Implementación en Hardware de un Entrelazador/Desentrelazador de Datos con Interface AXI Stream

Laura García Luciano, Ian Camacho Pichardo, Salvador Ibarra Delgado, Jorge Flores Troncoso, Remberto Sandoval Árechiga

Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica.
Av. López Velarde 801, Col. Centro, Zacatecas, Zac., México, 98000.
35162643@uaz.edu.mx, ian.cam57@gmail.com,sibarra@uaz.edu.mx,jflorest@uaz.edu.mx,rem.sandoval@gmail.com.

2016 Published by $DIFU_{100}ci@$ http://difu100cia.uaz.edu.mx

Resumen

En éste artículo se muestra la implementación de un IP Core de un algoritmo de dispersión de errores llamado "Entrelazador", y de su decodificador llamado "Desentrelazador". Un IP Core (Núcleo de Propiedad Intelectual) es un módulo de hardware de aplicación específica, el cual es encapsulado para ser usado como dispositivo en algún sistema que lo necesite. Específicamente hablando, éste entrelazador/desentrelazador como unidad lógica es totalmente funcional y rehusable, pues al estar empaquetado puede facilmente ser colocado en cualquier sistema que lo requiera. En un sistema de comunicaciones que consta de un transmisor y un receptor, existen distintas fases por las cuales viajan los mensajes, cada uno dividido en tramas. El algoritmo estudiado dispersa los errores en el mensaje para su fácil supresión en etapas posteriores. Éste IP Core convive con otros IP Cores que tienen distintas funciones en un sistema de comunicaciones, por lo que la salida del entrelazado es la entrada que alimenta otro proceso, o dicho de otro modo, a otro IP Core. Entonces, es necesario contar con un modo estándar que hará que se comunique el IP Core de Entrelazado/Desentrelazado con la cadena del transmisor y receptor respectivamente. Razón por la cual se hace uso del protocolo AXI, permitiendo la intercomunicación con el IP Core que esté antes en la cadena y con aquel que esté después de la cadena.

Palabras clave: IP Core, Entrelazado, Desentrelazado, FPGA, AXI Stream.

1. Introducción

En un sistema de comunicaciones que se compone básicamente de un transmisor, un canal y un receptor, existen módulos específicos que tratan la información desde su origen analógico/digital hasta ser transmitida por un canal y despues al ser recibida existen también módulos que corresponden a los procesos inversos según corresponda y obtener el mensaje original, o la información original analógica/digital.

Casi todo el tiempo las perturbaciones están presentes, el ruido por ejemplo, y dependiendo de algunas condiciones, aveces se presentan ráfagas de errores. Para ésto, hay una etapa en el transmisor que se encarga de

disipar los errores en ráfagas llamada ***"Entrelazado"*** y así, otro módulo se encargue de corregir los errores de forma eficiente.

El algoritmo de entrelazado en términos generales, revuelve o coloca en posiciones distintas conjuntos de bits de tamaño previamente establecido, y es por ello que los errores quedan separados entre sí. Entonces, el entrelazado del mensaje antes de la transmisión y el desentrelazado después de la recepción causa que una ráfaga de errores sea dispersada y se convierta en errores aleatorios como lo muestran en [1] .

Existen dos tipos de algoritmos de entrelazado, uno es el entrelazado de bloque y el otro es el entrelazado convolucional. Para [2], un entrelazador convolucional es en la mayoría de casos, implementado en hardware por su gran popularidad y simpleza, en [3] se muestra un entrelazador convolucional que se usa para el análisis de modelos.

Un *"Entrelazado de bloques"* acepta símbolos codificados en bloques, los permuta y su salida alimenta al módulo que le sigue en la cadena que conforma un transmisor, el modulador. En el receptor, el *"Desentrelazador de bloques"* realiza la operación inversa, es decir, acepta los símbolos desde el demodulador, los procesa o reacomoda y alimenta a los módulos que le siguen para obtener el mensaje original.

Un aspecto muy importante en el desarrollo del proyecto, es el protocolo AXI que se usa para poder comunicar el entrelazador/desentrelazador con la memoria de la tarjeta de desarrollo y así poder comprobar su funcionamiento. El protocolo AXI Stream es utilizado para intercomunicar de froma rápida y eficiente la memoria externa por medio del DMA(Direct Acces Memory) con los IP Cores desarrollados.

