# Universidad de Costa Rica
# Escuela de Ingeniería Eléctrica

IE-0523 Circuitos Digitales II

## Proyecto I: Capa física de PCIe
## Grupo 3

Profesor:
Jorge Soto.

Gokeh Ávila Blanco - B50747
David Campos Espinoza - B51479
Isaac Fonseca Segura - B52786
Andrés Vega Zamora - B57739

13 de junio de 2021

# 1. Resumen

En el presente proyecto se trabajó la implementación a nivel de RTL y síntesis de la capa física del estándar de comunicaciones PCIe (Peripheral Component Interconnect express), el cual es un estándar de conexión de alta velocidad para componentes que usamos todos los días como tarjetas de Wi-Fi, tarjetas de vídeo o discos de estado sólido. Este estándar logra velocidades desde los 8 GB/s hasta los 256 GB/s hoy en día [1].
En breve, PCIe permite comunicar a altas velocidades componentes con el procesador o con otros componentes, como tarjetas y periféricos, por medio de un protocolo de arbitraje que permite el control de flujo.
Una peculiaridad de nuestro diseño es contar con señales de reset y la posibilidad de recibir datos tan sólo 4 ciclos de reloj después de haber sido enviados.
Se realizaron pruebas individuales para cada módulo de la arquitectura, esto para asegura el correcto funcionamiento del diseño una vez que todo se uniera. Las pruebas consistieron en generar un estímulo a las entradas y observar que el comportamiento a la salida fuera congruente con el establecido en el diseño. Algunos problemas que surgieron se dieron a nivel de síntesis a la hora de no contar con señales de reset, los cuales luego se solucionaron. Otros problemas tuvieron que ver con condiciones de reloj, los cuales se arreglaron usando flip flops de flanco negativo a lo interno del diseño para mantener un dato, pero la salida siempre se reporta con flip flops de flanco positivo.
Se logró observar más claramente la importancia y utilidad de los comandos AUTOTEMPLATE,AUTOINST y AUTOWIRE para realizar la instanciación de módulos en proyectos grandes, también se logró trabajar exitosamente en grupo a la hora de organizarse y de resolver eventuales problemas en el código. Como recomendaciones se puede mencionar que cuando se hacen trabajos grandes es importante poner nombres claros a las variables por facilidad a la hora de armar los módulos finales. Y por último es recomendable el uso de herramientas como las antes mencionadas de AUTOINST, AUTOTEMPLATE Y AUTOWIRE, así como un uso de un control de versiones como github y hacer uso de makefiles.

# 2. Descripción arquitectónica

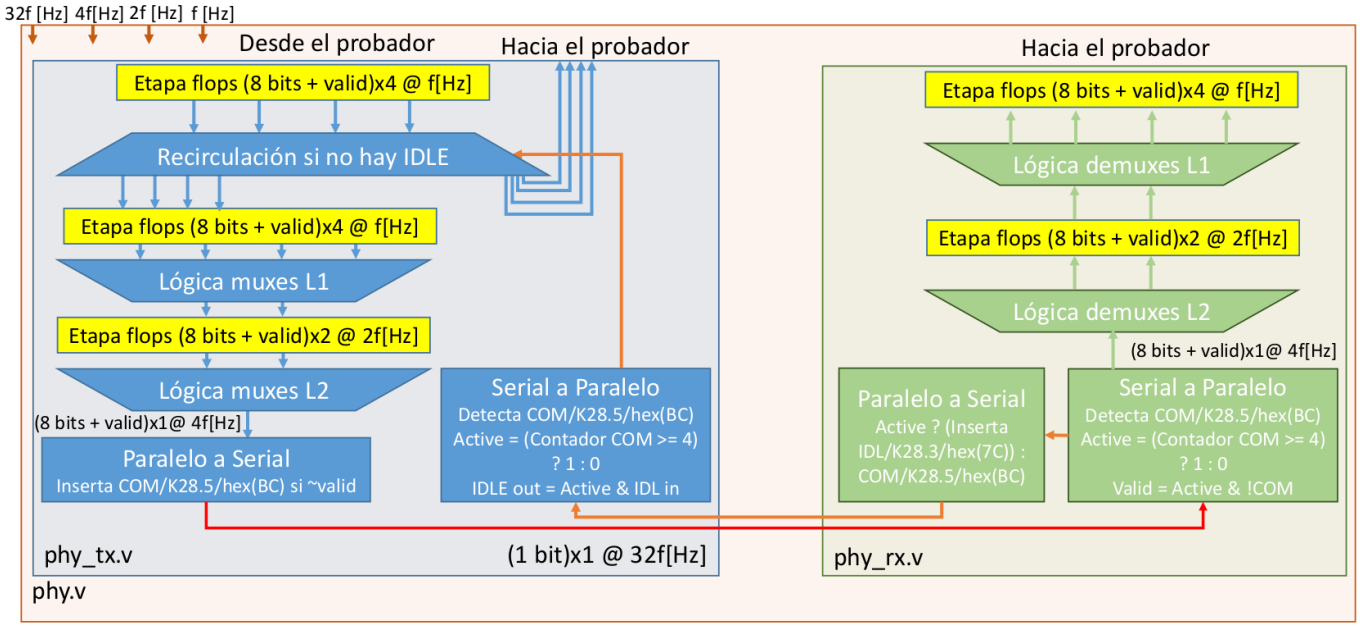


Figura 1: Diagrama de bloques de los módulos

Basados en la arquitectura mostrada en la Figura 1, se describe el funcionamiento de los módulos que componen la misma. El módulo exterior phy.v contiene dentro de sí al módulo transmisor phy_tx.v y el módulo receptor phy_rx.v.

El módulo phy_tx.v recibe desde el probador 4 buses de datos de 8 bits cada uno a una frecuencia base de f Hz, con una señal de valid asociada a cada bus de datos. El primer bloque donde entran dichas señales corresponde al **Recirculador**. A partir de este bloque los datos pueden seguir dos caminos posibles dependiento del valor de IDLE que reciba el recirculador. Si IDLE = 0, entonces recircula los datos hacia el probador; de lo contrario si IDLE = 1, los datos continúan hacia la lógica de muxes.

La **lógica de muxes** consta de 2 niveles, el primero recibe los datos mencionados anteriormente y los convierte en 2 buses de 8 bits cada uno a una frecuencia de 2f Hz, manteniendo el valor de los valid. Se utiliza un selector automático que va alternando entre los buses de entrada para ordenar los datos en los buses de salida. El segundo nivel de los muxes aplica la misma lógica para su funcionamiento, pero su salida corresponde a 1 bus de 8 bits a frecuencia 4f Hz.

El bus final de la lógica de muxes se conecta con el bloque **paralelo a serial**, este se encarga de convertir el bus de 8 bits en una salida serial de 1 bit a una frecuencia de 32f Hz. El funcionamiento es muy similar a la lógica de muxes, por el hecho de que cuenta con un selector automático que define cual bit de entrada corresponde ordenar en la salida. Posee una lógica adicional encargada de que al recibir el valid en 0, la salida no corresponde a los datos de entrada, sino al código COM (BC en hexadecimal). La salida de este bloque se conecta al bloque serial a paralelo del módulo receptor phy_rx.v.

