debido a que es uno de los componentes principales en lo que a las pruebas se refieren. Sin embargo, el sistema también requiere de capacidad de almacenamiento de los datos que son provistos desde la PC y de una señal de reloj.

Para cubrir los requerimientos de capacidad de datos, se utiliza una memoria FIFO. Así, los datos que son enviados desde la PC, se almacenan en la memoria FIFO a medida que arriban, atravesando la interfaz. Luego, estos datos son reenviados desde la misma memoria FIFO, emprendiendo el circuito hacia la PC.

Esto no me queda claro: yo diria que para realizar la prueba se conecto el bloque presentado en el capitulo anterior a una memoria FIFO (que tambien fue desarrollada para esta aplicacion no?) que se utiliza para almacenar los datos aleatorios provenientes de la PC que luego seran reenviados. y tambien presentaria la Figura 4.1 aca.

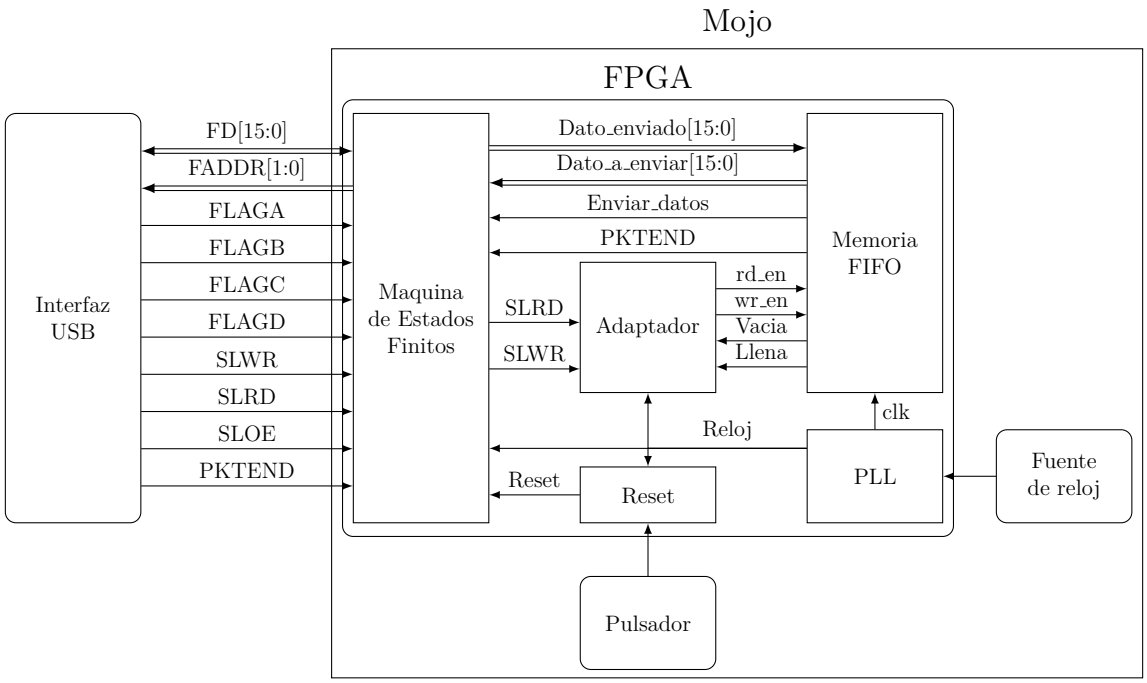


Figura 4.1: Diagrama en bloques del sistema implementado en el FPGA

La señal de reloj del FPGA es provista por un oscilador incorporado en la placa de desarrollo Mojo v3. [illegible] el oscilado

Yo pondria: La señal de clock se obtuvo de una de las salidas del PLL de la placa mojo, este bloque se utiliza para incrementar la frecuencia del clock interno de XX Mhz a la frecuencia utilizada XX Mhz.

Tanto el PLL como la memoria FIFO incorporadas al sistema implementado dentro del FPGA, fueron generadas por la herramienta *Core Generator*, provista por la empresa Xilinx Inc. en el software de programación ISE.

