Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE0523-Circuitos Digitales II

Grupo:1
Profesor: Ing. Jorge Soto Avendaño

Proyecto 1

PHY Layer PCI Express

Autores: *Carnés:*
Bryan Cervantes Ramírez B31726
Leonel Sánchez Lizano B26213
Jose Ricardo Soro Jara B36853

30 de mayo de 2018

Índice

1. Introducción **3**

2. Estudio de Mercado **4**

3. Descripción arquitectónica **6**

4. Módulos Individuales **7**
4.1. Mux 7
4.2. Byte Striping 7
4.3. Paralelo a Serial 8
4.4. Generador de Clocks 9
4.5. Serial a Paralelo 9
4.6. Byte Unstriping 10
4.7. Demux 10

5. PHY Layer **11**
5.1. Tx 11
5.2. Rx 12

6. Conclusiones **13**

Referencias **14**

1. Introducción

Los puertos PCI (Peripheral Component Interconnect por sus siglas en inglés) tiene como función básica exprimir al máximo la velocidad de la computadora. El sistema se basaba en colocar un puerto PCI por conexión para lograr el objetivo. PCI Express amplía y duplica la velocidad de transferencia de datos de la original PCI. PCI Express es una conexión serial de dos vías, que lleva los datos en paquetes a lo largo de dos pares de carriles de datos punto a punto. [1] Hay 5 formatos de PCIe:

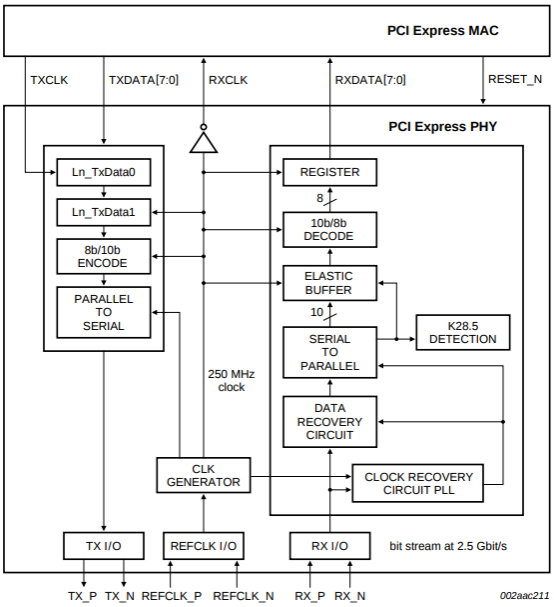
- PCI Express 1x con una capacidad de 250 MB/s, la misma aparece en la mayoría de tarjetas madre actuales.
- Express 2x con una potencia de 500 MB/s es menos común, reservada para los servidores.
- PCI Express 4x con una potencia de 1000 MB/s asimismo está reservado para los servidores.
- PCI Express 16x con una potencia de 4000 MB/s presente en todas las tarjetas modernas, es el equipo estándar para tarjetas gráficas. La ranura de 16-lane (x16) ha reemplazado el puerto de gráficos acelerado (AGP) en muchas placas base y se ajusta a una tarjeta gráfica PCIe.
- PCI Express 32x con una potencia de 8000MB/s tiene el mismo formato que el 16x PCI Express, a menudo se utiliza en las placas base de gama alta para la alimentación del bus SLI.

2. Estudio de Mercado

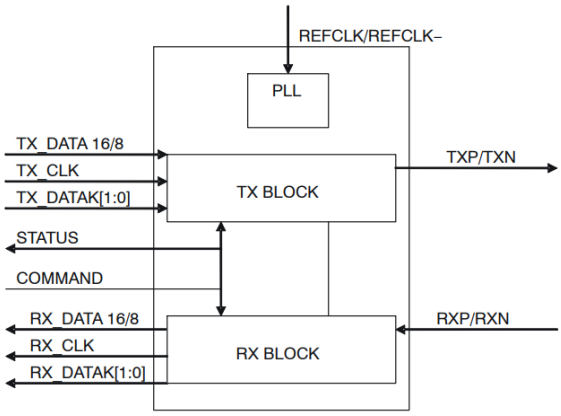
Se escogieron los componentes `PX1011B` y `XIO1100` de los fabricantes NXP Semiconductors y Texas Instruments respectivamente para analizar los circuitos integrados con los cuales se puede implementar la capa PHY de una interfaz PCIe. Por su parte, para la interfaz USB se escogieron los componentes `USB3280` y `PTN5100D` de los fabricantes Microchip Technology y NXP Semiconductors

Bloques funcionales de capa PHY para interfaz PCIe

En la Figura 1 se puede apreciar los diagramas con los bloques funcionales para cada componente proporcionados por los fabricantes. En el caso del `PX1011B` la descripción es más detallada, por tanto se pueden identificar bloques como el 'Encoder', 'Parallel to serial', 'Serial to parallel', 'Elastic buffer', etc. En contraste, para el `XIO1100` se presenta un diagrama mucho más general en donde solamente se indican las entradas y salidas a los bloques 'TX' y 'RX' así como la especificación del bloque 'PLL'.