2. Entralazado / Desentralazado

El entrelazador de bloques implementado, permuta símbolos de tamaño de un byte que conforman el mensaje. El IP Core de entrelazado recibe como entrada 256 bytes, mismos que guarda en memoria como un arreglo de M-renglones x N-columnas. Como primera instancia el IP Core intercambia dos columnas entre sí a conveniencia, y luego los muestra en salida por columnas hasta terminar con los 256 bytes y vuelve ha comenzar mientras exista información a procesar. El desentrelazador, recibirá entonces los 256 bytes permutados y con su algoritmo inverso se obtendrá la información original.

Para ejemplificar el proceso de acomodo de un mensaje, se tiene una matriz de M=2 y N=4 y se intercambiarán las columnas 2 y 3 mostradas en la figura 1

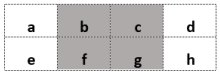

Figura 1. Matriz ejemplo de 2 x 4

Ahora, el algoritmo de entrelazado queda reflejado en la ecuación ec.(1):

$$renglones(k-1)+i-1 = columnas(renglones-i)+j-1 \quad (1)$$

Entonces, cuando el mensaje se almacenan en la memoria, se realiza por renglones seguidos, en éste ejemplo la memoria queda llena como se muestra en las figuras 2 y 3.

Figura 2. Contenido de una matriz de 2 x 4 acomodado en una memoria de 8 símbolos de 8 bytes cada uno.

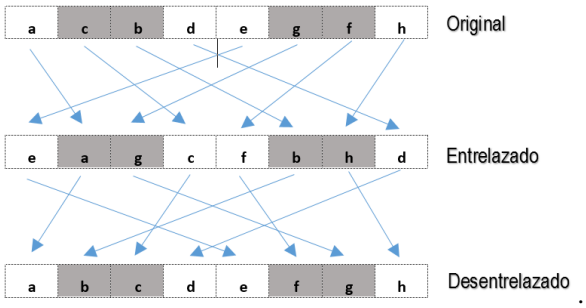

Figura 3. Proceso de Entrelazado y Desentrelazado

Para el desarrollo de éste proyecto, el entrelazador/desentrelazador recibe paquetes de 256 bytes y delvuelve el mismo tamaño del paquete a los procesos siguientes. El valor de M y N están ligados al tamaño de la memoria donde se almacenará, por tanto los valores multiplicados en conjunto deberán dar como resultado 256, por ejemplo $(32)(8) = 256$.

3. Bus AXI

3.1. Origen

El AXI(Advance eXtensible Interface) es un protocolo que determina una interfaz de conexión entre IP Cores (Núcleos de Propiedad Intelectual). Un bus AXI, es un protocolo de bus que soporta fases separadas de dirección/control y datos, transferencias de datos no alineados usando un flash de byte, transacciones basadas en ráfagas con sólo emitir la dirección de inicio, separa los canales de datos de escritura y lectura para habilita el bajo costo del DMA, habilidad para emitir para

emitir múltiples direcciones de salida, sin chocar con la ejecución de las transacciones, y fácil unión de las etapas.

3.2. Protocolos de bus on-chip AMBA AXI

Los protocolos ARM (*Advanced Risk Machine*) AMBA (*Advanced Microcontroller Bus Architechture*) son un estándar para comunicación *on-chip*. AMBA es un estándar abierto para la administración y conexión de bloques funcionales en un *System-on-chip (SoC)*. Facilita el desarrollo de diseños multi-procesador con un gran número de controladores y periféricos.

AXI *Advanced eXtensible Interface* es parte de ARM AMBA, una familia de buses para microcontroladores introducido por primera vez en 1996. La primera versión de AXI fue incluida en AMBA 3.0, lanzado en 2003. AMBA 4.0, lanzado en 2010, incluye la segunda versión de AXI, AXI4 [9].

Hay tres tipos de interfaces AXI4:

- **AXI4**: para requerimientos de mapeo de memoria de alto rendimiento.
- **AXI4 Lite**: para comunicación de mapeo de memoria simple o de bajo rendimiento (por ejemplo, desde y a registros de estado y control).
- **AXI4 Stream**: para transmisión a alta velocidad de datos.