La arquitectura del sistema implementado en la placa de desarrollo Mojo v3 se observa en el diagrama en bloques de la Figura 4.1. Los datos que provienen desde el controlador FX2LP (originados en la PC en forma previa), llegan al FPGA a través de la MEF. Luego, la MEF remite los datos arribados a la memoria FIFO.

Las señales SLRD y SLWR debieron ser adaptadas para compatibilizar el funcionamiento de la memoria generada por la herramienta *Core Generator*.

La señal de reloj emitida por la salida del sincroniza tanto a la MEF como a la memoria FIFO.

Por su parte, la placa Mojo v3 incorpora un pulsador, que es usado en este trabajo como señal de reset asíncrona. El sistema tiene un reset síncrono realizado a través de un contador, que es utilizado para establecer los valores iniciales del circuito.

unir parrafos

4.1.1. yo uniria estas dos secciones emas adicionales

Dos de los módulos implementados en el sistema de pruebas no fueron desarrollados en el presente trabajo, sino que se generaron a través de soluciones que provee la empresa Xilinx Inc. para el desarrollo de sistemas.

Los módulos provistos por Xilinx Inc. fueron el PLL, a través del cual se obtuvo la señal de reloj del sistema, y la memoria FIFO sintetizada en el FPGA. Se explicitan los detalles de la

No recuerdo haber leido que el desarrollo y la implementacion del sistema fue realizada utilizanodo el entorno de desarrollo ISE, si no se aclaro antes estaria bueno hacerlo.

configuración realizada en el marco del trabajo que se presenta en este informe.

El código completo de la descripción realizada en lenguaje VHDL del sistema de pruebas, en donde se declaran e instancias los componentes que se describen en esta sección junto con la MEF desarrollada y las señales necesarias, se puede apreciar en el Apéndice B.

Generación de Señal de Reloj

Las especificaciones de la interfaz indican que la máxima frecuencia de funcionamiento del reloj debe ser de 48 MHz[35]. La placa de desarrollo Mojo V3, por su parte, posee un oscilador que provee al FPGA una señal de 50 MHz. Para lograr la señal de reloj con la frecuencia de 48Mhz se utiliza un PLL incorporado dentro del circuito integrado del FPGA

El PLL fue configurado a través de la herramienta *Core Generator* provista por Xilinx junto con el entorno de desarrollo ISE, utilizado en este trabajo para realizar los codigos en VHDL, yo sacaria esta oracion y dejaria la referencia. ésta última en el FPGA[36]. A través de esta herramienta, se indicó que la señal de entrada es de 50 MHz.

Se aprovechó que el PLL integrado en el FPGA Spartan 6 permite configurar hasta 4 salidas con frecuencias diferentes y se seleccionaron señales de salida con 50, 48, 40 y 35 MHz, para tener una forma de reducir la frecuencia si se presentaban problemas de sincronismo durante la prueba del sistema.

La herramienta *Core Generator* de Xilinx entregó un código de VHDL en donde se declara una entidad para que pueda ser utilizada como componente y se instancia el PLL. Luego, la entidad de dicho código se declaró e instanció en la descripción del sistema de pruebas.

Implementación de la memoria FIFO en el FPGA

Esto no se que es. Se puede aclarar mejor?

La memoria FIFO sintetizada en el FPGA también se implementó a través de la herramienta *Core Generator* de Xilinx. Dicha herramienta permite configurar una memoria de 512 o 1024 B.

Se configuro la memoria con una capacidad de 1024 B, ancho de bus con 16 bits, señal de reconocimiento de escritura y reloj unificado, tanto para el puerto de entrada como el de salida. La memoria cuenta con puertos de entrada y salida independientes.

Con la configuración mencionada, *Core Generator* entregó una plantilla para utilizar la memoria generada. Dicha plantilla se utilizó para declarar e instanciar el componente en el sistema implementado.

Es de destacar que la memoria FIFO generada por la herramienta *Core Generator* no es directamente compatible con la MEF diseñada, por lo que se realizó un pequeño adaptador para que reconozca las señales de lectura y escritura de la MEF. El adaptador está compuesto por dos máquinas de estados que habilitan la escritura y la lectura con los flancos descendentes de las señales *SLRD* y *SLWR*, en función del estado (vacío o lleno) de la memoria.

para que la MEF pueda reconocer las señales....

