Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica

Implementación de una comunicación entre un computador y un FPGA en tiempo real, por puerto USB .

1. Resumen

En este documento se presenta una guía para implementar una comunicación por el puerto USB entre un computador y una FPGA. En particular se desarrolla una aplicación para implementar la comunicación entre una tarjeta de desarrollo Nexys 2 y un computador con sistema operativo Windows. La base de este sistema radica en la emulación de un puerto pararlelo, lo cual se lleva a cabo por medio de un microcontrolador Cypress CY7C68013A en la tarjeta, el cual se encarga de controlar la comunicación.

Para motivar el trabajo presentado se desarrolla una pequeña aplicación en la FPGA la cual permite almacenar datos provenientes desde el computador en una memoria y acceder a ellos desde los periféricos disponibles en la tarjeta. [1]

2. Introducción

Al momento de desarrollar proyectos que involucren el trabajo con una FPGA, frecuentemente nos vemos enfrentados al problema de generar una comunicación en tiempo real con el computador, ya sea con el fin de poder *debuguear*, cambiar o configurar registros *on line*, entre otros. Para solucionar este problema existen diversos métodos que implementan distintos protocolos de comunicación. Sin embargo, en la actualidad, el protocolo USB se ha vuelto casi un estándar, siendo utilizado por un sin número de dispositivos. De hecho es muy probable que la elección de cualquier otro protocolo implique tener que adquirir un dispositivo que se encargue de traducir desde dicho protocolo a USB. Es por esta razón que este documento presenta una guía, en la cual se expone una implementación de una comunicación entre un computador y un puerto USB. Esta implementación

[1] Esta aplicación se encuentra basada en el proyecto entregado por el fabricante disponible en https://reference.digilentinc.com/digilent_adept_2

fue desarrollada para un computador con sistema operativo Windows y una tarjeta de desarrollo para FPGA Nexys 2. Sin embargo la metodología para otras tecnologías es bastante similar a la propuesta.

3. Comunicación USB

La comunicación por USB es un protocolo del tipo maestro/esclavo, en donde el maestro es el computador o host, mientras que el dispositivo o periférico implementa el esclavo. De este modo, para poder realizar una correcta comunicación es necesario contar con inteligencia correcta tanto en el maestro como en el esclavo. Por otra parte, en el maestro es necesario contar con los controladores del esclavo, mientras que en el esclavo es necesario contar con algún tipo de interprete de las señales provenientes desde el maestro. En nuestro caso particular, los drivers necesarios están disponibles en la página del distribuidor en el SDK dpcutil.dll [2], la cual implementa todas las funciones de bajo nivel para poder realizar la comunicación. En el esclavo, que en esta caso representa nuestra tarjeta Nexys 2, contamos con el integrado CYPRESS 68001A, el cual se encarga de interpretar el protocolo USB y emula un puerto paralelo (protocolo EPP[3]) disponible para ser utilizado por la FPGA. De este modo, es como si la FPGA estuviera conectada a un puerto paralelo.

En resumen la tarea a realizar se compone de dos partes, la primera consiste en crear un código que se ejecute desde el computador, el cual utilice las funciones entregadas por el fabricante, que estan disponibles en la biblioteca dpcutil.dll y por otro lado crear una aplicación en la FPGA que sea capaz manejar una comunicación con el protocolo EPP.

3.1. protocolo EPP

El protocolo de puerto paralelo consiste físicamente en un bus bi-direccional de 8 *bits* el cual sirve tanto para enviar como recibir datos, más un set de 4 señales que coordinan la dirección del flujo de datos.

Señal	Detalle	Dirección
DB [7:0]	Bus de datos	Bidirecciónal
WRITE	Control de flujo de escritura	Desde el puerto paralelo emulado hacia la FPGA
ASTB	Strobe de dirección	Desde el puerto paralelo emulado hacia la FPGA
DSTB	Strobe de datos	Desde el puerto paralelo emulado hacia la FPGA
WAIT	Estado del canal	Desde la FPGA hacia el puerto paralelo emulado

Para poder implementar una interfaz en la FPGA que se comunique con el puerto paralelo es necesario contar con dos registros de 8 bits, uno de dirección y uno de datos. El registro de dirección almacena la dirección de destino para el dato a transferir. De este modo existen cuatro operaciones posibles, las cuales se muestran en las imágenes 1,2,3 y 4.

La señal WRITE indica si la operación es de escritura(low) o de lectura(high). Las señales ASTB y DSTB indican si la operación es de dirección o de datos, solo una de estas dos puede estar

[2]todos los archivos necesarios están disponibles en https//:reference.digilentinc.com/digilent_adept_2

[3]para más información revisar https://reference.digilentinc.com/_media/dpimref_programmers_manual.pdf

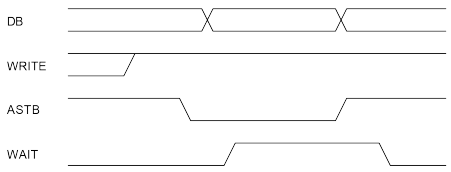

Figura 1: Lectura de dirección.