3.3. AXI Stream

El protocolo AXI4 Stream actúa como un canal unidireccional para un flujo de datos con un *handshake* entre 2 y 9 señales. Lo que quiere decir que la información o datos son transmitidos de maestro a esclavo [7]. Para el caso más simple con un*handshake* entre dos señales, se requieren las señales:

- **TDATA**: El vector de datos.
- **TVALID**: Indica que los datos están presentes y son válidos para el esclavo.

Pero en la mayoría de los casos se utiliza una señal adicional:

- **TREADY**: Bandera presentada por el esclavo para indicar que está listo para recibir los datos.

Es posible prescindir de la señal TREADY y en este caso, se toma en cuenta que la transmisión es sin confirmación del esclavo.

Las otras señales que pueden ser utilizadas en la transferencia son:

- **TID**: Identifica la transmisión de la trama actual de datos(en caso de que diferentes tramas sean transmitidas entrelazadas).
- **TDEST**:Funciona como una dirección de destino en el caso de varios esclavos o diferentes destinos en el esclavo.
- **TLAST**: Indica el final de una trama, útil cuando las tramas varían en tamaño.
- **TKEEP**: Para especificar si todos los datos transmitidos en la trama deben tomarse en cuenta o no.
- **TSTRB**: Indica si los datos en cierta posición son útiles. El byte de posicionamiento puede darse el lujo de no actualizar cierta parte de la información que poseé el esclavo.
- **TUSER**: Datos en paralelo a la información principal que pueden incluir información adicional para el esclavo.

Considerando el caso más simple, un ejemplo de una transacción con las tres señales básicas se representa en la figura 2.7. La información es transferida cuando TVALID y TREADY se encuentran en estado alto (*handshake*).

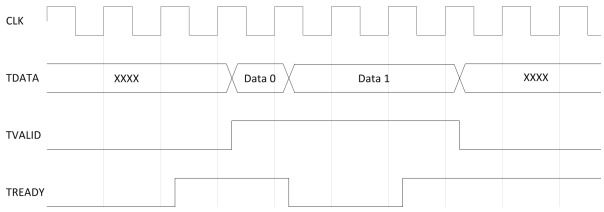

Figura 4. Transferencia de datos mediante protocolo AXI4 Stream. Imagen obtenida de [8].

4. Implementación del IP Core en Hardware y Software

La implementación de un IP Core se conforma de dos etapas que trabajan en conjunto. En la primera etapa, cuando se tiene un diseño funcional, se codifica en un lenguaje descriptor de hardware en algún software de desarrollo, que ya finalizado se empaqueta de manera que el IP Core está listo para añadirse al sistema que lo requiera. Además, se necesita crear una plataforma de hardware, que contendrá las conexiones entre el SoC Zynq 7020, el IP Core empaquetado y la comunicación AXI Stream para el rápido y eficiente acceso a memoria externa.

En la segunda etapa, con la ayuda de una herramienta de desarrollo de software se puede acceder y manipular el hardware, puesto que ahora ocupan de forma individual, una dirección de memoria. Finalmente, cuando se programe la tarjeta de desarrollo con el hardware y software construido, podremos controlar las acciones que se deseen desde la PC mediante el puerto serie.

4.1. Implementación en Hardware

Hablar de una implementación en hardware implica conocer los recursos disponibles. Se necesita de un software en el cual se pueda describir el hardware y los conocimientos del lenguaje descriptor a usar. También se debe contar con un FPGA que tenga la capacidad de albergar la aplicación, en éste caso el IP Core de Entrelazado/Desentrelazado. Es entonces que la implementación puede realizarse, desde que se diseña el algoritmo, se presenta como un IP Core, se descarga al dispositivo lógico programable, hasta la manipulación del mismo a través de una terminal.

4.1.1. Descripción del Hardware

La descripción del hardware se llevó a cabo con el uso del lenguaje verilog usando la herramienta de diseño electrónico ISE Design Suite 14.7 de Xilinx. En ésta última se plasmó el diseño del Entrelazador/Desentrelazador de datos, que muestra el esquemático de la figura 5.