Se podria detallar un poco mas estos adaptadores? y especificar porque es necesario su utilizacion.La maquina de etados funciona por flancos y los cores no?

4.1.2. Verificación Funcional del sistema de pruebas

Debido a la incorporación de nuevos componentes, como al desarrollo de nuevas señales que adaptan las ya existentes, se realizó una verificación funcional del sistema de pruebas desarrollado.

Para ello, se generó un código de descripción en VHDL, en donde se simula la entrada de datos al sistem Con este objetivo, se estableció un

Yo pondria: debido a la incorporacion de los nuevos bloques empleados para realizar las pruebas y la adaptacion de señales, se realizo una nueva verificacion funcional del sistema implementado en la FPGA.

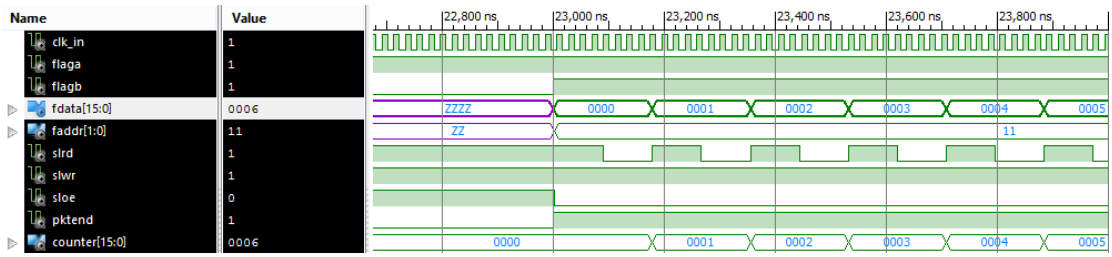


Figura 4.2: Simulación del inicio del ciclo de lectura

no entiendo esta oracion

contador que avanza a medida que encuentra la señal de lectura de la interfaz USB, *SLRD*. También se coloca en bajo la señal *FLAGB*, indicando que la memoria del controlador FX2LP contiene datos, por lo que requiere efectuar sobre ella una operación de lectura.

unir

El código completo realizado para la verificación se puede leer en el Apéndice B.

Una vez que el contador alcanza un nivel aleatorio, se procede a bajar la señal *FLAGB* para indicar que la memoria de entrada de datos se encentra vacía. Resta esperar que el sistema devuelva los datos almacenados a la interfaz USB a través de la operación de escritura.

Se coloca un cero logico en la ¿entrada FLAGB?

La Figura 4.2 muestra el diagrama temporal entregado por el simulador en donde se detalla el inicio de la operación de lectura de la memoria de la interfaz USB. Cuando la señal *FLAGB* se encuentra en alto, el sistema activa la memoria FIFO del controlador FX2LP con dirección ″00″ y habilita también su salida a través de la señal *SLOE*. Luego, procede a leer la memoria colocando en bajo la señal *SLRD*.

Una vez leído el dato, se simula un cambio en la memoria a través del incremento en el contador y se lee el dato siguiente a través del flanco descendente de la señal *SLRD*.

Cuando fueron leídos todos los datos contenidos en el controlador FX2LP, este colocará en bajo la señal *FLAGB*. Con esta señal, el FPGA comenzará el ciclo de escritura, reenviando los datos almacenados en su memoria. En la Figura 4.3 se puede observar que cuando baja la señal, se interrumpe el ciclo de lectura y el sistema vuelve al de reset, para seguidamente iniciar el ciclo de escritura.

quien?

Durante la operación de escritura, el bus *FADDR[1:0]* es colocado en ″11″ indicando la memoria FIFO receptora de los datos. Luego, activa y desactiva, en forma intermitente la señal *SLWR*, enviando los datos almacenados en la memoria del FPGA hacia la memoria de la interfaz.

direccion de la memoria FIFO

Si por algún motivo la memoria de la interfaz USB se queda sin más espacio para el almacenamiento de los datos que provienen del FPGA, el controlador FX2LP coloca en bajo la señal *FLAGA*. En la Figura 4.4 se observa que luego del flanco negativo de la señal... el FPGA interrumpe el envío. Una vez que *FLAGA* es nuevamente alta, la memoria sigue enviando los datos almacenados.

