Reporte del Proyecto 1

Jorge Muñoz Taylor
Daniel Díaz Orozco
César Valverde Zúñiga

Circuitos Digitales II
Profesor: Jorge Soto Avendaño

31 de marzo 2018

RESUMEN

En el presente trabajo se tenía como requerimiento diseñar una interface de comunicación física entre dos dispositivos (Transmisor) y (Receptor) de ellos, además se debía obtener una descripción conductual y estructural en Verilog.
Se diseña conductualmente y estructuralmente el transmisor y el receptor, mediante la integración de los diferentes módulos que conforman el dispositivo final.
Por último, se define un plan de pruebas para verificar el funcionamiento de los módulos diseñados.
Palabras Clave: *Interface, Transmisor, Receptor, Descripción, Conductual, Estructural, Verilog, Módulos.*

1. Estudio de mercado

Se seleccionaron los dispositivos:

- PX1011B de NXP.
- USB 2.0 PHY IP core de Arasan.

1.1. ¿Qué bloques funcionales trae la capa PHY para una interfaz PCIe?

Consiste en 4 grandes bloques: la interface Tx/Rx, el PLL y PMA[1], en la Figura 1 se muestra el diagrama funcional del chip escogido.

[1] Acrónimo del inglés **Physical Media Attachment Layer**, este modulo comunica la **PHY** con el exterior.

Figura 1. Diagrama de bloques funcionales de la interfáz PCIE

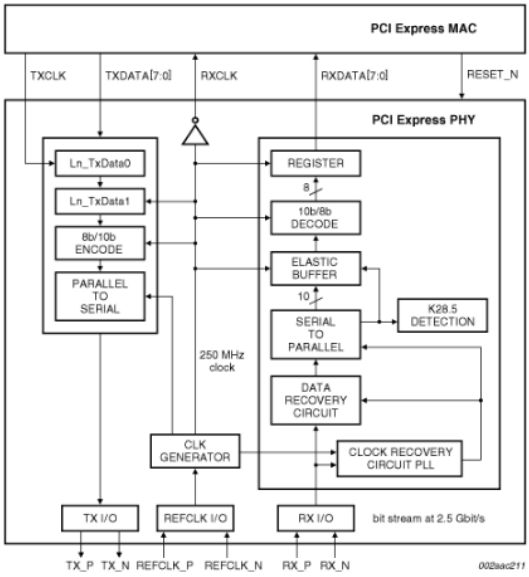


Imagen propiedad de NXP.

1.2. ¿Qué bloques funcionales trae la capa PHY para una interfaz USB?

Se compone de 3 módulos: interfáz Rx/Tx, PLL y OTG. En la Figura 2 se muestra el diagrama del circuito.

Figura 2. Diagrama de bloques funcionales de la interfáz USB 2.0

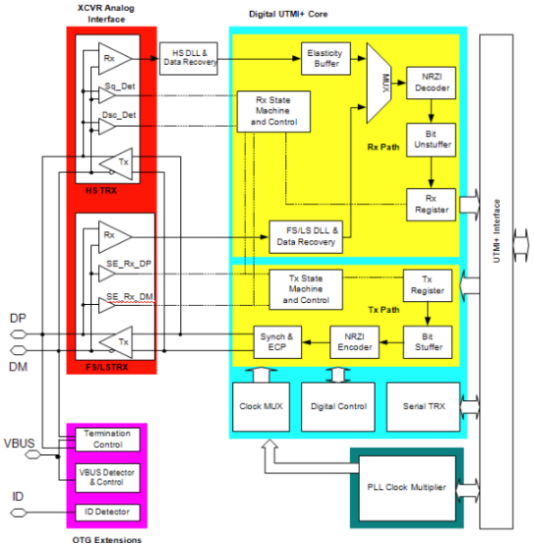


Imagen propiedad de Arasan

1.3. ¿Cuáles son las diferencias entre las capas PHY de PCIe y USB?

La diferencia entre lo dos protocolos es el *OTG*[2], este módulo permite que los dispositivos conectados a través de un puerto USB puedar ser conectados/desconectados en caliente sin perder datos.