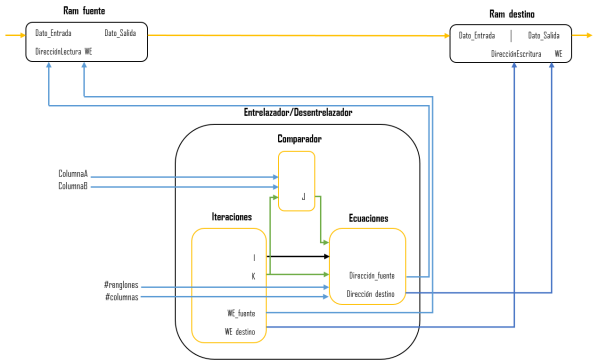

Figura 5. Esquemático del algoritmo de Entrelazado/Desentrelazado.

El algoritmo de entrelazado se forma de los siguientes componentes:

1. Aquel que intercambia las columnas , llamado “intercambia_columnas”.
2. Aquel que calcula la dirección de lectura y escritura de una memoria origen, llamado “calculo_direccion”.
3. Aquel que controla el inicio y fin del proceso de entrelazado/desentrelazado, llamado “FSM_Interleaver”.

Sin embargo, el agoritmo no trabaja por si solo, necesita una fuente de donde tomar los datos que va a procesar, a alguien que le diga en que momento hacerlo, también necesita saber donde colocar los datos procesados y avisar que dichos datos están listos; y es por ello que la arquitectura se compone del correspondiente algoritmo y a su vez de dos memorias tipo RAM de doble puerto. Una memoria contendrá los bytes que procesará el algoritmo, y la segunda memoria contendrá los bytes conmutados(como se vió en la figura 5).

intercambia_columnas Éste componente se encarga de intercambiar las columnas escogidas, de modo que recibe como entrada un contador que recorre todas las columnas, y cuando ese contador es igual a COL_A la salida $j = COL_B$ ó cuando es igual a COL_B la salida $j = COL_A$; pero cuando es diferente a esas dos columnas, la salida j es igual a la entrada j_{col}.

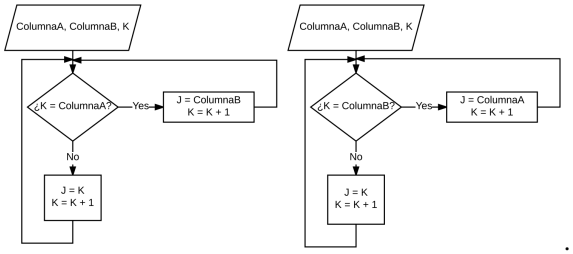

Figura 6. Toma de decisiones para intercambiar las columnas.

calculo_direccion Éste componente contiene una ecuación (ver 1) que representa la relación entre la dirección de lectura de una memoria origen, y la dirección de escritura de una memoria destino. La ecuación necesita saber el número de columnas COL y renglones REN totales; dos contadores, uno que recorra los renglones i_{ren} y otro que recorra las columnas j_{col} y un indicador que muestre si se está recorriendo una columna intercambiada.

FSM_Interleaver Éste componente se encarga dirigir a los dos últimos, así como proporcionarles las cuentas que recorren renglones y columnas, habilitar/deshabilitar la escritura en una memoria destino y avisar que el proceso del Interleaver terminó. Tiene como entradas una señal de reloj clk, una señal de reset rst, una señal de comienzo $start$, una señal de comenzar un nuevo proceso después de haber terminado uno anteriormente $restart$ y el número de columnas COL y renglones REN de la

matriz de 256 elementos. Las salidas, como se mencionaba anteriormente, son las cuentas para los renglones i_{ren} y para las columnas j_{col}, una señal que se conectará al WE de una memoria destino para habilitar/deshabilitar la escritura en ella, así como el fin de su proceso indicado con $done = 0$.

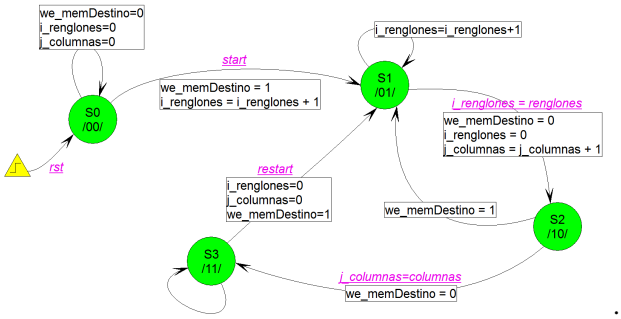

Figura 7. Máquina de estados que contiene el algoritmo de entrelazado/desentrelazado.