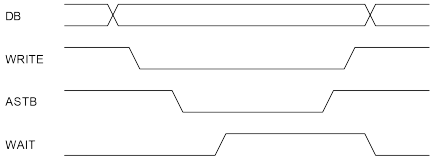

Figura 2: Escritura de dirección.

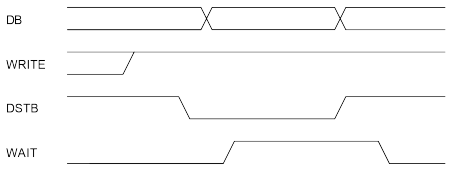

Figura 3: Lectura de dato.

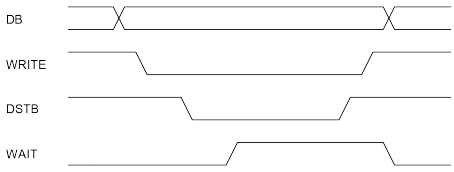

Figura 4: Escritura de dato.

activa(low) en todo momento. Por ultimo, la señal WAIT sirve para evitar colisiones e indica por parte de la FPGA que la aplicación esta ocupada en alguna operación.

Con estas cuatro operaciones se implementan funciones de más alto nivel, de las cuales a nosotros nos interesan las de escritura y lectura de registros. La función de escritura se compone de una escritura de dirección seguido de una escritura de dato. Por otro lado la función de lectura se compone de una escritura de dirección seguido de una lectura de dato. Estas funciones son llevadas

a cabo por el host nuestra tarea es implementar un driver en la FPGA que permita encargarse de estas desde el lado de la aplicación.

4. Detalle de la aplicación

La figura 5 entrega un diagrama del sistema a implementar para un caso general. En el sistema presentado en esta guía, la aplicación en la FPGA consiste en una memoria de dos puertos. Uno de estos puertos, el puerto A, quedará a disposición del host, mientras que el puerto B quedara a disposición del usuario por medio de los periféricos disponibles en la tarjeta.

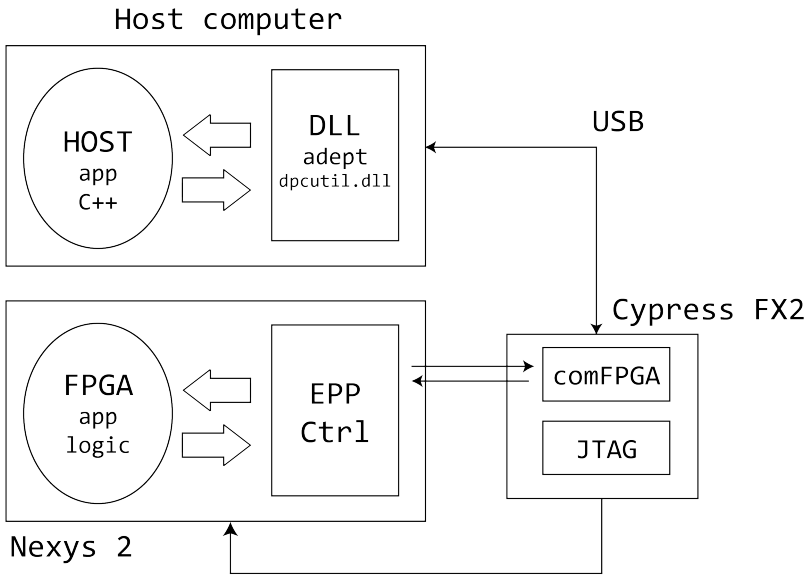

Figura 5: Diagrama del sistema implementado. En este esquema el master es implementado por el host y el esclavo por la Nexys 2

En las siguientes secciones explicaremos como implementar este sistema. En la sección 5 se detalla el esclavo, el cual esta compuesto por el driver de la comunicación EPP[4] y la aplicación en la FPGA. Por otro lado, en la sección 6 se detalla la implementación del master, el cual esta compuesto por una aplicación en C++ en el computador, la cual se encarga de llamar a los drivers y controlar la comunicación con la tarjeta.

5. Implementación del esclavo

La figura 6 presenta el diagrama de los bloques a implementar en la FPGA. Esta implementación cuenta con tres módulos, el primero solo es un driver para recibir las señales provenientes desde el host a través del emulador de puerto paralelo. El segundo módulo, BramComCtrl se encarga

[4] Controlador capaz de llevar a cabo la comunicación entre el emulador de puerto paralelo y la aplicación en la FPGA.

de interpretar estas señales y controlar la comunicación con la memoria, y por último tenemos la memoria de dos puertos.

EppCtrl Este módulo se encarga de lidiar con el tráfico de datos desde el puerto paralelo emulado hacia el resto de la aplicación. La comunicación esta controlada por tres señales **Write**, **Astb** y **Dstb**. Según estas señales el módulo se encarga de actualizar la dirección actual de la aplicación, actualizar los datos de entrada a la aplicación y actualizar los datos de salida hacia el emulador del puerto paralelo.

BramComCtrl A partir de las señales generadas en el módulo EppCtrl genera las señales necesarias para escribir o leer en la memoria. Este modulo es síncrono para evitar latchs.