El último bloque del módulo transmisor phy_tx.v corresponde al **Serial a Paralelo**. Dicho bloque recibe una entrada de datos seriales del módulo phy_rx. Posee un contador que al llegar a 4 códigos BC, asigna un alto al Active. La salida del bloque corresponde a un AND entre Active y IDL in. El

valor de IDL in se pone en alto cuando se recibe el código hexadecimal 7C. Esta salida es la que controla el flujo de datos del recirculador.
El módulo receptor phy_rx.v recibe los datos en el bloque **Serial a paralelo**, la entrada corresponde a la señal serial proveniente del bloque paralelo a serial de phy_tx.v. La lógica del bloque incluye un Active que sigue el mismo comportamiento que en el bloque transmisor. Una de las salidas corresponde al valid que acompaña el bus de datos de salida, el valid se mantiene en alto mientras el Active está en alto y en los datos de entrada no se reciba un BC. Además el bloque se encarga de convertir la señal serial de entrada (32f Hz) en un bus de salida de 8 bits a una frecuencia de 4f Hz, utilizando un selector de salida automático para ordenar los datos en el bus de 8 bits y enviarlos a la lógica de demuxes. El active es enviado al bloque de paralelo a serial.
El bloque **paralelo a serial** del módulo receptor recibe la señal active y se encarga de enviar de manera serial el código IDLE si el active está en alto y el código COM si el active está en bajo. Corresponde a la entrada del serial a paralelo del módulo phy_tx.v
Finalmente tenemos la **lógica de demuxes**, funciona de manera inversa a la lógica de muxes del transmisor. Recibe un bus de 8 bits a 4f Hz con su señal de valid correspondiente, en su primer nivel, utilizando un selector de salida automático genera 2 buses de 8 bits a 2f Hz. En el segundo nivel toma esos dos buses y aplicando el mismo principio de funcionamiento genera 4 buses de 8 bits a f Hz, cada uno con se señal de valid asociada y los envía hacia el probador.

# 3. Plan de pruebas

## 3.1. Recirculador

Para este módulo se espera probar con una entrada de datos aleatorios desde el probador, y luego ir variando la señal de IDLE para poder observar en las salidas del recirculador si la entrada se está yendo hacia la lógica de muxes o si está volviendo al probador, es decir, siendo recirculada.

## 3.2. Lógica de muxes

Para probar el funcionamiento de la lógica de muxes se busca generar señales como las mostradas en la Figura 2.

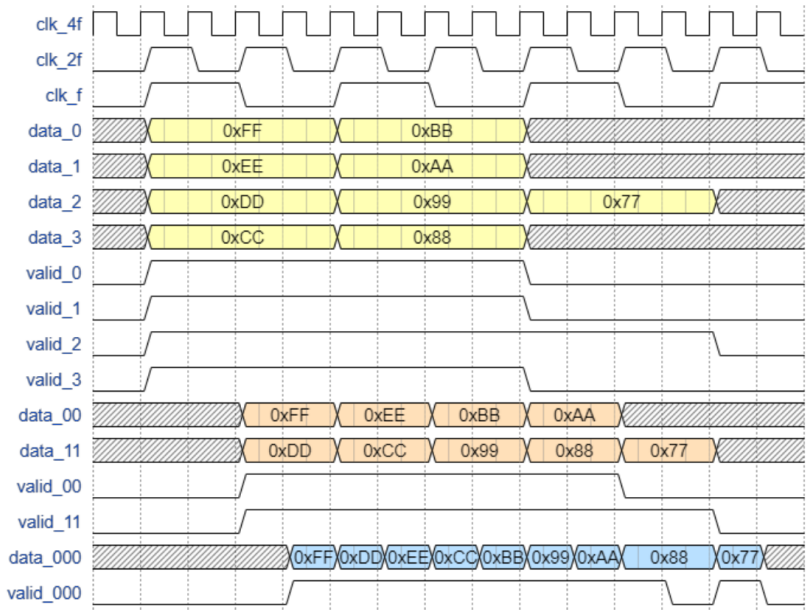


Figura 2: Ejemplo de funcionamiento esperado de lógica de muxes. [2]

La prueba consiste en verificar que los datos de entrada recibidos por el bloque pasen a 2 buses con el doble de frecuencia en el primer nivel. El primer bus toma los datos de entrada 0 y 1 y el segundo los datos 2 y 3. Además si el valid de la señal que está leyendo no está activo, mantiene el último valor en memoria, sin importar si cambian los datos que se están recibiendo. Los dos buses de datos de salida del primer nivel se acompañan con su señal de valid correspondiente, generada a partir de los valid de los datos de entrada. En el segundo nivel se sigue el mismo funcionamiento pero se pasa a un bus al doble de la frecuencia de la salida de primer nivel. Igualmente acompaña el bus con un valid generado a partir de los 2 valids de primer nivel.
El diseño cumplió con todas las especificaciones funcionales de la prueba.

## 3.3. Paralelo a serial phy_tx

Para la prueba del bloque paralelo a serial del transmisor se quieren verificar tres especifiaciones de funcionamiento.

- Al recibir una entrada de 8bits a 4f Hz, se obtenga una salida serial (1 bit) a 32f Hz que represente de manera ordenada los datos recibidos en la entrada.

- Si el valid de entrada está en bajo, la salida serial corresponde al código COM (BC en hexadecimal) y no al equivalente de los datos paralelos de la entrada.
- Si el reset está en cero, la salida corresponde a cero.

Inicialmente se recibe el primer dato de entrada con el reset en 0 y el valid en 1. Luego el reset se pasa a 1 (y se mantiene el resto de la prueba) y se recibe el segundo dato de entrada. Para el tercer dato de entrada el valid baja a 0. Finalmente, para el cuarto y quinto dato de entrada el valid vuelve a 1.
El diseño cumplió con las especificaciones funcionales listadas anteriormente.

## 3.4. Serial paralelo phy_tx

El bloque serial paralalo phy_tx requiere como entrada la señal de idle que es enviada por un módulo del receptor. Es este caso, para las pruebas del bloque se pretende que se recibe la entrada del IDLE y conforme esta variable cambia una vez que se han hecho el conteo de 4 BC la salida IDLE_OUT se envía hacia al bloque del recirculador para que los datos puedan ser transmitidos a través de los multiplexores.

## 3.5. Phy_tx

El módulo phy_tx.v corresponde a la unión de los bloques de Recirculador, Lógica de muxes, Paralelo a serial y Serial a paralelo. Debido a que cada uno ha sido probado individualmente, lo que se debe probar para este módulo es que funcione la comunicación entre dichos bloques.
Se verifica que cuando el bloque de serial a paralelo envía el IDLE = 0, el recirculador envíe los datos de entrada de vuelta al probador y no a la lógica de muxes. Caso contrario cuando el bloque serial a paralelo envía IDLE =1, el recirculador envíe los datos de salida hacia la lógica de muxes. Recibidos los 4 buses de datos a f Hz en los muxes, deben salir 1 bus de datos a 4f Hz hacia el bloque paralelo a serial. Finalmente que el bloque de paralelo a serial tenga como salida el equivalente en serial a 32f Hz del bus de datos de entrada.

## 3.6. Lógica de demuxes

Para comprobar el correcto funcionamiento de la logica de demuxes se busca generar señales como las mostradas en la Figura 3.