1.4. ¿Qué posibilidad hay de hacer un diseño que combine la funcionalidad de ambas interfaces: PCIe y USB?

Si existe la posibilidad de desarrollar un dispositivo que integre una solución con funcionalidades de ambas interfaces, es más, un ejemplo de ello son las ExpressCard, estas son tarjetas compactas (small form factor) para ordenadores y otros dispositivos portátiles, la tarjeta se puede comunicar con el host mediante dos interfaces distintas: PCI-Express, velocidad básica x1, o bien USB 2.0, donde la elección de una interfaz u otra queda a elección de la propia tarjeta en función de cual sea la interfaz más apropiada para realizar su tarea.

1.5. ¿Cuál debería ser el precio de cada circuito para ser competitivo con los productos que ya están en el mercado?

Debe definirse un precio acorde a las realidades del mercado y dependiendo de los avances que se generen o se logren en dichos dispositivos, es muy difícil establecerlo antes del desarrollo del mismo, por lo que debería tratar de homologarse.

1.6. ¿Cuál debería ser la frecuencia de operación y el consumo de energía para ser competitivos?

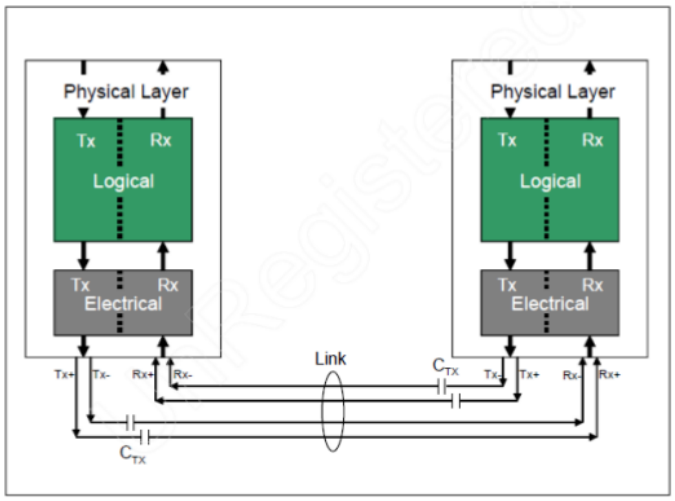
Como ejemplo podría ser la frecuencia de los productos PCIe en el mercado como la tarjeta de adquisición NI PCIe-6251 de National Instruments de 16 entradas y 4 salidas analógicas con una resolución de 16 bits y una frecuencia de muestreo de 1.25MSample/s (entre otras prestaciones) dotada de conector PCI-Express x1 (250MB/s) por lane. Es decir, cada canal genera un tráfico de 1.25MS/s*16bits = 2.38 MB/s, por lo que con un conector x1, se podría tener hasta un máximo de 80 canales aproximadamente. Además de un consumo bajo de potencia

[2]Acrónimo del inglés **On The Go**.

2. DESCRIPCIÓN ARQUITECTÓNICA

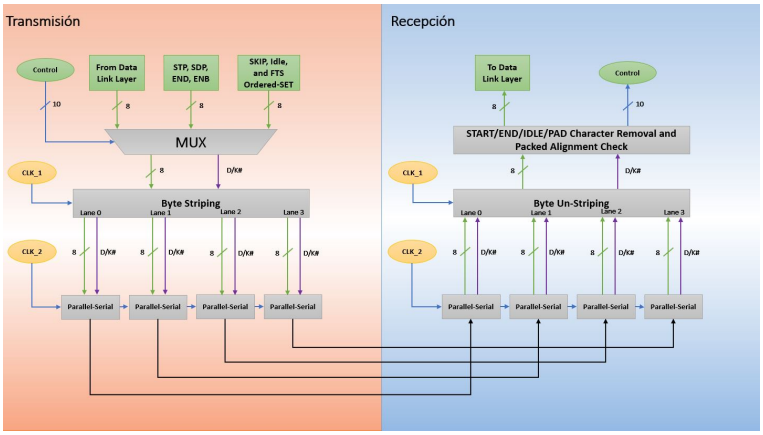
Se estable el enlace de dos capas físicas, es decir, entre un transmisor y un receptor, tal como se denota en la figura Figura 3.