Cuando finalmente se tiene todo el sistema descrito en verilog, es posible crear el IP Core con ayuda de otra herramienta e incorporarlo en la plataforma de hardware para que pueda ser alimentado y probado en solitario.

4.1.2. Plataforma de Hardware

La plataforma de hardware se desarrolla en la herramienta de software Vivado 2016.1 de Xilinx, y se compone delSystem on Chip(SoC) Zynq 7020 que es el procesador del FPGA, el IP Core de Entrelazado, el IP Core de Desentrelazado y del protocolo de comunicación AXI Stream que hace posible la interacción con los datos guardados en memoria y entre los Cores (ver Figura 8). Una forma de poder apreciar de mejor manera el desarrollo de la plataforma, es tener al sistema en bloques como lo maneja Vivado por defecto, para así ver claramente las conexiones entre los mismos y detectar errores casi de forma inmediata.

En la Figura 8) se identifican perfectamente las partes necesarias para arrancar nuestro sistema de Entrelazado y Desentrelazado.

1. El $Zynq7ProcessingSystem$ es quien contiene el sistema de procesamiento, los controladores de memoria, los protocolos de comunicación.
2. El $Interleaver\_AXI2$ es el IP Core que trabaja como Entrelazador con interfaz AXI.
3. El $Interleaver\_AXI3$ es el IP Core que trabaja como Desentrelazador con interfaz AXI.
4. El $AXIDirectMemoryAccess$ es quien interactua directamente con la memoria sin la intervención del procesador, haciendo eficiente en tiempo la lectura y escritura de datos en el memoria.

Finalmente se encuentran bloques como el $reset$ activo en bajo y el reloj clk, que alimentan a todo el sistema.

4.2. Plataforma de Software

La plataforma de software es desarrollada en la herramienta de software *Xilinx Software Development Kit 2016.1* usando el lenguaje de alto nivel C para realizar la interacción con el hardware a través de una terminal de tipo serial(ver Figura 9).

Desde ésta plataforma se configura la comunicación serial para así interactuar con el software e indicarle mediante el teclado en que momento transmitir entre otras configuraciones.

5. Resultados

El primer paso para observar el funcionamiento del algoritmo implementado es simular desde el más mínimo bloque en el diseño y comprobar la salida esperada. Por tanto se proseguirá mostrando los resultados en la implementación de hardware(ver Figura 8).

Mediante la herramienta de software SDK, se descargó la plataforma de hardware en la tarjeta de desarrollo ZedBoard (ver Figura 10).

El proceso en general consiste en entrelazar una cadena de letras(cada letra representa 1 byte=8 bits) guardadas en la memoria de la tarjeta de desarrollo y desentrelazar la cadena conmutada, para depués guardarla en la misma memoria, y obtener de un solo paso la correcta salida que verifica la decuada funcionalidad de los dos IP Cores respectivamente.

Una teminal serial fue usada para que se determinara qué hacer y cuándo en el sistema de Entrelazado/Desentrelazado de datos. En la plataforma de software se definieron distintos comandos para dicho fin y los siguientes son los que se usan por defecto:

comando e Mediante éste comando se carga el contenido de un archivo a memoria(ABCDEFGHABCDEFGH...).

comando b Mediante éste comando se define el tamaño en bytes del paquete que se va a procesar, para nuestro caso se define en 256.

comando m Mediante éste comando se puede desplegar el contenido de un segmento de la memoria(memoria_origen [o]) del cual se alimentará el IP