(a) Diagrama funcional PX1011B

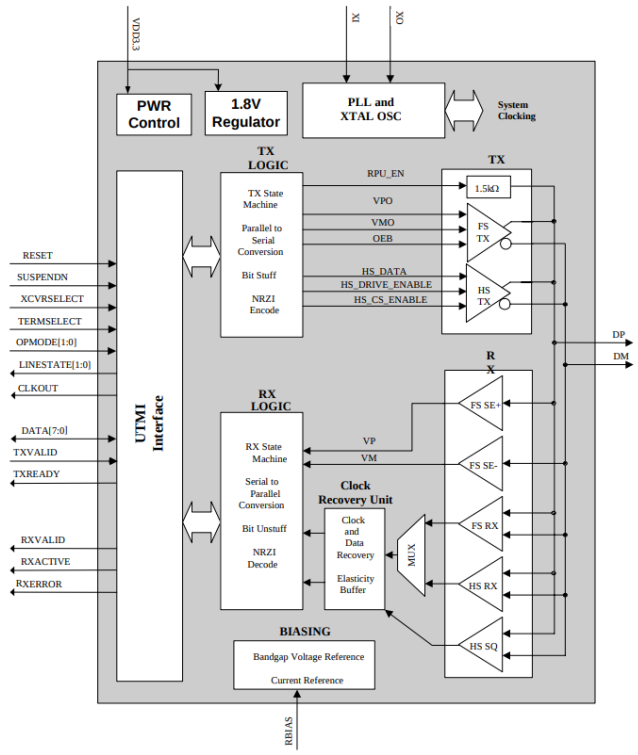


(b) Diagrama funcional XI01100.

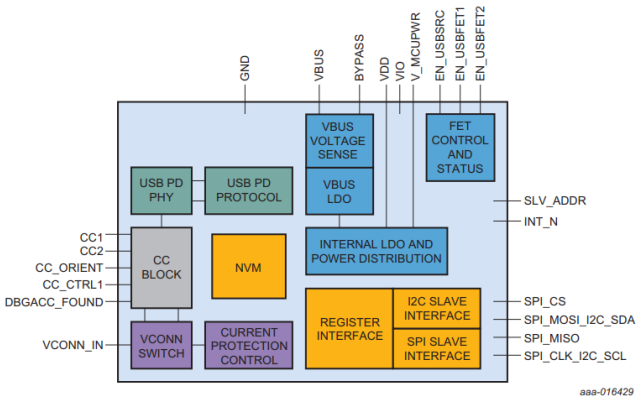
Figura 1: Bloques funcionales para ambos componentes. Tomado de [2] y [3] respectivamente

Bloques funcionales de capa PHY para interfaz USB

De forma análoga a lo realizado anteriormente, se aprecia en la Figura 2 un diagrama mucho más concreto que el otro, en el `USB3280` se identifican claramente los módulos tales como el 'Parallel to serial', 'Serial to parallel', entre otros.



(a) Diagrama funcional USB3280



(b) Diagrama funcional PTN5100.

Figura 2: Bloques funcionales para ambos componentes. Tomado de [4] y [5] respectivamente

Diferencias entre las capas PHY de PCIe y USB

Dentro de las diferencias que hay entre las capas, las más significativas están relacionadas con su velocidad, mientras que para PCIe se soportan velocidades de transmisión de datos de hasta 32 GT/s, para USB lo máximo viene a ser 10 GT/s. Además para USB se cuenta con 'Low Frequency Periodic Signaling' (LFPS), que es empleado para comunicar dos puertos a través de un link en estado bajo. Otro de los aspectos que los diferencia es el *lane* margining' en PCIe, que consiste básicamente en hacer aislar uno de los lanes para poder obtener un determinado margen eléctrico según corresponde.