Figura 3



Este esquema de comunicación de esta arquitectura se muestra en la figura Figura 4, en ella se evidencia como el multiplexor de control es encargado de manejar las entradas provenientes desde una secuencia en el probador, y además se encarga de generar los códigos de interpretación para que posteriormente sean acomodados y clasificados por el byte striping. Este elemento se encarga de enviar estos bloques de 8 bits a los serializadores para que luego en el receptor, sean empaquetados en bloques de datos y pasados al un-byte striping para que continuen su camino hasta el de-multiplexador de control en el destino final propiamente en el receptor.

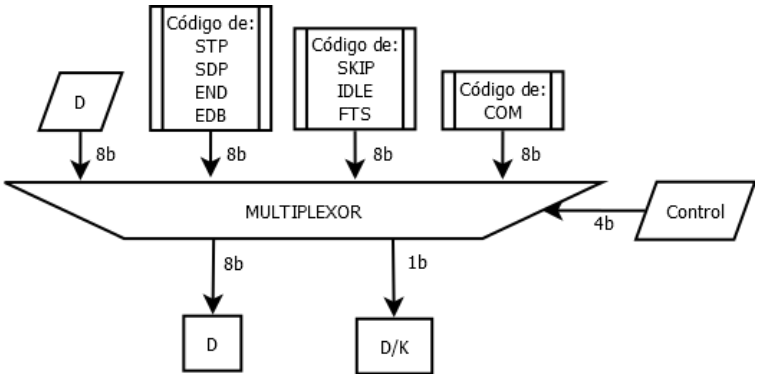
Figura 4



2.1. Multiplexor y lógica de control

El MUX tiene cuatro entradas y dos salidas.En la Figura 5 se ve un esquemático con sus entradas y salidas.

Figura 5. Diagrama del bloque de multiplexado.



Note que las entradas D y control son manejados por el probador mientras que las demás ya están predefinidas en la arquitectura del circuito, la entrada de control actúa como el selector.

Entradas

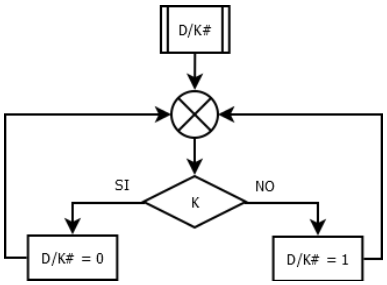
- D: Son los paquetes provenientes de la capa **DLL**, estos consisten en palabras de 8 bits que serán redirigidas al byte-striping.
- Control: Start y End son palabras de control o tipo "K", estos se agregan al inicio y final de un paquete **TLP** y **DLLP** de esta forma el receptor puede reconocer fácilmente el inicio y el final de un paquete. Hay dos tipos de paquetes de inicio, el STP[3] y el SDP[4] y dos tipos de paquetes de final, el END[5] y el EDB[6].
- Order set: Son secuencias de comandos (en múltiplos de 4 comandos) que inician con la señal de control **COM** usados para propósitos específicos. Se transmiten a través de las siguientes condiciones:
- Secuencia lógica de reposo (**IDLE**): Cuando no hay paquetes a transmitir, antes que dejar el lane flotando o no transmitir nada, se pasan carácteres nulos es decir 00h[7].

Salidas

- **D/K#**: Salida combinacional, indica el tipo de datos que se están transmitiendo por el MUX, si son tipo **D** la salida es 1, si son tipo **K** es 0. En la Figura 6 se ve claramente su comportamiento.

[3]Start cadena TLP
[4]Start cadena DLLP
[5]Cuando el TLP o el DLLP finalizaron correctamente.
[6]Cuando ocurrió un error en la cadena TLP.
[7]00H es el estándar para el carácter nulo representado por el valor cero.

Figura 6

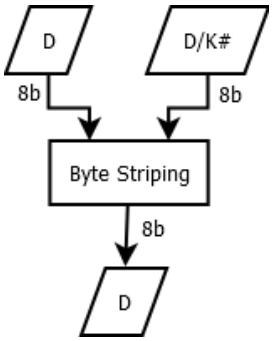


- **Out**: Salida combinacional, transmite los datos tipo D.