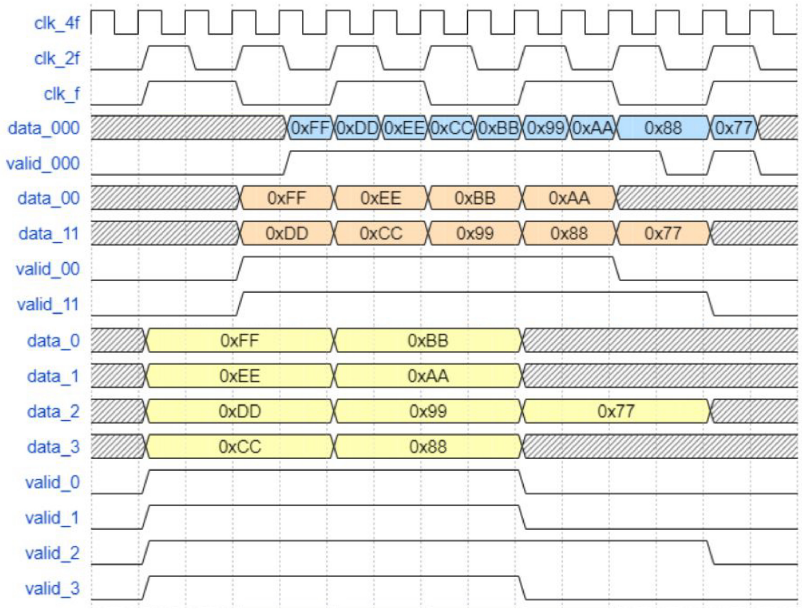


Figura 3: Ejemplo de funcionamiento esperado de lógica de demuxes. [2]

En la Figura anterior podemos ver como la idea de la logica de demuxes es recibir primero en el nivel 1, logica demuxes L2, un bus de 8 bits más 1 valid a una frecuencia de 4fHz, y lo que debe de salir de esta primera sección de la lógica de demuxes son dos buses de datos de 8 bits más 2 valids y la mitad de la frecuencia que le entró a la lógica demuxes L2. Luego para la segunda sección de esta lógica de demuxes, la cual corresponde a la caja llamada: Lógica demuxes L1 de la Figura 1, tenemos que esta recibe la salida de la lógica demuxes L2, es decir, 2 buses de datos de 8 bits más 2 valid a 2fHz, y su función es demultiplexar dichos datos en una salida que corresponda a 4 buses de 8 bits más 4 valids a la mitad de la frecuencia de entrada, como se puede ver en la Figura 3.

## 3.7. Serial paralelo phy_rx

Este el primer bloque de interés en el funcionamiento del bloque phy_rx, ya que es el que recibe una entrada de datos serial y reordena estos datos a paralelo, es decir toma una secuencia serial de 8 bits y los reordena en paralelo para poder ser leídos como un solo dato.
Ahora, estos datos entrantes no serán válidos hasta que haya entrado una secuencia de cuatro hex(BC), que es cuando el módulo se reportará como activo con la señal de active y además indicará que el siguiente dato es válido. Esto es importante porque en la entrada serial no hay información sobre la validez de los datos.
Entonces para probar este módulo es necesario una entrada aleatoria de datos en la cual haya en algún punto cuatro hex(BC) continuos, o bien separados. Lo cual genere que el módulo levante la señal de active y luego para el siguiente dato después del último hex(BC) levante la señal de valid para este.

## 3.8. Paralelo serial phy_rx

Este módulo se encarga de de informar al módulo transmisor sobre el estado actual para que el transmisor pueda generar correctamente su señal de IDLE. Su funcionamiento consiste en el siguiente.

Si detecta active envía un hex(7C) (IDLE) de forma serial, si no detecta activo envía un hex(BC) (COM) de forma serial. Estos cambios los lee en clock $1f$.
Entonces lo que se necesita para probarlo es una señal de active que esté cambiando y observar qué pone el módulo a la salida, para comprobar si cumple el comportamiento descrito anteriormente.

## 3.9. Phy_rx

Para probar este módulo correctamente sólo es necesario tener una entrada de datos aleatoria que algún punto contenga, ya sea consecutivamente o aleatoriamente, cuatro señales hex(BC). Una vez obtenido esto a la salida deberían obtenerse resultados congruentes con los de la lógica de demuxes basados en la entrada serial *después* de los cuatro hex(BC).

## 3.10. Phy

Para la comprobación del funcionamiento de este bloque se utiliza el autoinst para ambos módulos phy_tx y phy_rx, dónde se debe hacer el cambio para las variables idle y los datos que son recibidos en el módulo rx. Para probar el funcionamiento de la salida del recirculador así cómo del módulo phy_rx es necesario generar varias entradas con diferentes señales de valid para que los dos escenarios se presenten y en cuánto se reciben los 4 BC los datos son transmitidos y se evidencia cómo tienen un retardo en comparación a la entrada por tratarse de lógica secuencial.

# 4. Instrucciones de utilización de la simulación

## 4.1. Recirculador

- Directorio: ../../Proyecto-Digitales-II-G3/phy_tx/recirculador
- Correr el comando `make gtk`, el cual compilará todos los archivos necesarios para la simulación, generará la síntesis y hará los cambios respectivos, y luego abrirá los resultados en GTKWave.

## 4.2. Lógica de muxes

- Directorio: ../../Proyecto-Digitales-II-G3/phy_tx/logica_muxes
- Para correr la simulación, se debe abrir en terminal el directorio especificado y escribir el comando "make". Este ejecuta las subrutinas "yosys", "sed" y "verilog" del makefile, en el orden respectivo. De tal manera que se sintetiza la descripción estructural, luego se cambian los nombres de los módulos sintetizados para evitar choques y luego se compila el código en Icarus Verilog y se abre el archivo .gtkw para mostrar los resultados obtenidos.

## 4.3. Paralelo a serial phy_tx

- Directorio: ../../Proyecto-Digitales-II-G3/phy_tx/paralelo_serial
- Para correr la simulación, se debe abrir en terminal el directorio especificado y escribir el comando "make".

## 4.4. Serial paralelo phy_tx

## 4.5. Phy_tx

- Directorio: ../../Proyecto-Digitales-II-G3/phy_tx
- Para correr la simulación, se debe abrir en terminal el directorio especificado y escribir el comando "make".

## 4.6. Serial paralelo phy_rx

- Directorio: ../../Proyecto-Digitales-II-G3/phy_rx/serial_paralelo_rx/
- Correr el comando `make gtk`, el cual compilará todos los archivos necesarios para la simulación, generará la síntesis y hará los cambios respectivos, y luego abrirá los resultados en GTKWave.

## 4.7. Paralelo serial phy_rx

- Directorio: ../../Proyecto-Digitales-II-G3/phy_rx/paralelo_serial_rx/
- Correr el comando `make gtk`, el cual compilará todos los archivos necesarios para la simulación, generará la síntesis y hará los cambios respectivos, y luego abrirá los resultados en GTKWave.