Se puede observar que cuando la señal FLAGA pasa a un nivel alto la memoria .....

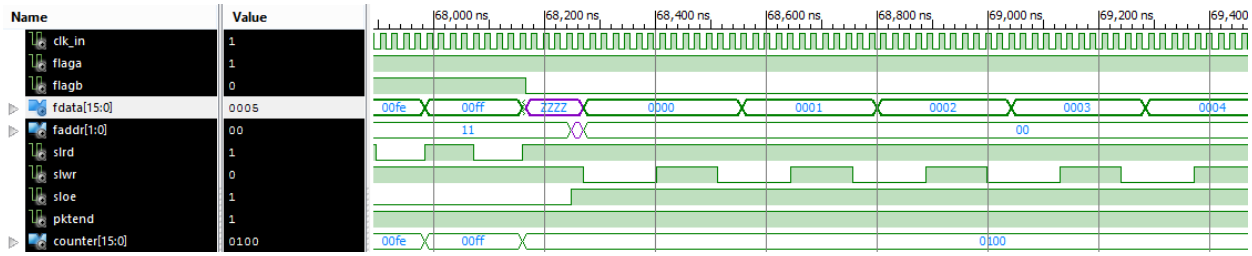


Figura 4.3: Simulación del inicio del ciclo de escritura

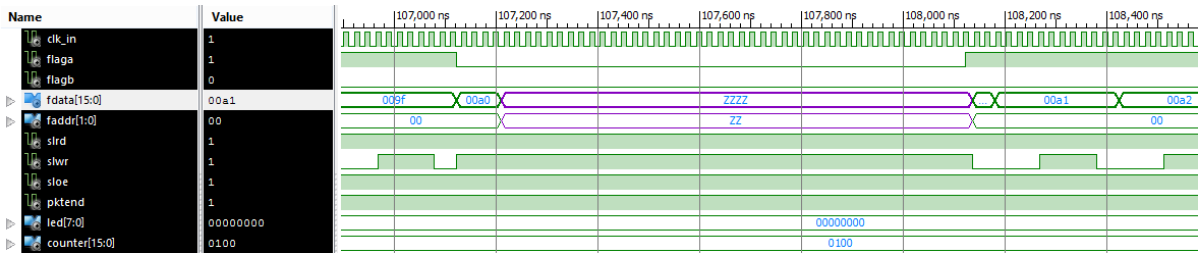


Figura 4.4: Simulación de una interrupción en el ciclo de escritura por falta de espacio en la memoria de la interfaz USB

Yo pondria: Cuando la memoria interna de FPGA no posee mas datos para transmitir se activa la señal PK....

Una vez que la memoria interna del FPGA se quedó sin datos, activa la señal PKTEND, indicando que terminó el envío de datos. Esto se puede observar en la Figura 4.5. En ella se sabe que los datos de la memoria se acabaron debido a que el bus de datos alcanzó el mismo valor que el contador utilizado para emular el envío de datos desde la interfaz FX2LP.

Luego de realizada la simulación y verificado que el funcionamiento del sistema es el esperado, se está en condiciones de cargar la síntesis del circuito en el FPGA.

Yo redactaria esto mejor

4.1.3. Carga del sistema de pruebas en el FPGA

Unir parrafos

Para efectuar la carga del sistema fue necesario especificar en el entorno de desarrollo ISE provisto por la compañía Xilinx Inc, en qué pines del FPGA debe asignar cada uno de los puertos descriptos en el código del sistema.

Para indicarle al programa ISE dónde debe colocar los puertos se escribe un archivo denominado *User Constraints File* (que significa Archivo de Restricciones del Usuario) y lleva como extensión la sigla *ucf*.

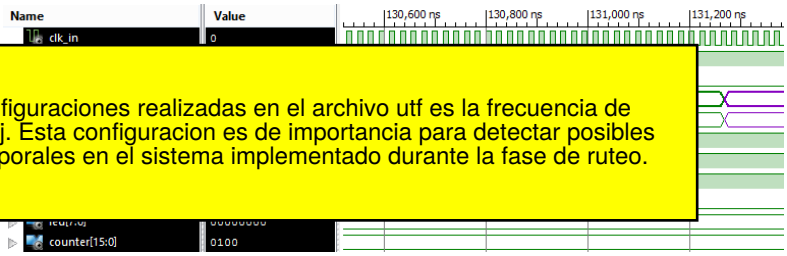
El archivo *ufc* se generó a través de una interfaz gráfica provista por Xilinx Inc., denominada *Plan Ahead*. En ella se puede cargar en una planilla la equivalencia entre los pines y los puertos, como así también los niveles de tensión lógicos necesarios.