BRAMAB Este bloque implementa una memoria de dos puertos. El puerto A es dedicado a la comunicación desde el host. Por otro lado, el puerto B queda a libre disposición para cualquier aplicación en la FPGA. En esta oportunidad simplemente podra ser accedida por medio de los switch y la salida del puerto será observable por los LEDS.

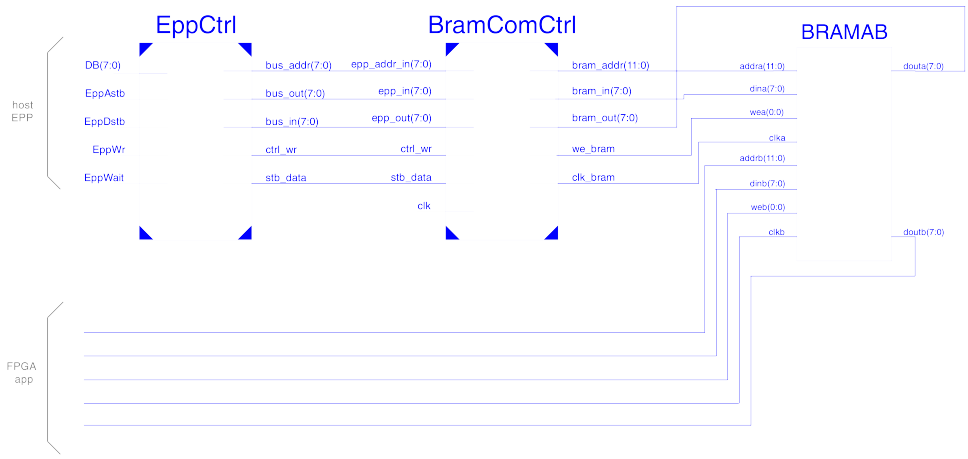

Figura 6: Diagrama de los bloques implementados en la FPGA.

6. Implementación del maestro

Para poder implementar correctamente el maestro, necesitamos desarrollar un programa que sea capaz de manipular la biblioteca de funciones anteriormente mencionada. No existe una restricción sobre el lenguaje en el cual implementar dicho programa. En esta guía la aplicación fue desarrollada en $C++$ por medio del IDE Microsoft Visual 2010[5], en el cual crearemos una aplicación para consola. El programa implementado utiliza básicamente 6 funciones entregadas por el fabricante[6] las cuales se detallan a continuación

[5]IDE descargable de forma gratuita desde el sitio https://www.microsoft.com/es-cl/download
[6]disponibles en https://reference.digilentinc.com/_media/dpcutil_programmers_reference_manual.pdf

Funcion	Salida	Entrada	Descripción
DpcInit()	bool	Puntero a ERC	iniciliza los DLL. Debe ser llamada antes de cualquier otra función de la aplicación
DpcOpenData()	bool	▪ Puntero para manipular la interfaz de datos. ▪ Nombre del interfaz utilizad. ▪ Puntero a un ERC. ▪ Null[7]	Abre una interfaz de comunicación, es necesaria para utilizar cualquier otra de las funciones de comunicación
DpcCloseData()	bool	Puntero para manipular la interfaz de datos	Libera la interfaz de comunicación especificada y cierra el módulo de comunicación
DpcTerm()	ninguno	ninguno	Termina el uso de los DLL cuando la aplicación es finalizada.
DpcPutReg()	bool	Puntero para manipular la interfaz de datos, Variable BYTE con el registro dirección, Variable con el dato a enviar, puntero a un error, Null[8]	Envía el dato dado a la dirección especificada.
DpcPutRegRepeat	bool	Puntero para manipular la interfaz de datos, Variable BYTE con el registro dirección, Puntero a un conjunto de Byte de datos a enviar, puntero a un error, Null[9]	Envía un conjunto de datos a una única dirección

6.1. Incluir *headers* y bibliotecas

Para poder llamar correctamente a las funciones entregadas por el fabricante, debemos incluir los headers y las bibliotecas disponibles en el SDK [10]. Los headers que utilizaremos en nuestro programa son

- dpcdec1.h
- depp.h
- dmgr.h
- dpcutil.h
- dpcdefs.h

Estos deben agregarse a nuestro directorio de trabajo del proyecto y posteriormente agregarlos al programa utilizando `add/from existing item` desde nuestra carpeta de `headers file`.

Luego debemos agregar las bibliotecas a nuestro programa, para esto debemos copiar la carpeta `lib`, disponible en los SDK del fabricante, a nuestro directorio de trabajo. Una vez hecho esto, debemos ir a `Proyect/Properties` buscar la opción `Configuration Properties/Linker` y agregar la carpeta `lib` en la línea `Additional Library Directories`. Finalmente dentro de la misma pantalla de `Configuration Properties/Linker` debemos ir la opción `Input` y en la línea `Additional Dependencies` debemos agregar las siguientes bibliotecas.

- dpcomm.lib
- depp.lib
- dmgr.lib
- dpcutil.lib

[10] https://reference.digilentinc.com/digilent_adept_2