3. Descripción arquitectónica

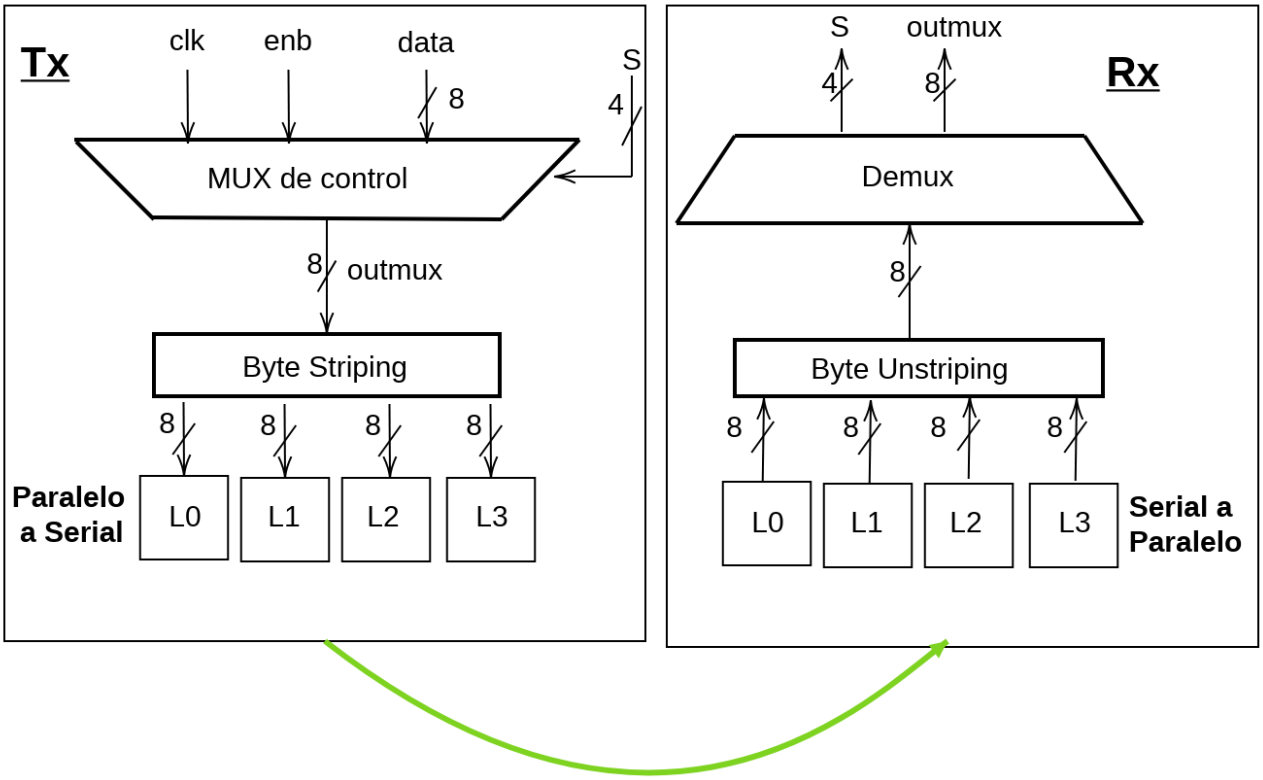


Figura 3: Descripción general

Como bien se especificó previamente, el proyecto consistió en la implementación de una parte de la capa física, al analizar la Figura 3 se puede observar que lo primero que se incorpora al diseño es un Mux de control encargado de discriminar entre la entrada `data` y determinados parámetros (paquetes) a partir de una codificación dada por `S` y en donde la salida de 8 bits del Mux sirve como entrada al 'bloque' del Byte Striping. El `Byte Striping` recibe estos bits y empieza a asignarlos a cada uno de los 4 *lanes*correspondientes dependiendo del tipo de señal que recibe. Una vez superada esta fase, los *lanes*ingresan a la etapa de `Paralelo a Serial`, en donde tal y como su nombre lo sugiere, a partir de cada set de 8 bits en cada *lane*, la salida en esta etapa será de bit en bit y con esto se finaliza la etapa `Tx`.

Para la etapa `Rx` se toman los bits individuales como entradas para la etapa `Serial a Paralelo` de cada *lane* para ir formando paquetes de 8 bits que serán pasados como entradas para el módulo `Byte Unstriping` en donde la salida proporciona los paquetes de 8 bits deseados a partir de determinas reglas que serán explicadas con mayor detenimiento más adelante. Seguidamente se encuentra el `Demux`, quien se encarga de identificar los 8 bits que recibe en la entrada y junto con la señal `S` especificar si se trata de paquetes de control específicos (SDP, IDL, COM, etc) o bien otro tipo de data que se esté transmitiendo. A continuación se presentan los detalles de la implementación de cada módulo a partir del diseño explicado recientemente.

4. Módulos Individuales

4.1. Mux

El módulo MuxCtrl.v consta de las entradas `enb`, `clk`, `[7:0]data`, `[3:0]S` y una serie de parámetros que definen los diferentes paquetes de control que podrían ser reconocidos por la señal `S` (ver Cuadro **??**) a partir de distintos cases, o bien, el `data` proporcionado desde el Testbench y que será lo que se almacene en la salida `outmux`. Este módulo en términos generales contiene una lógica interna sencilla, los resultados con las pruebas respectivas se aprecian en la Figura 4.

S	outmux	Hexadecimal
0000	data	XX
0001	COM	BC
0010	PAD	F7
0011	SKP	1C
0100	STP	FB
0101	STD	5C
0110	END	FD
0111	EDB	FE
1000	FTS	3C
1001	IDL	7C