2.2. Byte striping

Este módulo se encarga de asignar los bytes a los lanes, esta asignación se hace en orden, del lane 0 al lane 3, en la Figura 9 se muestra un diagrama de bloques básico del diseño.

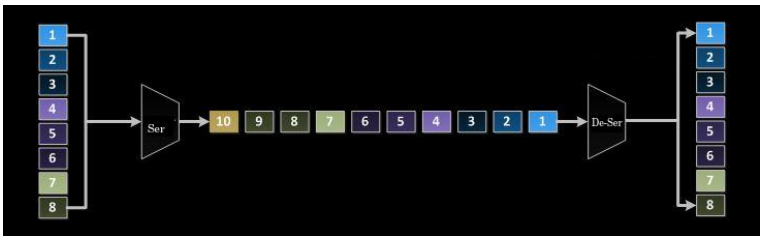
Figura 7. Diagrama de bloques del byte striping.



2.3. Serializador-De-serializador

Es el módulo encargado de la serialización de los datos, en este se tiene una entrada en paralelo y se genera una salida serializada para poder ser transmitida por el transmisor, junto con una señal de enlace o unión para los dos módulos. El módulo el des-serializador trabaja a la inversa, creando paquetes de 8 bits estos módulos constan de dos entradas y una salida. Su funcionamiento se puede evidenciar en la figura Figura 8.

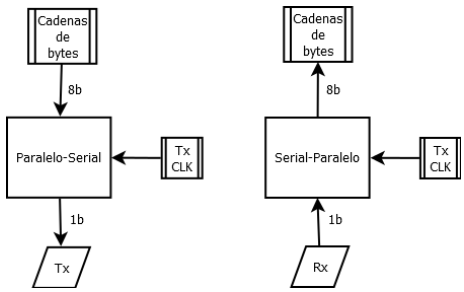
Figura 8. Imagen tomada de *PCI Express system architecture.*



Específicamente el módulo Serializador posee las siguientes entradas y salidas.

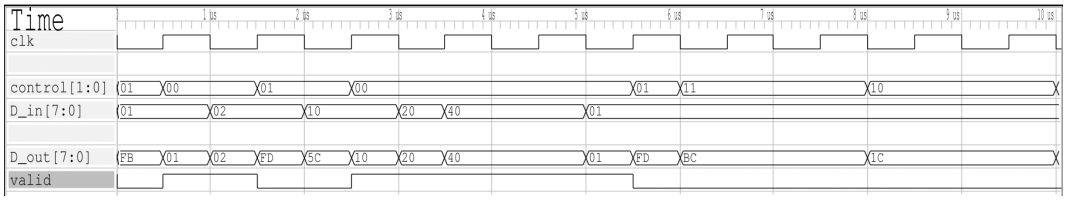
- Entradas:
  - clk: Reloj del sistema (250MHz).
  - enb: Comienzo de la transmisión. Cuando está a 1, se captura el carácter que entra por data y se empieza a enviar de manera serial.
  - data: Dato de 8 bits a enviar.
- Salida:
  - out: Salida serie del dato. Conectar a la línea de transmisión.
  - DK: Salida serie del dato. Conectar a la línea de transmisión.

Figura 9. Diagrama de bloques del serializador.