En la planilla se cargaron los puertos como se indican en la Tabla 3.2. ~~Con respecto a los~~

, los miosmos fueron configurados utilizando el estadar LVTTL que es el utilizado tanto por la placa Mojo V3 como por el controlador FX2LP [].

También se especificó en el archivo *ucf* la frecuencia de la entrada de reloj. Con esto, a la hora de generar el ruteo del FPGA, puede hacer un chequeo e informar si existe algún problema temporal en la implementación.

Finalmente, se ejecutó el compilador para ejecutar el archivo que fue cargado en la memoria



Yo pondria:

Otra de las configuraciones realizadas en el archivo utf es la frecuencia de entrada de reloj. Esta configuracion es de importancia para detectar posibles problemas temporales en el sistema implementado durante la fase de ruteo.

Figura 4.5: Final de la simulación

Flash de la placa de desarrollo Mojo v3 constatando que no existían errores.

El texto completo del archivo *ucf* generado, se observa en el Apéndice B.

4.2. Desarrollo del programa de PC para enviar y recibir datos

La prueba del sistema consintió en el envío de una trama de datos desde la PC. La trama debe arribar al FPGA, atravesando la interfaz FX2LP. El FPGA, por su parte, almacena la trama de datos y, una vez finalizada la recepción, procede a la transmisión de la trama en el camino de regreso.

lenguaje

Se generó un programa de computadoras, escrito en C++, con el objetivo de poder crear tramas con datos aleatorios y repetir la prueba en forma automática.

Se eviaron datos a la sistema desarrollado ininterrumpidamente durante 24 horas con el objetivo de probar la robustez en la comunicación y calcular la tasa de datos transmitida por el bus.

Los datos fueron generados en forma aleatoria para que el ruido no sea enmascarado por una trama invariable en el bus de datos.

Para elaborar el programa que se encarga de enviar y recibir los datos se utilizó la biblioteca `libusb-1.0`.

En esta Sección se justificará la elección de la biblioteca `libusb-1.0` y se detallará la secuencia del programa desarrollado.

El código completo con la implementación del programa se encuentra en el Apéndice C

esto no lo entiendo, se puede explicar con mas detalle?

4.2.1. Elección de la biblioteca libusb-1.0

La elaboración de un programa en lenguaje C++ que pueda controlar en forma adecuada los puerto USB de una PC es una tarea que escapa a los objetivos de este trabajo. En su lugar, se recurrió al uso de una biblioteca ya definida y bien conocida, que permita obtener acceso a los puertos USB, y con ella, poder realizar el envío y la recepción de datos necesaria para las pruebas del sistema desarrollado.

Se eligió la biblioteca `libusb-1.0` para la realización del programa debido a que es una biblioteca de código abierto, es decir, que sus archivos fuente pueden ser leídos, modificados y/o utilizados por cualquier persona sin la necesidad de pagar una licencia.

Otro motivo para su adopción es su característica de multiplataform que permite realizar codigos que funcionen...

e funcione en sistemas operativos tan diversos como Windows, Linux, Mac Os, Android, entre otros. Lo que no puede ser logrado con el uso de bibliotecas privativas como WinUSB.

Adicionalmente, la biblioteca `libusb-1.0` no tiene un autor específico, sino que existe una gran comunidad que contribuye al crecimiento del proyecto, como así también otros proyectos que utilizan esta biblioteca. Así, existe una gran variedad de ejemplos y foros que facilitan el aprendizaje en su utilización y adaptaciones para diferentes lenguajes de programación, que se adapte a los conocimientos previos de la persona que desarrolla programas.