## 4.8. Lógica de demuxes

- Directorio: ../../ Proyecto-Digitales-II-G3-main/phy_rx/logica_demuxes

- Para correr la simulación, se debe abrir en terminal el directorio especificado y escribir el comando "make", el comando anterior ejecuta las subrutinas yosys, sed y verilog del makefile en dicho orden. De tal manera que se sintetiza la descripción estructural, luego se cambian los nombres de los módulos sintetizados para evitar choques y luego se compila el código en Icarus Verilog y se abre el archivo .gtkw para mostrar los resultados obtenidos.

### 4.9. Phy_rx

- Directorio: ../../Proyecto-Digitales-II-G3/phy_rx/
- Correr el comando `make gtk`, el cual compilará todos los archivos necesarios para la simulación, generará la síntesis y hará los cambios respectivos, y luego abrirá los resultados en GTKWave.

### 4.10. Phy

- Directorio: ../../Proyecto-Digitales-II-G3/phy/
- Correr el comando `make all`, el cual compilará todos los archivos necesarios para la simulación, generará la síntesis y hará los cambios respectivos, y luego abrirá los resultados en GTKWave.

## 5. Resultados

### 5.1. Recirculador

En la Figura 4 se muestra a la entrada una serie de datos aleatorios con sus respectivas señales de valid. Además se muestra el clock $1f$ ya que los datos ingresan a esta frecuencia al recirculador. Lo más importante es prestar atención a la señal de *idle* la cual varía en dos ocasiones durante la simulación. Se observa claramente que cuando la señal de idle está en bajo los datos de la entrada pasan a las salidas de recirculación, mientras que cuando la señal de idle pasa a alto la entrada pasa hacia las salidas que van a la lógica siguiente, en este caso la etapa de multiplexación. Cabe notar que no hay interacción con el valid en esta etapa, el módulo simplemente pasa la entrada, no interactua con esta.

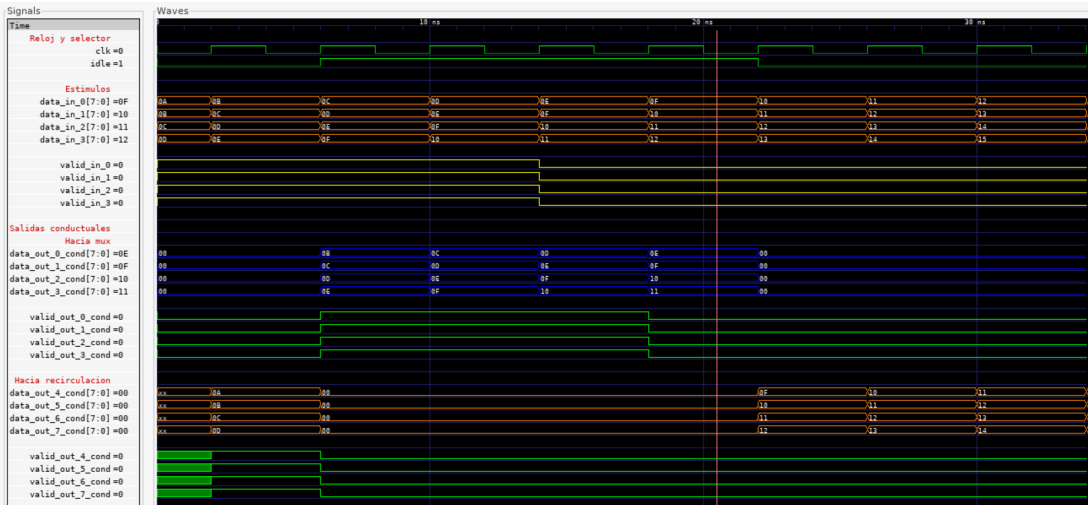


Figura 4: Resultado de pruebas en el recirculador.

## 5.2. Lógica de muxes

En la Figura 5 se muestra que los resultados obtenidos de la lógica de muxes lograron reproducir lo deseado en la Figura 2, por lo tanto se cumple la prueba planteada y se determina que el diseño funciona.

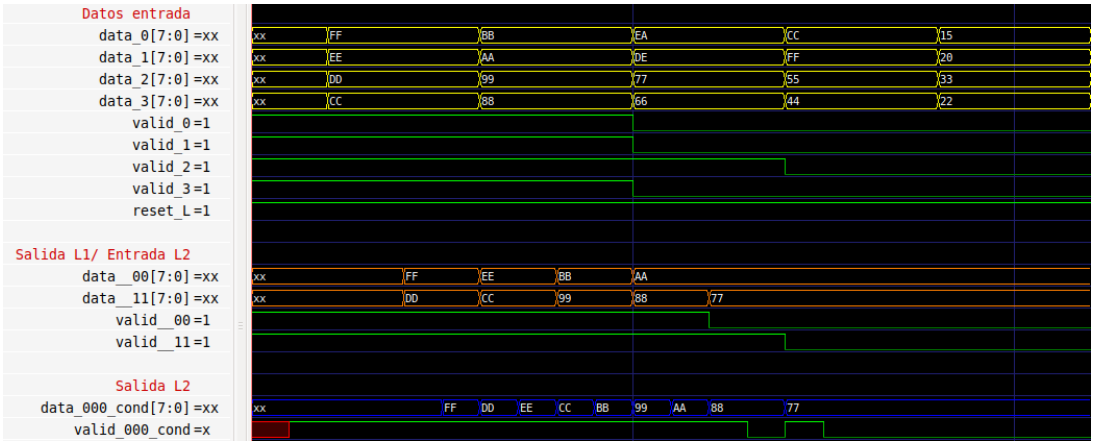


Figura 5: Resultados de la lógica de muxes

## 5.3. Paralelo a serial phy_tx.v

En la Figura 6 se muestra que los resultados de la prueba al bloque serial paralelo del transmisor, cumplen con las especificaciones funcionales planteadas en el plan de pruebas.

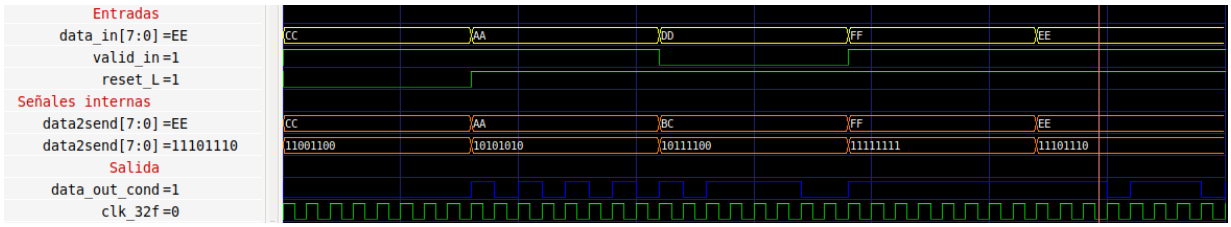


Figura 6: Resultados del paralelo_serial del transmisor

## 5.4. Serial paralelo phy_tx

En la Figura 7 se muestra que los resultados de la prueba al bloque serial paralelo del transmisor, cumplen con las especificaciones funcionales planteadas en el plan de pruebas.