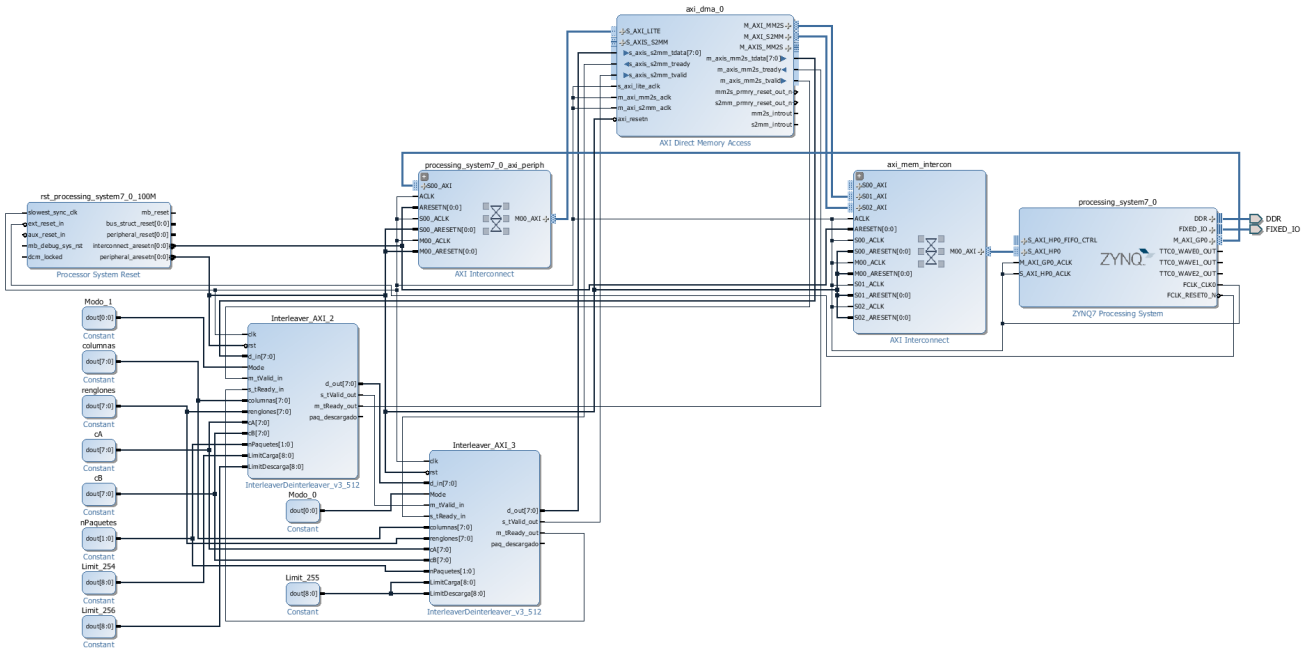

Figura 8. Diagrama de Bloques de la Plataforma de Hardware

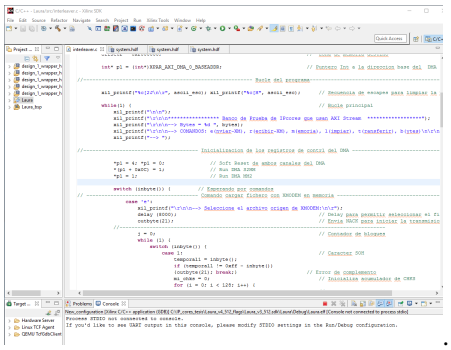

Figura 9. Código en la Plataforma de Sofware Vivado SDK

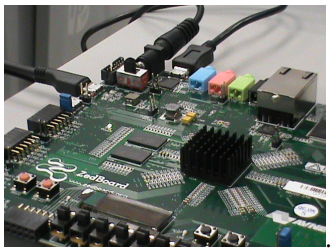

Figura 10. Tarjeta de desarrollo ZedBoard

Core de Entrelazado, y otro segmento de la memoria(memoria_destino [d]) que contendrá el paquete procesado.

comando t Mediante éste comando se realiza la transmisión de la infomación.

Al cargar el contenido del archivo a memoria y definir los 256 bytes se puede teclear el comando t para iniciar la transmisión, que de ser correcta, se podrá revisar la memoria destino y observar el resultado. En la Figura 11 se observa el procedimiento antes descrito y también que la cadena original(en memoria origen) es igual a la cadena obtenida(en memoria destino), desmostrando que el procesamiento fue exitoso.

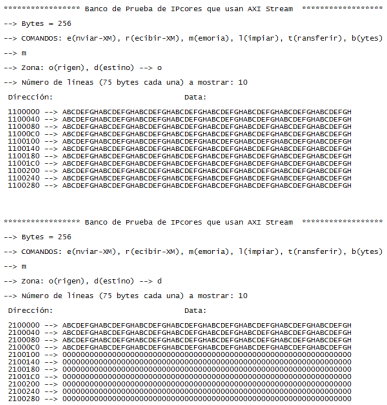

Figura 11. Terminal que muestra el contenido de la memoria origen y destino.