Yo escribiria distinto esto, me parece que no esta bueno decir que escapa a los objetivos del trabajo y por eso no lo hiciste, yo resaltaria que te tomaste el trabajo de elegirla y que la usaste porque es conocida como decis despues, por lo que pondria algo asi:

Existen librerias que son frecuentemente utilizadas para controlar diversos perifericos mediante el lenguaje C++, una de estas librerias es libusb que permite controlar puertos USB. Esta libreria ya esta bien definida y es bien conocida, permite acceder a los puertos USB y poder realizar.....

				Paridad columna
Fila 1	0	0	1	1
Fila 2	1	1	0	0
Fila 3	0	1	0	1
Paridad fila	1	0	1	0

Figura 4.6: Esquema de paridad par bidimensional de 4x4 bits.

4.2.2. Programa de PC desarrollado

El programa de PC desarrollado para el envío y recepción de datos desde la PC hasta el FPGA consta de tres modulos? La primera de ellos se encarga de inicializar la biblioteca, identificar que el dispositivo USB conectado se corresponda con la comunicación con el FPGA y solicitar al sistema operativo el acceso a la comunicación. El segundo módulo se encarga de generar la trama de datos, enviarla hacia el FPGA y esperar su recepción. La trama de datos generados es almacenada para la tercera etapa del programa que, una vez recibida la transferencia desde el FPGA, corrobora que los datos recibidos en la PC sean iguales a los enviados.

por el tercer modulo?

en el primer modulo del programa se inicializa...

Durante la primera etapa del programa, se inicializa la biblioteca `libusb-1.0` y se identifica el dispositivo. Para ello es necesario generar una lista con todos los dispositivos USB conectados. La lista de los dispositivos es informada por el sistema operativo a través de los identificadores contenidos en los descriptores del dispositivo. Si en la lista de dispositivos se encuentra el sistema desarrollado, se solicita acceso al sistema operativo. En caso contrario, el programa finaliza informando la situación.

En

Durante el segundo módulo del programa, se generan datos en forma aleatoria. Con el objetivo de obtener información adicional en caso de que existan errores en la información transmitida, se agregan bits de paridad par bidimensional (ver Figura 4.6), en tramas de 8x8 bits. Es decir, se genera un número aleatorio de 7 bits. Luego, se agrega un bit adicional, de forma tal que la cantidad de unos existentes en la palabra enviada sea par. Finalmente, se calcula la paridad de los bits con igual significancia en grupos de 7 números y se agrega el número resultante en el octavo lugar del grupo. Una vez generados los datos, estos son transmitidos hacia el FPGA. Los datos generados suman 128 B para cada una de las transferencias realizadas.

En la tercera etapa, el programa espera hasta la recepción del mensaje reenviado. Una vez que este arriba, se procede a corroborar que los datos recibido sean iguales a los enviados. La función encargada de verificar los datos corrobora que los bits de paridad sean correctos y, en caso de haber un error, informa cuál fue el dato erróneo.

Tanto los datos enviados como los recibidos son almacenados en un archivo mientras el programa es ejecutado para un análisis posterior de los resultados.

El código completo del programa desarrollado, escrito en C++, se puede observar en el Apéndice C

No tengo esta foto!!! ahora tengo que hacerme un escape al CAB para sacarla!

Figura 4.7: El sistema desarrollado en funcionamiento

4.3. Pruebas de la comunicación entre el FPGA y la PC

El sistema completo, montado y en funcionamiento, se muestra en la Figura 4.7. En él se puede apreciar el FPGA conectado a la interfaz USB través de la Placa de Interconexión. A su vez, la interfaz USB se enlaza con la PC a través de un cable. La conexión entre la interfaz USB con la PC sirve no solo para transferir los datos que llegarán el FPGA, sino también el programa que ejecutará el controlador FX2LP.

Además, el FPGA también se conecta a la PC a través de un cable con el propósito de transferirle el archivo de programación y de proveerle alimentación a través del puerto USB.

Con los dispositivos dispuestos en la configuración descripta, se procedió a cargar los diferentes programas elaborados para cada uno de ellos y, finalmente, se ejecutó el programa de pruebas.

El programa de pruebas fue ejecutado por más de 24 horas con el objetivo de tener una buena cantidad de datos estadísticos como para probar la robustez del sistema, como así también su tasa de transferencia.

Todas las transferencias, tanto de entrada como de salida a la PC fueron guardas en un archivo de registro, documentando la fecha, el sentido de la comunicación y los datos intercambiados.