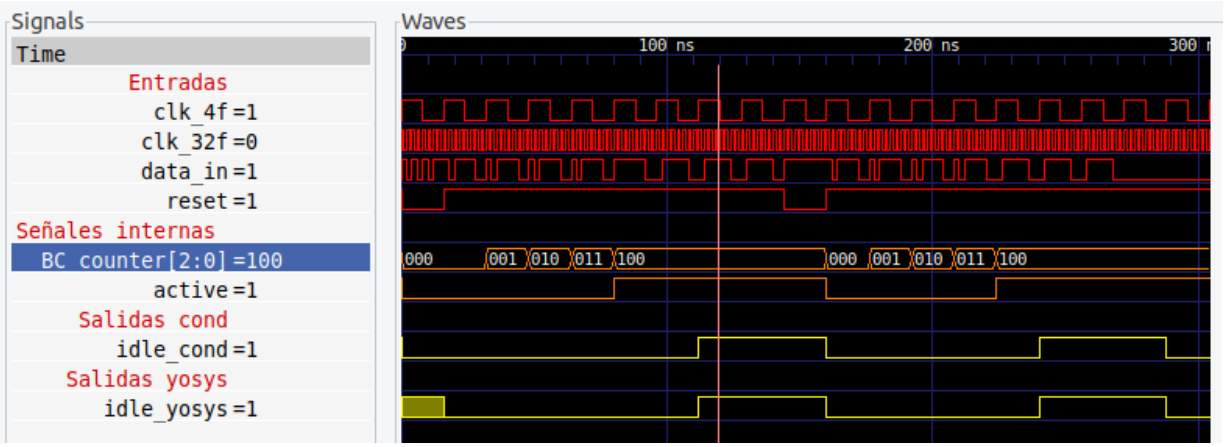


Figura 7: Resultados del serial_paralelo del transmisor

## 5.5. Phy_tx

En las Figuras 8 y 8 se muestran que los resultados de la prueba al bloque completo del modulo phy_tx, cumplen con las especificaciones funcionales planteadas en el plan de pruebas. Se puede apreciar como las entradas que son recibidas en tramas de 8 bits se envian a una mayor frecuencia y en solo 1 bit.

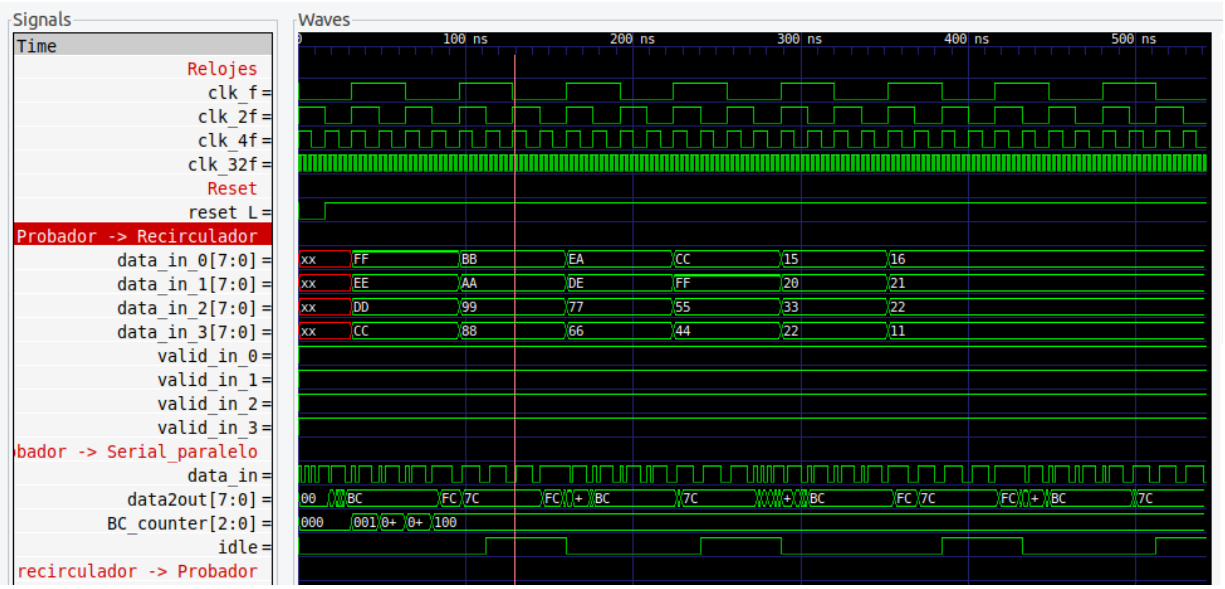


Figura 8: Resultados del módulo phy_tx

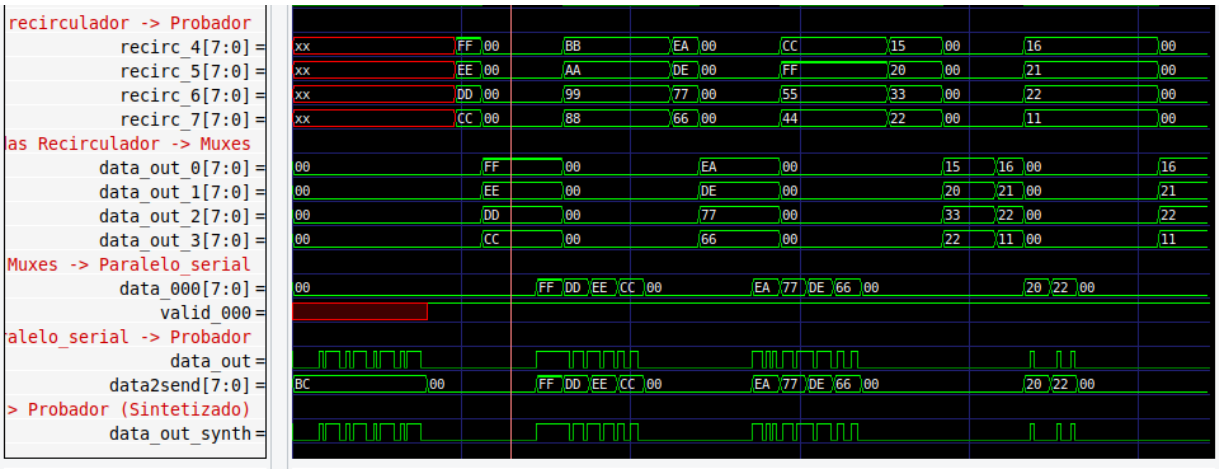


Figura 9: Resultados del módulo phy_tx

## 5.6. Serial paralelo phy_rx

En la Figura 10 se observa donde al principio la señales de active y valid están en bajo, el módulo recibe datos aleatorios y los pasa como no válidos. Luego llegan cuatro hex(BC) los cuales activan la señal de active, requerida en el módulo paralelo_serial_rx, y seguidamente en el próximo flanco positivo se activa el valid para el dato entrante, ya que este sí es valido y se pasa como tal.