Cuadro 1: Codificación para `S`

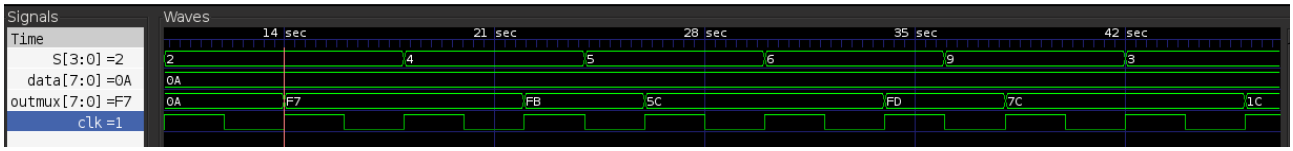


Figura 4: Prueba funcionamiento de Mux de control

4.2. Byte Striping

El funcionamiento del módulo striping.v, a grandes rasgos es el siguiente. Este recibe paquetes de datos de 1 byte uno por uno (cada ciclo de reloj) y se encarga de distribuir estos bytes en 4 salidas que van hacia los *lanes* respectivos uno por uno, en orden del primero al cuarto, y empezando de nuevo cuando se acaban. Sin embargo no es tan sencillo como distribuir constantemente cada byte que se recibe, ya que además de datos, puede recibir otros tipos de señales, como por ejemplo COM, IDLE, o SKIP, las cuales se deben mandar a todos los *lanes* simultaneamente y además, los datos solo se deben transmitir cuando se recibe una señal de start, ya sea STP o SDP, y se debe finalizar la transmisión cuando se reciba la señal END. Las señales start siempre deben ser enviadas por el primer *lane* mientras que las señales END, deben ser enviadas por el último lane.
En la figura 5 se observan las ondas generadas por la prueba realizada al módulo. la entrada es *fromMux*, y las salidas son llamadas *TLn*, donde TL significa *To Lane*, y *n* es el numero de *lane* correspondiente. En la prueba podemos apreciar lo siguiente:

- Inicialmente se envía una señal SKIP (1C), esta se ve reflejada en las 4 salidas.
- Después se envía la señal IDLE (7C). Al igual que la señal anterior, esta se envía a todos los *lanes*.

- Se envía la señal de START, STP(FB). Esta señal automáticamente se transmite al *lane* cero en TL0 exclusivamente. Las demás salidas se mantienen intactas. Al detectar la señal de START, el *reg D*, pasa a encendido, lo cual significa que estamos transmitiendo datos.
- Se envía la secuencia de datos 01, 02, 03, 04, 05, 06. Se puede observar como después de la señal start, se empiezan a acomodar en orden uno por uno, estos bytes en el *lane* correspondiente. Esto se controla por medio de un contador de dos bits que indica en que *lane* se debe transmitir el dato que se recibe del Mux. Este incrementa en uno, cada vez que se envía un dato
- Se envía la señal END(FD), y esta se transmite al último *lane*. Al detectar una señal de END, *D*, se apaga, indicando que ya finalizó la transmisión de la secuencia de bytes actual, de esta manera el módulo estará esperando una señal COM, IDLE, o SKIP, para transmitirlo a todos los *lanes*, y interpretará alguno de estos bytes como datos normales de manera que no los enviará a un solo *lane* específico.

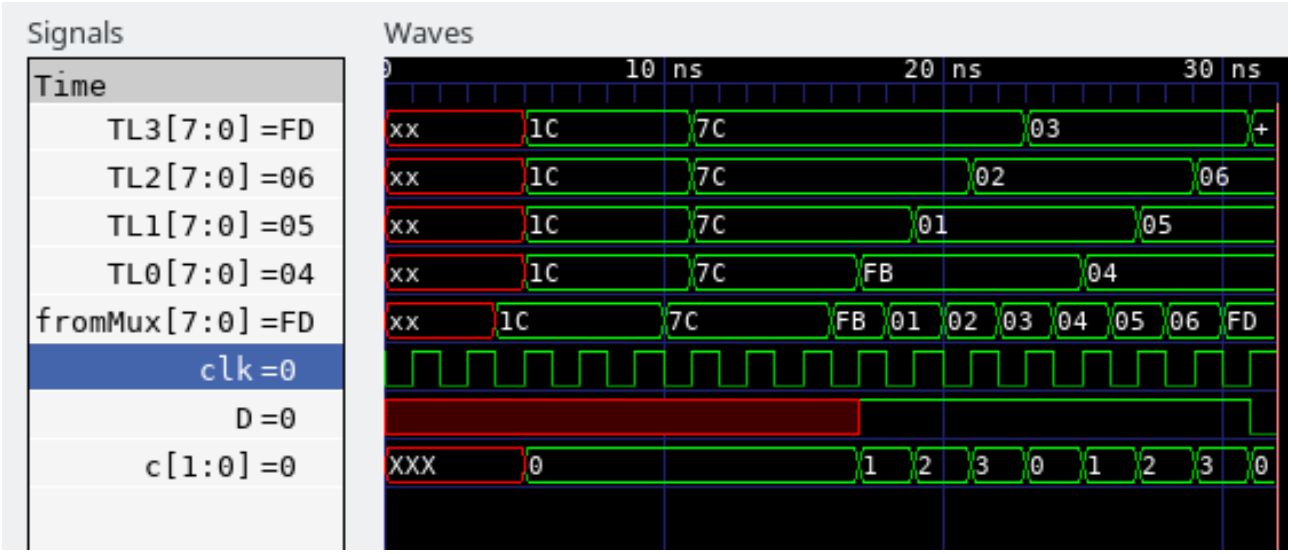


Figura 5: Prueba de módulo striping.v

4.3. Paralelo a Serial

El módulo ParaleloSerial.v se encarga de tomar una entrada de 8 bits y dar una salida de un único bit de forma ordenada. El módulo cuenta con cuatro entradas: enable y reset de un bit cada uno y tienen como función el control del módulo; al igual que el clock que determina la frecuencia a la que se leen los bits en la entrada; por último está el input Entrada el cual recibe 8 bits y representa la información se quiere serializar. La única salida es la de un bit y es importante notar que hay un registro de contador de 4 bits con el cual se cuenta del 0 al 7 para saber que bit de la entrada se quiere enviar a la salida.
Para las pruebas que se realizaron se utilizaron tres entradas: [11100110], [00001101] y [01011101]. En la siguiente figura se aprecia los resultados de esta prueba.