4.4. Resultados

esto ya lo dijiste varias veces yo sacaria esta oracion

La prueba de sistema envió y recibió mensajes con datos aleatorios durante más de 24 horas. A través de un análisis del registro de los datos enviados y recibidos se pudo determinar con mayor precisión cuanta información fue transmitida y durante que intervalo de tiempo.

El registro dió cuenta de que el sistema estuvo funcionando durante 87134 segundo, lo que equivale a 24 horas, 12 minutos y 14 segundo. durante este lapso de tiempo fueron enviados 388.191.289 paquetes. Se recibió la misma cantidad de paquetes sin pérdida de información ni errores en la transmisión.

El programa de PC desarrollado genera paquetes de 128 B. Se debe considerar que cada paquete contiene un encabezado y una cola que también se transmite junto a los datos. Los valores típicos de encabezado y cola para transferencias en masa y para transferencias isócronas, que es el tipo de transferencia que llega a la PC desde la interfaz USB, este valor es de 38 B[27]. Por tanto, por cada una de los ciclos de envío y recepción de datos, se transfirieron 349 B:

esto no lo entiendo el overhead no es el mismo en ambas direcciones?

$$128B \times 2 + 55B + 38B = 349B$$

Multiplicando el valor de datos transmitidos, por la cantidad de veces que la operación fue realizada, otorga la cantidad de bytes totales enviados. Esto es:

\cdot

$$388x10^3 \times 349B = 135,5 \times 10^6 B$$

\times

388\,\times10 3\cdot349\,B\,=

Cada Byte contiene 8 bits, por lo que al efectuar la operación de conversión se tiene que:

$$135,5 \times 10^5 B \times 8bit/B = 1,08 \times 10^12bit$$

aca formatea la ecuacion como te dije en el comentario anterior y pone la unidad bit/B entre parentesis, pone entre llaves el 12 para que te queden los dos numeros como superindices

NO DIGAS ESO!!!! Para que hiciste la tesis si la tasa parece insuficiente?

Finalmente, la tasa de bit se obtiene al dividir los bits transmitidos por la cantidad de tiempo total empleada en la prueba. Lo que arroja:

$$\frac{1,08 \times 10^{12} bit}{87,1 \times 10^{3} s} = 12,4 \times 10^{6} bit/s$$

Por tanto, la tasa de bit lograda en la transmisión USB fue de $12.4\,\mathrm{Mbit\,s^{-1}}$.

La tasa de transmisión podría parecer insuficiente a los efectos de trabajo. Incluso se puede suponer que la tasa de transmisión es más similar a un sistema USB de velocidad completa que a uno de alta velocidad.

Sin embargo, se considera que si el sistema USB de la PC en donde se realizó la prueba funcionase a velocidad completa, significaría que el total del ancho de banda estuvo dedicado al sistema. No obstante, el bus USB tenia conectado también un ratón y un teclado, y ambos funcionaron con normalidad durante toda la prueba, por lo que no puede decirse que el sistema USB estuviese dedicado al dispositivo desarrollado en forma exclusiva. Esto quiere decir que el sistema efectivamente funcionaba en modo de alta velocidad, aunque dedicaba un ancho de banda muy bajo a la comunicación entre la PC y el dispositivo desarrollado en este trabajo.

Se podría explicar el bajo rendimiento a que el programa de pruebas se detiene a la espera del paquete entrante cuando este es solicitado. Esto hace que, si bien la prueba desarrollada sirve para testear la confiabilidad del sistema, logrando una tasa de errores de bit baja (se recibieron más de 10^{12} bit $\mathrm{s^{-1}}$ sin errores), no es la óptima para probar el ancho de banda máximo posible. Sin embargo, dicha prueba escapa al tiempo disponible para seguir profundizando las pruebas de este trabajo.

Hay que reescribir esto, no se puede terminar asi

Sumario del capitulo

4.5. Conclusiones

En el transcurso del trabajo reportado por este informe fue cumplido el objetivo general, elcual consistió en el desarrollo de un sistema de comunicación USB 2.0 de alta velocidad destinado al intercambio de datos entre una PC y un FPGA.