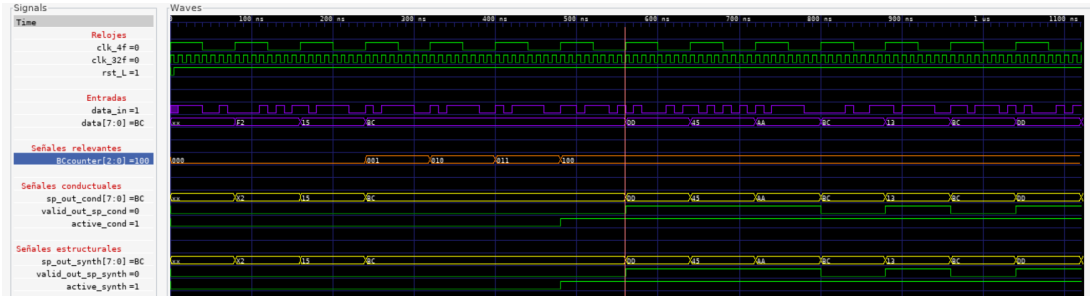


Figura 10: Resultados del serial_paralelo del módulo receptor

## 5.7. Paralelo serial phy_rx

En la Figura 11 se muestra que cuando en el flanco positivo del reloj $1f$ se detecta active a la salida se comunica serialmente una señal hex(7C), mientras que cuando en el flanco positivo no se detecta active entonces a la salida se comunica serialmente una señal hex(BC).

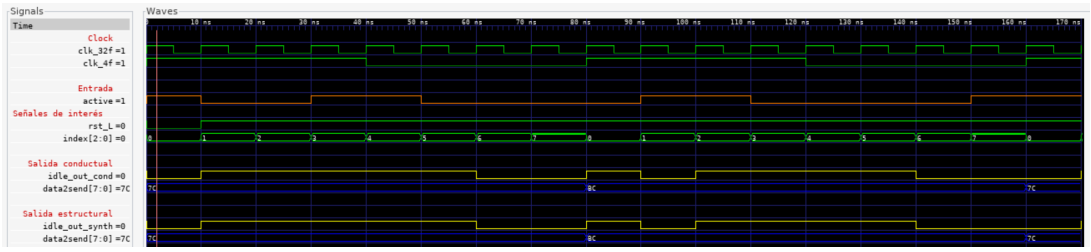


Figura 11: Resultados del paralelo_serial del módulo receptor

## 5.8. Lógica de demuxes

En laFigura 12 podemos ver como el resultado que muestra el gtkwave es análogo al de laFigura 3, con lo cual se concluye que el diseño es funcional.

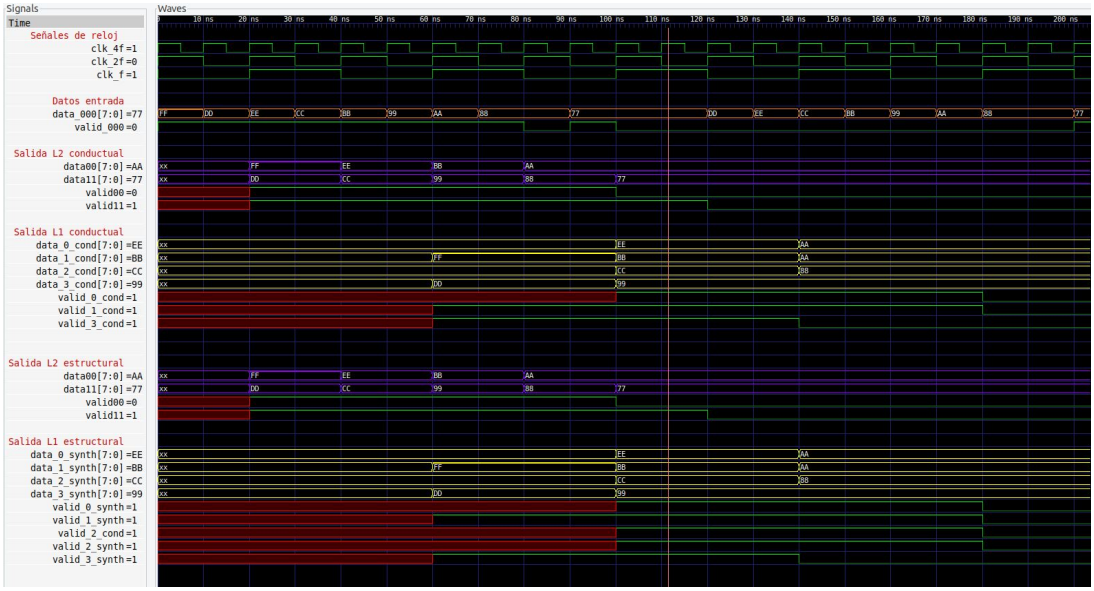


Figura 12: Resultados de la lógica de demuxes

## 5.9. Phy_rx

En la Figura 13 se observa una entrada de datos seriales aleatorios, los cuales en cierto punto contienen una cadena de cuatro hex(BC). Se observa claramente que antes de este evento la salida pasa las entradas como no válidas. En este caso como antes no se había activado un evento las salidas están en condición de no importa. Se observa también que la señal de IDLE se encontraba transmitiendo hex(BC).
Una vez que llegan los cuatro hex(BC) (COM) la salida comienza a pasar los datos y los pasa como válidos dependiendo de qué dato sean (observar un hex(BC) por aparte que no pasa como válido y se replica el dato anterior). Además la salida de IDLE se pone en hex(7C) (IDLE).
Es importante observar que el orden de los datos a la salida es congruente con el orden establecido para la lógica de demuxes.

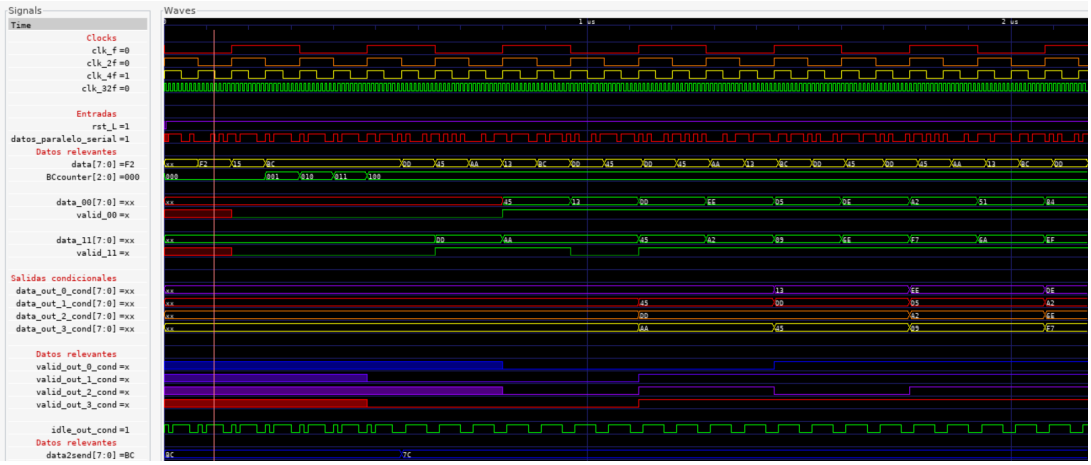


Figura 13: Resultados del módulo receptor phy_rx

## 5.10. Phy

En lasFiguras 14, 15 y 16 se pueden ver todos los valores obtenidos tanto de entrada como de salida para el modulo phy. Se reciben entradas que en un inicio son enviadas de vuelta al probador por medio del probador hasta que luego de un conteo de 4 BC se permite el paso de los datos a traves de los multiplexores asi como de los demuliplexores luego de pasar por un proceso de serialización.