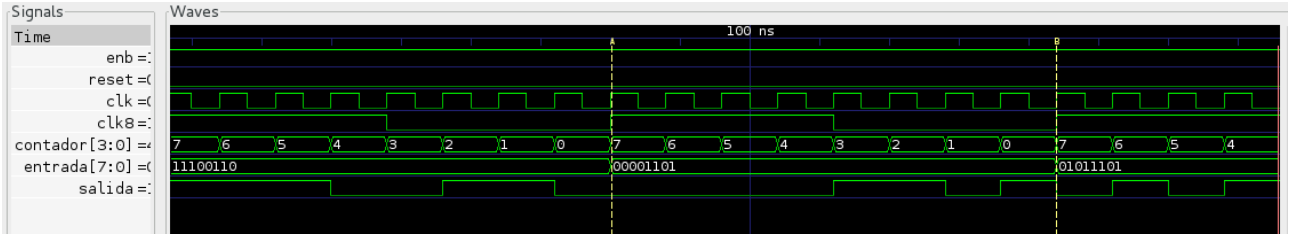


Figura 6: Resultados de prueba de Paralelo a Serial

Se nota entonces que el funcionamiento es el correcto pues se logra serializar perfectamente cada entrada; esto gracias a que en las pruebas se utilizo un generador de Clocks para leer entradas a un ritmo diferente del que se extraen salidas.

4.4. Generador de Clocks

El módulo GeneradorClk.v se encarga de generar señales CLK que funcionen a diferentes frecuencias proporcionales y cuyos flancos positivos se alineen entre sí. El módulo presenta dos entradas de control reset y enabler; y una entrada de CLK que opera a la mayor frecuencia. En el funcionamiento interno se tiene un registro de contador que se encarga de marcar la primera frecuencia a la que funcionará la salida CLK10 al contar 2 posedges. Se cuenta con tres salidas: CLK10, CLK20 y CLK40; las cuales funcionan respectivamente 4 veces más lento que CLK; 8 veces y 16 veces más lento.

A continuación se muestra una prueba del funcionamiento del generador de CLKS.

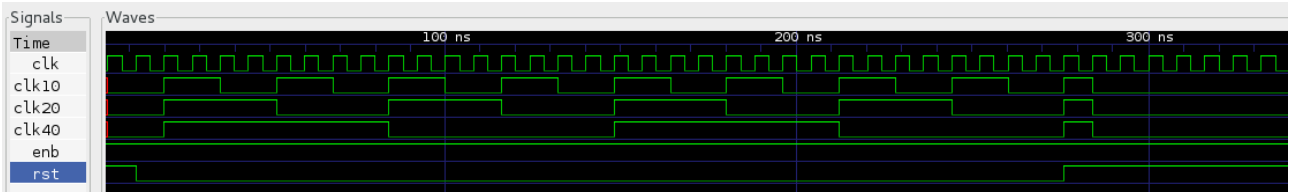


Figura 7: Demostración de Generador de Clocks a 4 frecuencias distintas

4.5. Serial a Paralelo

El módulo serialParalelo.v se encarga de tomar 8 entradas ordenadas de un bit y dar una salida de 8 bits de forma paralela. Las entradas son similares a su contraparte ParaleloSerial.v donde hay dos entradas de control reset y enb; y una entrada de CLK que determina a que frecuencia se leen entradas seriales para ponerlas en la salida paralela; el input principal es el de Entrada el cual recibe bits de uno en uno para irlos guardando en un registro de Buffer llamado Bits de tamaño 8; cuando este se llena, condición determinada por un registro contador de cuatro bits este buffer se envía al único output llamado Salida.

Para las pruebas de este módulo se utilizaron entradas de los bits 0, 1, 1, 0, 0, 1, 1, 0; 1, 0, 1, 0, 0, 1, 0, 1; 1, 0, 0, 0, 0, 0, 0, 0; 0, 1, 1, 1, 1, 1, 1, 1. En la siguiente figura se puede notar como se ingresa esa secuencia de entradas y la salida obtenida es [0, 1, 1, 0, 0, 1, 1, 0], [1, 0, 1, 0, 0, 1, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0] y [0, 1, 1, 1, 1, 1, 1, 1] como se esperaba.

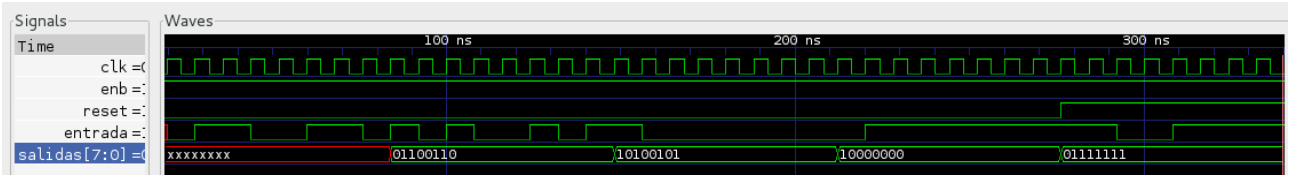


Figura 8: Prueba de módulo serialParalelo.v

4.6. Byte Unstriping

El módulo byte unstriping se encarga de hacer lo inverso del byte striping. Este recibe bytes de los 4 lanes y los transmite a la salida del módulo. En la figura 9 se observa como se recibe en los lanes, la secuencia de datos: FB, FF, FF, FD. Esta secuencia después se ve transmitida al lane de salida llamado *toDemux*.