Además de cumplimentar con el objetivo general perseguido por este trabajo, se logró entender conceptos fundamentales del funcionamiento del USB, tal como el empaquetamiento de lo datos y el tipo de transferencias que pueden realizarse a través de él. También se logró comprender cómo debe ser descripto un dispositivo USB al ser desarrollado y cómo debe ser informado a la PC.

El sistema de comunicación implementado se compone de un software de computadora, una interfaz USB y un FPGA.

Se utilizó el controlador FX2LP, comercializado por la empresa Cypress Semiconductor como interfaz USB. En el transcurso de este trabajo se pudo comporender su aruqitectura y funcionamientos. ¿a través de su estudio se pudo configurar dicho dispositivo, obteniendo un funcionamiento acorde a los requerimientos. El controlador FX2LP recibe transferencias en masa y transmite transferencias isócronas desde y hacia la PC respectivamente. Con el FPGA se comunica por un bus de 16 bits a través de una memoria FIFO comandada por el FPGA.

Se utilizó un FPGA Spartan 6 comercializado por la empresa Xilinx para implementar, en lenguaje VHDL, una Máquina de Estados Finitos capaz de enviar y recibir datos desde la interfaz USB. Dicha máquina de estados es utilizada para comandar la memoria FIFO presente en el controlador FX2LP. Se destaca de esta máquina de estados el bajo consumo de recursos

programables de FPGA, dejando lugar a la implementación de aplicaciones que utilicen la el sistema de comunicacion desarrollado

También se elaboró un circuito impreso destinado a la conexión física entre el FPGA y la interfaz USB.

Se desarrolló un software de computadoras que genera, envía y recibe datos hacia y desde el FPGA. Para la elaboración de este programa, se utilizó la biblioteca `libusb-1.0`, que permite la comunicación de programas con dispositivos conectados a través del bus USB. Esta es una biblioteca de código abierto y que puede ser ejecutado en cualquier sistema operativo.

Se implementó a su vez un sistema de pruebas compuesto de una memoria FIFO implementada en el FPGA, que recibe mensajes desde la interfaz USB y los retransmite de vuelta. Este sistema permitió testear la funcionalidad y robustez de la comunicación desarrollada.

El sistema desarrollado fue probado y logró una conexión efectiva entre una PC y un FPGA, logrando en la prueba un intercambio de mas de 1×10^{12} bits sin pérdida de información. La tasa de transferencia lograda por la prueba de comunicación fue de $12.4\,\mathrm{Mbit\,s^{-1}}$.

4.6. Consideraciones Finales

Durante el transcurso del presente trabajo se logro establecer una comunicación entre una F
No digas eso, no se puede finalizar una tesis con esta conclusion.
r

Se plantea para un trabajo futuro la posibilidad de implementar una nueva prueba de funcionamiento a través del envío ininterrumpido de tramas conocidas de datos, generados en forma local desde el FPGA. Esto permitirá conocer en mayor detalle la tasa máxima de bit lograda con la configuración desarrollada en este trabajo.

Otra mejora a implementar en el sistema desarrollado consta de la realización sincrónica de la comunicación entre el FPGA y la Interfaz USB. Para ello, es necesario reconfigurar el controlador FX2LP, indicando que el funcionamiento de la memoria FIFO será de modo sincrónico. A su vez se debe rediseñar la placa de interconexión de forma tal que tanto el controlador FX2LP como el FPGA puedan compartir el reloj. Se espera que esta mejora brinde mayor velocidad al sistema, como así también se releva al FPGA de utilizar un PLL para brindar la señal de reloj necesaria, liberando recursos para su utilización en otro tipo de desarrollos implementados dentro de este.

Yo diria que:

- La interfaz no tuvo errores y que cumple con la norma XXX....
- Se ocupo muy poco espacion en la FPGA por lo que queda mucho espacio para el desarrollo de otras aplicaciones.
- El ancho de banda fue de 12Mbps y que posiblemente sea porque el bus de la PC en la que lo probaste estaba ocupado con otras cosas.
- Hablar de trabajos futuros.
- Decir que esta interfaz puede ser utilizada en aplicaciones cientificas en las que se necesita transmitir grandes cantidades de datos desde una FPGA a una PC como detectores de particulas basados en sensores de imagen CMOS, euipos para la obtencion de radiografias, y neutrografias.