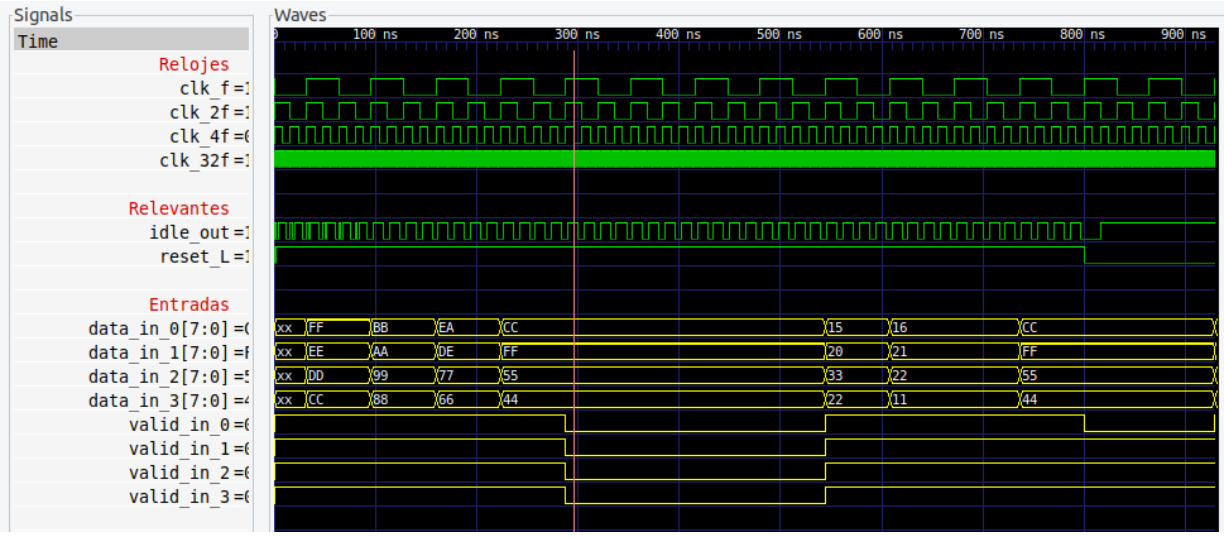


Figura 14: Resultados del módulo phy

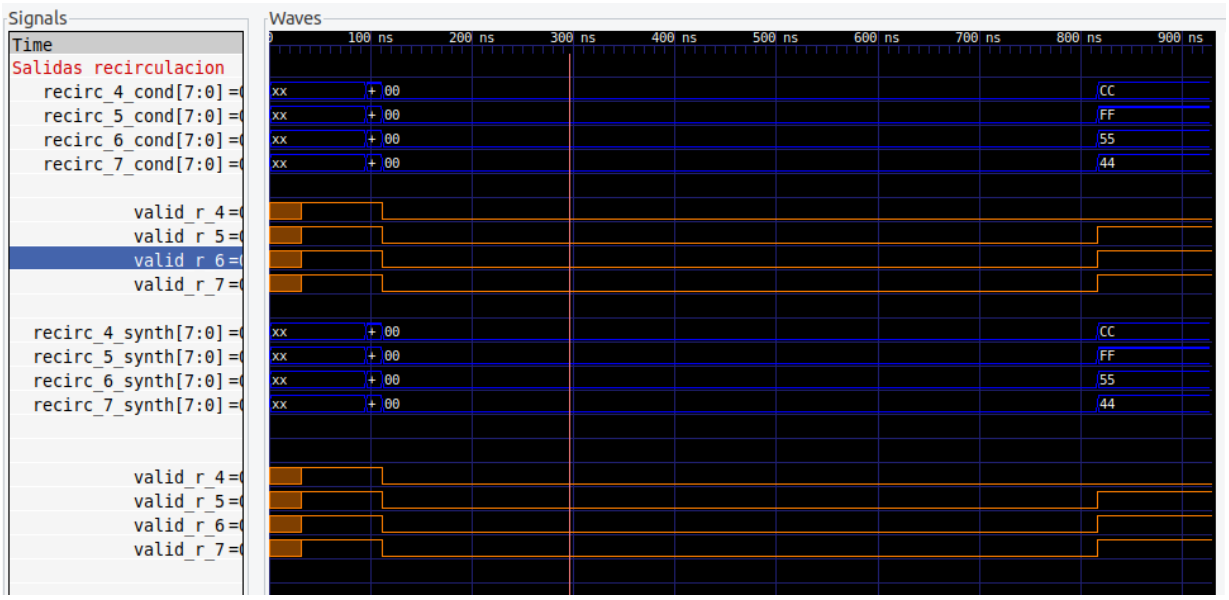


Figura 15: Resultados del módulo phy

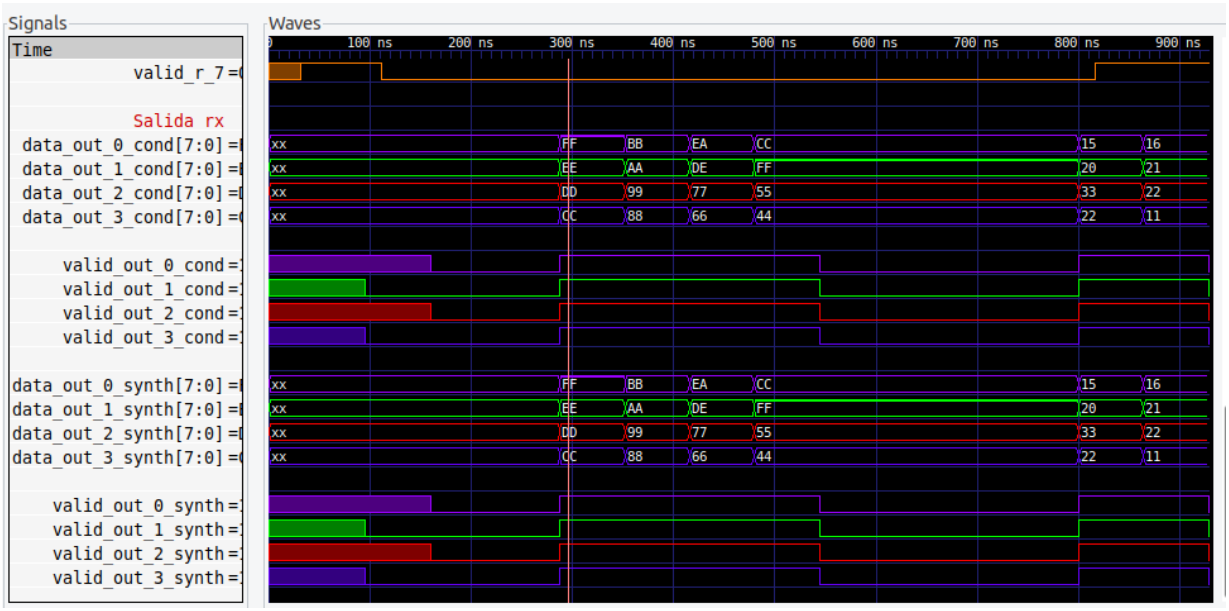


Figura 16: Resultados del módulo phy

# 6. Conclusiones y recomendaciones

## 6.1. Conclusiones

- Los comandos AUTOINST,AUTOTEMPLATE Y AUTOWIRE permite realizar instancias de módulos de manera más fácil y automatizada.
- El módulo paralelo-serial en conjunto con el módulo serial-paralelo permite enviar datos de varios bits por un solo canal, lo que a la hora de realizar un diseño físico se traduce en una menor cantidad de conexiones y por lo tanto produce un ahorro de material y dinero.