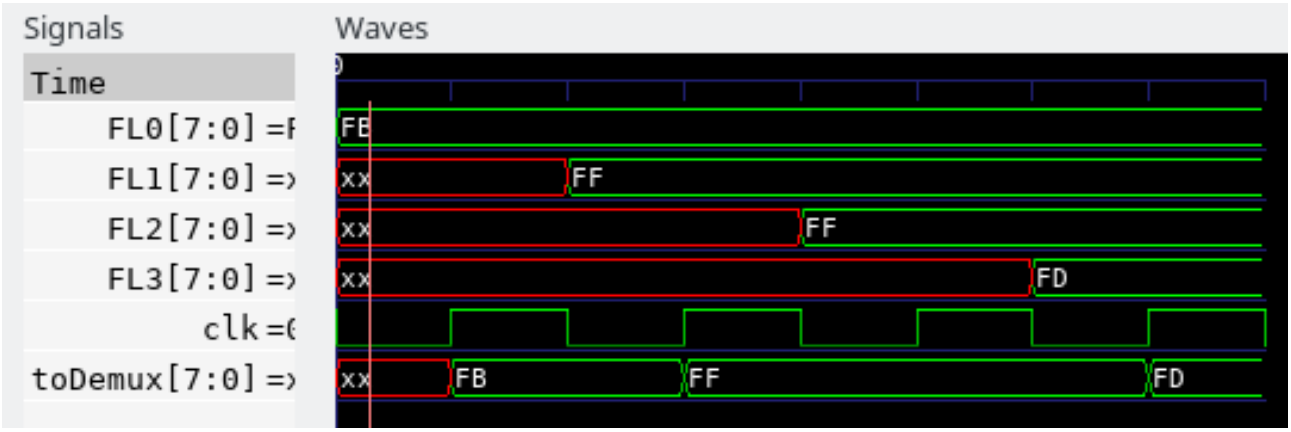


Figura 9: Prueba de módulo unstriping.v

4.7. Demux

El módulo deMux.v se dedica a interpretar los caracteres que le envía el ByteUnstriping; este interpreta los caracteres basándose en los caracteres reservados y dar como salida su respectivo código de Select definido anteriormente en el Mux; por otro lado se asegura de pasar solo aquellos caracteres que cumplan el estatus de Data (aquellos caracteres que se encuentran entre un start y un end). El módulo cuenta con una entrada enabler y una de CLK para controlar cada cuanto se procesa un carácter; además se tiene una entrada de Data que es de 8 bits para representar caracteres. Se tienen dos salidas; siendo una de ellas S la cual es de cuatro bits y representa el tipo de dato procesado; y una salida outMux que representa el dato que se quiere entregar, este es de ocho bits.
Para probar este módulo se le enviaron ciertos caracteres; Estos se enviaron de la forma; IDLE, IDLE, COM, STP, DATA, DATA, DATA, DATA, DATA, DATA, END; donde los caracteres de data pueden ser cualquiera, por lo cual se eligió utilizar 01, 02, 03, 04, 05 y 06 por lo cual se envían los siguientes caracteres hexadecimales: 7C, 7C, BC, FB, 01, 02, 03, 04, 05, 06, FD con lo cual se esperan salidas interpretadas de S en 1001, 1001, 0001, 0100, 0000, 0000, 0000, 0000, 0000, 0000, 0110 (basado en los códigos de select definidos en el Mux) y una salida outMux de XX, XX, XX, XX, 01, 02, 03, 04, 05, 06, XX. Los resultados se ven en la siguiente figura.

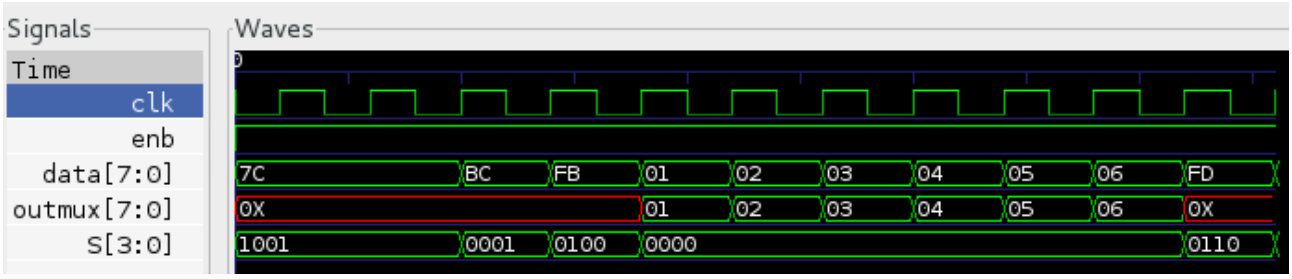


Figura 10: Prueba de módulo deMux.v

Se nota el correcto funcionamiento del modulo y como regresa 0X para no enviar el codigo cuando no se envia data.