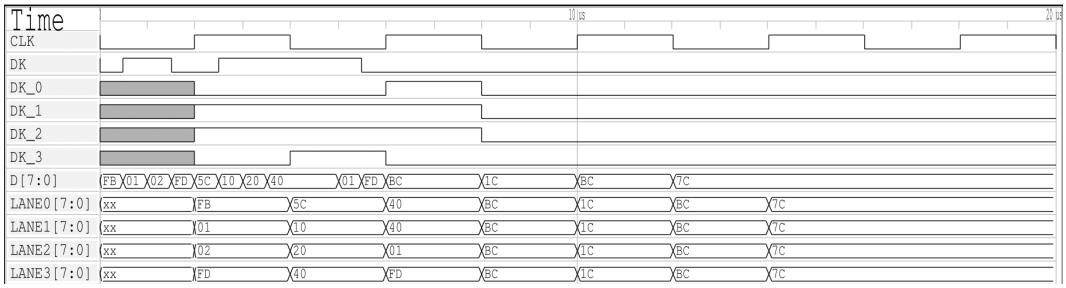
3. EJEMPLOS DE LOS RESULTADOS

Figura 10. MUX



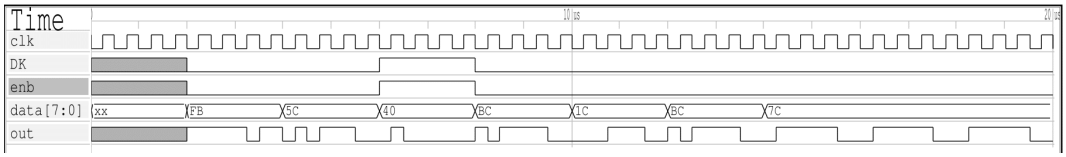
Entradas y salidas del módulo mux, note en la señal D_out como se adjuntan los códigos de control provenientes de D_in.

Figura 11. Byte Striping



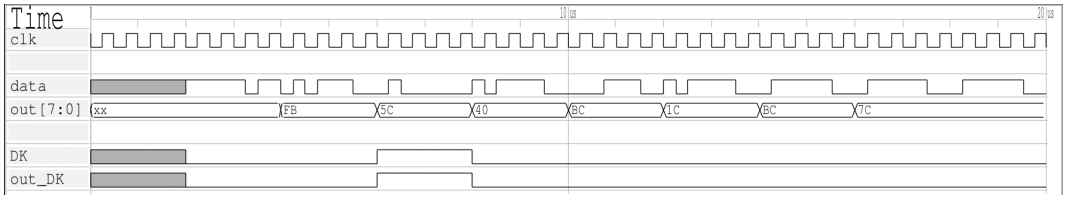
Entradas y salidas del módulo byte striping, los lanes se sincronizan en los flancos positvos y negativos del reloj.

Figura 12. Serializador



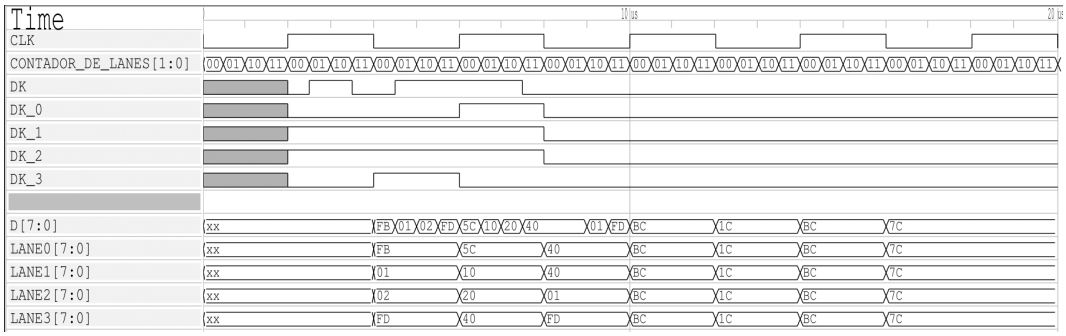
Entradas y salidas del módulo serializador, la señal OUT muestra la serialización de DATA, esto se logró implementando un contador de 0 a 7 que dividiera la señal CLK en 8 partes iguales y cada una de esas partes toma 1 bit de la señal de entrada.

Figura 13. Des-serializador



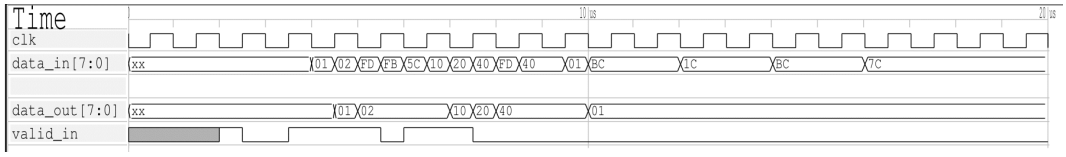
Entradas y salidas del des-serializador, es idéntico al serializador pero a la inversa, con la diferencia de que la señal OUT se actualiza únicamente en los flancos positivos y negativos de CLK.