- Los módulos de demuxes y muxes son de gran utilidad para de trabajar con grandes buses de información, así como con distintos valores de frecuencia.
- se logró implementar métodos de comunicación asertiva para la división de tareas y de depuración de código. El equipo mostro buena sinergia entre sus integrantes frente a los distintos errores o pruebas que el proyecto planteo.
- Se logró conectar con éxito los módulos de transmisor y receptor para formar el phy.

### 6.2. Recomendaciones

- Cuando se distribuye el diseño de módulos separados en un grupo de trabajo, definir una convención para los nombres de señales de entrada y salida. Así se facilita la lectura del código y se evitan confusiones a la hora de conectar los módulos separados.
- Al sintetizar la descripción estructural de un módulo, asegurarse de cambiar el nombre del módulo sintetizado y todos los submódulos que lo componen. Para evitar choques en el banco de pruebas y que no funcione la simulación.
- Hacer un bosquejo del diseño a mano de la lógica antes de saltar a la codificación del diseño puede ahorrar mucho tiempo a la hora de intentar resolver problemas.
- Incluir señales de reset es necesario en algunos casos para generar una síntesis correcta. Especialmente en módulos con lógica secuencial.
- El uso de AUTOINST, AUTOTEMPLATE y AUTOWIRE facilitan mucho la instanciación de nuevos módulos.
- El uso de makefiles y make agiliza las pruebas rápidas para la verificación del correcto funcionamiento de la lógica.
- El uso de control de versiones, como github, agiliza el flujo de trabajo en equipo, ya que todos siempre tienen a disposición los últimos cambios hechos por los compañeros.

## 7. Plan de trabajo

Para el planteamiento del plan de trabajo se parte del diagrama de bloques que se obtiene de la presentación de la capa PHY para la interfaz PCIE de laFigura 1.
A partir del diagrama en la Figura 1 se desarrolla el proyecto primeramente dividiendo las etapas del transmisor y del receptor. Para la etapa del transmisor se trabajará inicialmente cada multiplexor así como el bloque de recirculación como un módulo aparte para comprobar el funcionamiento independiente de cada etapa. Luego, se le realizarán las pruebas utilizando como referencia los resultados de la Figura 2 para determinar el funcionamiento de cada etapa. Una vez determinado los módulos conductuales se procede a unir los módulos de los multiplexores para poder realizar la síntesis de YOSYS considerando todos los componentes.
Por otro lado, para la etapa del receptor se realizará de igual manera el módulo de cada demultiplexor por separado y luego de comprobar el funcionamiento de cada uno se unirán para realizar las síntesis. Después, se realizarán los módulos de paralelo a serial y de serial a paralelo dónde se tomará en cuenta las entradas y salidas esperadas para verificar el funcionamiento de cada módulo y realizar la síntesis de cada uno por separado. Se tomára como referencias las salidas mostradas en lasFiguras 17 y 18.
Para cada etapa realizada previamente se van a realizar el respectivo probador y banco de pruebas que luego se irán reduciendo conforme se unan los módulos hasta obtener un único probador y banco de pruebas utilizando AUTOINST como herramienta de apoyo.

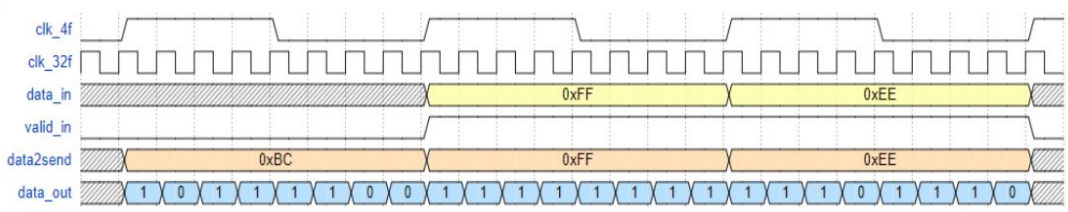


Figura 17: Salidas esperadas de conversión de paralelo a serial. [2]

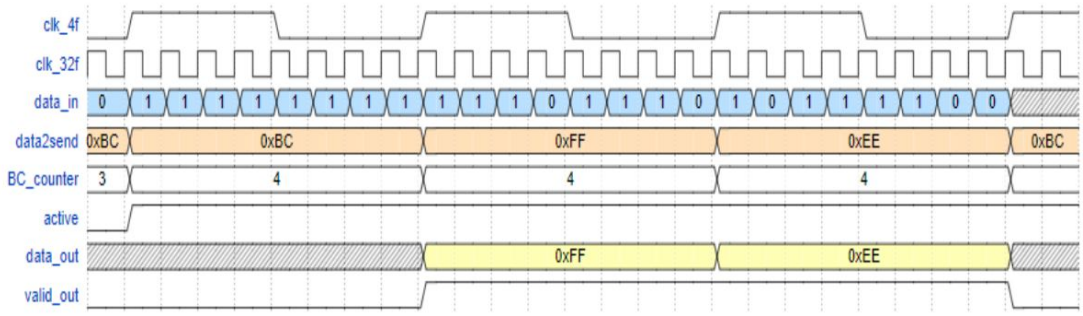


Figura 18: Salidas esperadas de conversión de serial a paralelo. [2]

# 8. Bitácora

**Avance #1: (25/5/2021)**
Se acordó trabajar en equipos de dos integrantes, donde David Campos y Andrés Vega forman un subgrupo y se encargan de trabajar el módulo phy_tx, y Gokeh Ávila e Isaac Fonseca forman otro subgrupo y se encargan de trabajar el módulo phy_rx.

*Asignaciones:*
- David Campos realizó el plan de trabajo.
- Andrés Vega realizó la lógica de muxes.
- Isaac Fonseca realizó la recirculación.
- Gokeh Ávila e Isaac Fonseca realizaron la lógica de demuxes.

*Nota:* para el primer avance faltó terminar bien la lógica de demuxes

**Avance #2: (1/6/2021)**
Se continuo trabajando en los mismos subgrupos.

*Asignaciones:*
- Andrés Vega realizó el paralelo a serial del módulo phy_tx.
- David Campos realizó el serial a paralelo del módulo phy_tx.
- Isaac Fonseca realizó el paralelo a serial del módulo phy_rx.
- Gokeh Ávila e Isaac Fonseca realizaron el serial a paralelo del módulo phy_rx.

*Nota:* para el segundo avance faltó terminar la síntesis del serial a paralelo del módulo phy_rx.

**Avance #3: (8/6/2021)**

*Asignaciones:*

- David Campos y Andrés Vega completaron el módulo phy_tx.
- Isaac Fonseca y Gokeh Ávila completaron lo que hacia falta del módulo phy_rx.
- Se realizó la conexión del módulo phy.

**Reporte: (12/6/2021)**
El reporte se hizo entre los cuatro integrantes del grupo.

# Referencias

[1] S. Harding, *What Is PCIe? A Basic Definition*, feb. de 2021. dirección: `https://www.tomshardware.com/reviews/pcie-definition,5754.html`.

[2] J. Soto. (2021). «PHY PCIE.»