5. PHY Layer

5.1. Tx

Para construir el modulo Tx.v se decidio utilizar una entrada única de enabler y reset que se asigna a todos los módulos para sincronizar el funcionamiento de estos. ademas de las señales CLK perinentes para mantener el funcionamiento adecuado del sistema. De igual forma se recibe una entrada de seleccion S de 4 bits que determina los caracteres que se están ingresando y una entrada Data de 8 bits que representa los caracteres de datos que se ingresan al transmitir propiamente bajo la etiqueta de Datos. Se tienen cuatro salidas donde cada Salida es de un bit y representa cada lane serializando cada salida del byte striping.

Para la prueba de este módulo se utilizó un set igual al utilizado en DeMux.v el cual era 7C, 7C, BC, FB, 01, 02, 03, 04, 05, 06, FD y sus resultados se adjuntan en la siguiente figura.

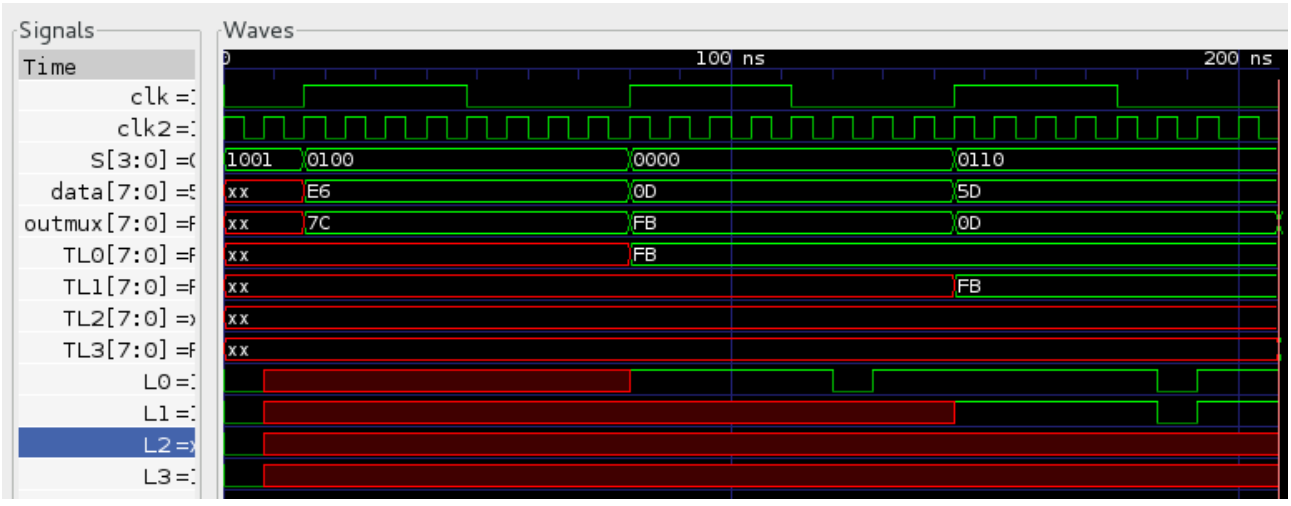


Figura 11: Prueba de módulo Tx.v

En la figura adjunta se nota que hubo un error en el cableado o el funcionamiento del Byte Striping el cual pierde sus salidas de 8 bits en cada Lane por lo cual se debe analizó posteriormente para corregir los errores.
Este análisis posterior se realizó y se lograron conseguir mejores resultados, los cuales se ven en la

figura 12. En esta podemos observar que los *lanes* del byte striping esta vez si reciben los datos tipo IDLE del Mux, sin embargo el problema sigue siento a la hora de sincronizar los *clocks*. La falta de sincronización de los *clocks* hace que los datos no sean transmitidos correctamente por los módulos.