Sin embargo, también se puede observar lo que pasa en tiempo de ejecución directamente en el SoC que la infomación sea la que arroja la terminal, y es por ello que muestro la carga de bytes desde memoria a

la memoria RAM dualPort del IP Core Entrelazador(ver Figura 12).

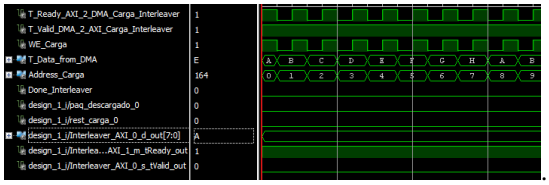

Figura 12. Carga de bytes en el IP Core de Entrelazado

Y de igual manera, como el proceso de entrelazado es transparente, debido a que sólo se muestra el resultado final, en la siguiente figura(ver Figura 13) se encuentra parte de los bytes entrelazados.

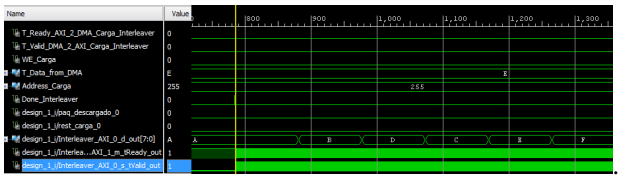

Figura 13. Salida IP Core Entrelazado

6. Conclusiones

Con los recursos de software y hardware, fue posible desarrollar la implementación en hardware del IP Core de Entrelazado/Desentrelazado. El Zynq, tiene la capacidad requerida para contener todo un sistema de telecomunicaciones, además, gracias al procesador y por facilidad, se pudo manipular el IP Core desde la PC mediante una terminal de tipo serial.

En cuanto a las herramientas de software usadas, se puede decir que cuentan con las funciones necesarias y más, en Vivado por ejemplo, su interfaz gráfica nos permitió hacer todo tipo de pruebas en el menor tiempo posible, haciendo una ventaja potencial por disminuir el tiempo de desarrollo.

Finalmente fue posible verificar que la comunicación entre IP Cores fue exitosa pues con ayuda de la herramienta “debug” llamada ChipScope, la información pasa de uno al otro sin pérdidas y en el momento requerido, así como los resultados esperados.

Referencias

[1] Sklar, Bernard *“Digital Communications. Fundamentals and Applications”*. Prentice Hall, 2001.

[2] Abdelmohsen, A. Khater Mohamed M, Khairy and S.E.D, Habib. *“Efficient FPGA Implementation for the IEEE 802.16e Interleaver”*. 2009 International Conference on Microelectronics.

[3] Upadhyaya, BK and Sanyal, SK. “VHDL modeling of convolutional interleaver-deinterleaver for efficient FPGA implementation”. International Journal of Recent Trends in Engineering, Academy Publisher, Finland. Vol. 2, Num. 6, p. 66-68, 2009.

[4] Andrews, Kenneth Heegard, Chris and Kozen, Dexter. “A Theory of Interleavers”. Cornell University, 1997.

[5] Eric Tell and Dake Liu. “A HARDWARE ARCHITECTURE FOR A MULTI MODE BLOCK INTERLEAVER”.

[6] VanCourt, Tom and Herbordt, Martin “Application-Specific Memory Interleaving for FPGA-Based Grid Computations: A General Design Technique, *“Field Programmable Logic and Applications, 2006. FPL’06. International Conference on”*. IEEE, p.1-7,2006.

[7] Toussaint, Cedric y Benharbone, William “AXI4-Stream” $https : //wiki.electroniciens.cnrs.fr/index.php/FPGA\_CPLD\_ : \_Guides\_ : \_AXI4 - Stream$ 2015 Consultado Enero, 2016

[8] Analytics Engines “Introduction to AXI” $http : //www.analyticsengines.com/developer - blog/introduction - to - axi/$ 2015 Consultado Enero, 2016

[9] Xilinx, Inc AXI Reference Guide $http : //www.xilinx.com/support/documentation/ip\_documentation/ug761\_axi\_reference\_guide.pdf$ 2015 Consultado Enero, 2016