Figura 14. Byte unstriping



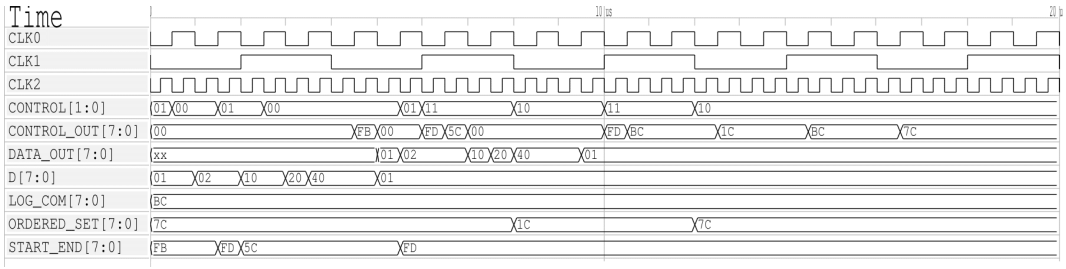
Entradas y salidas del byte unstriping, su comportamiento es idéntico al byte striping, con la salvedad de que la salida D se actualiza 8 veces en cada ciclo CLK.

Figura 15. Demultiplexor



Entradas y salidas del demultiplexor, la salida data_out muestra los datos de prueba pero con desfases en la frecuencia, ese cambio se debe al buffer elástico que no se implementó.

Figura 16. Probador



Entradas y salidas de la versión conductual final de la capa física del PCIE, note especialmente como los datos transmitidos por DATA_OUT son los mismos de la entrada D salvo por el desfase de frecuencia, mismo que se debe a la falta del buffer elástico que no se diseñó para el proyecto.

4. UTILIZACIÓN DEL CÓDIGO

- Tener la carpeta **Proyect1** suministrada y ubicarse en ella.
- Ya ubicado en la raíz escriba uno de los siguientes tres comandos:
  - make conductual
  - make sintetizado
  - make clean

Los nombres de los comandos son descriptivos de su funcionamiento.

5. CONCLUSIÓN Y RECOMENDACIONES

EL proceso de diseño conductual se llevó a cabo correctamente, siendo los principales logros del proyecto:

- Se logró desarrollar la interface de comunicación mediante verilog.
- Se presentaron descripciones de los módulos mux,demux, bytestriping, Serializador y deserializador.
- Escritura de una prueba que demostró el funcionmiento del módulo final del proyecto.
- Demostrar que el diseño a través del paradigma conductual tiene la trivialidad que aparenta.

Sin embargo el principal problema que se tuvo en el proyecto fué la relacionala con la sintetización en Yosys, puesto que no basta con tener un código conductual funcional sino que es necesario cumplir una serie de requisitos lógicos necesarios para que Yosys despliegue el circuito esperado, por ejemplo:

- Latch inferidos: Principal verdugo y el más complicado de identificar en un principio, para evitarlo es necesario verificar que en los bloques combinacionales no hayan lazos retroalimentados, además también deben cubrirse TODOS los estados posibles de las entradas.
- Verificar si el comando es sintetizable, por ejemplo, el código REPEAT[8] no se sintetizará (aunque el código conductual compile y se comporte perfectamente) y en consecuencia se imprimirá un error en la terminal.

Los problemas mencionados pudieron ser corregidos $Yosys compila el archivo PCIE_y osys.vs in problemas$ pero no se pudo conseguir que la salida se comportara como lo hace en el modelo conductual, debido a causas desconocidas. Para finalizar queda decir que el proyecto aportó gran cantidad de conocimiento en HDL, se pudo ver que un diseño lógico funcional no necesariamente se traduce en un código sencillo y como es indispensable tener en cuenta aspectos lógicos intrínsecos no triviales de los dispositivos digitales.

[8] En lugar de WAIT se puede utilizar FOR que si es sintetizable por Yosys.