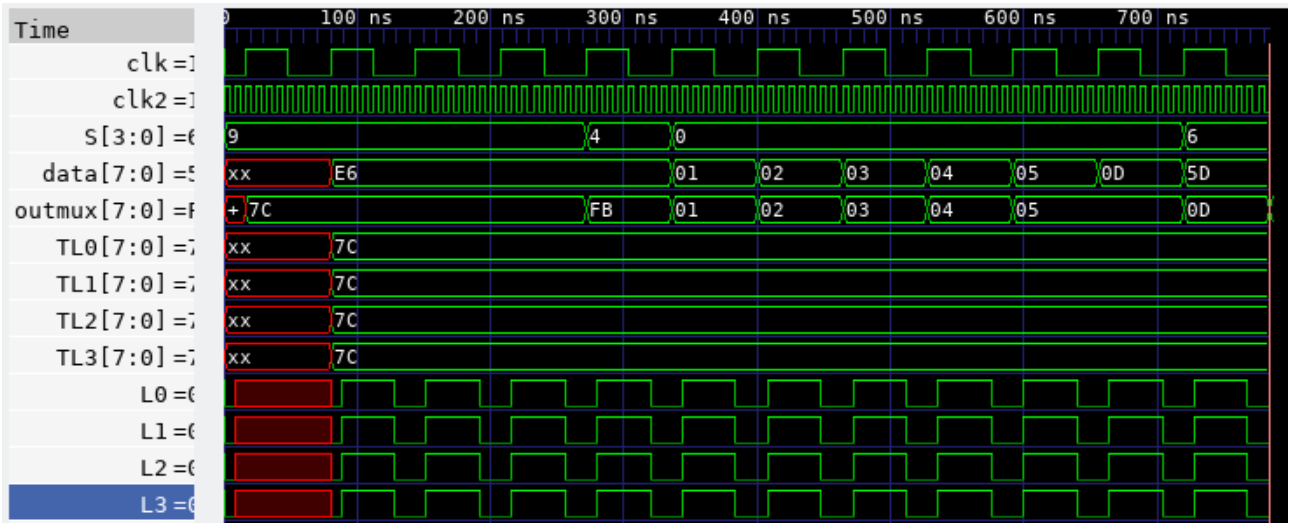


Figura 12: Segunda prueba de módulo Tx.v

5.2. Rx

El módulo Rx.v cuenta con las entradas de 1 bit de cada uno de los 4 lanes; además de las ya conocidas señales de control reset, enb y clk; este clock se encarga de generar los clocks a distinta frecuencia; este da como salida 8 bits de data que se dejan en 0X en caso de no transmitir palabras entre STP y END; además de una salida S de 4 bits que representa el caracter detectado y que coincide con la seleccion del Mux en Tx.

La siguiente figura incluye los resultados de una prueba donde se ingresaron las mismas palabras bit por bit de salida de los lanes de la prueba Tx

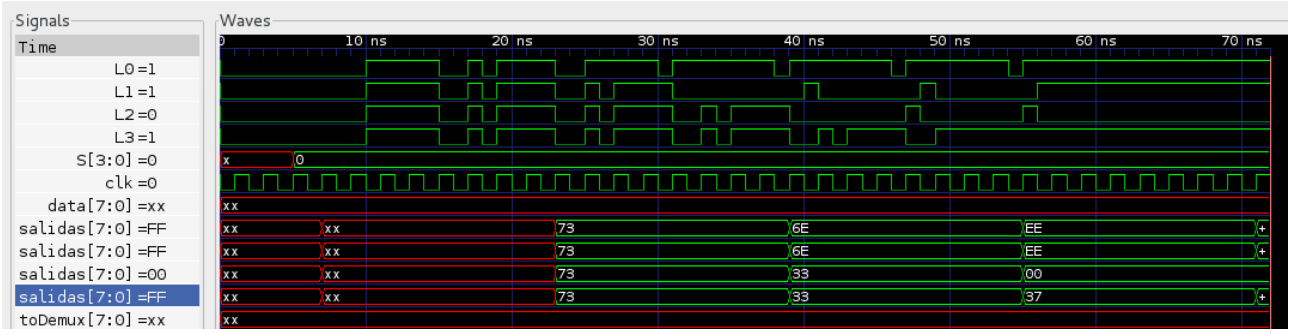


Figura 13: Prueba de módulo Rx.v

Se muestra entonces un problema en la conexión del byte unstriping con el demux pues este parece ser quien pierde las señales; y es aquí donde se deben concentrar esfuerzos para reparar el problema.

6. Conclusiones

La implementación de cada módulo por separado, y sus respectivas pruebas fue un éxito. Al realizar pruebas a cada módulo estos se comportaban de la manera esperada. Sin embargo los problemas surgieron a la hora de conectar todos los módulos para formar las cajas Tx y Rx. Se presentaron problemas de sincronización de relojes, lo cuál provocaba que la transmisión de datos a través de los módulo no fuera correcta, y por falta de experiencia, no se logró corregir estos errores.

En cuanto a la parte de sintetización con Yosys, a pesar de que se generaron los diferentes archivos con las descripciones estructurales de cada módulo, al efectuar las pruebas, las mismas no se comportaron según se esperaba producto de las dificultades presentadas con las conexiones entre dichos módulos. Sin embargo, para el caso del Mux de control, las pruebas sí mantuvieron el orden deseado, por lo que esto confirma que el diseño individual de los módulos no resultó ser tanto problema como sí lo fue el manejo de las etapas en conjunto.

Referencias

[1] C. Velásquez, "Puertos pci express: ¿qué son y para qué sirven?" http://www.islabit.com/76371/puertos-pci-express-tamanos.html, 2017, recuperado en mayo, 2018.

[2] "Px1011b datasheet," https://www.nxp.com/docs/en/data-sheet/PX1011B.pdf, 2011, recuperado en mayo, 2018.

[3] "Xi01100 datasheet," http://www.ti.com/lit/ds/symlink/xio1100.pdf, 2011, recuperado en mayo, 2018.

[4] "Usb3280 datasheet," http://ww1.microchip.com/downloads/en/DeviceDoc/3280.pdf, 2007, recuperado en mayo, 2018.

[5] "Ptn5100 datasheet," https://www.nxp.com/docs/en/data-sheet/PTN5100.pdf, 2017, recuperado en mayo, 2018.