Módulo 6: Sincronización e interfases entre sistemas activos en distintos dominios de reloj

Contenidos del módulo 6

- Manejo de relojes no coherentes
- Alineación de datos en comunicaciones *multi-lane*.

- http://sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

Problemas básicos de interfase: manejo de relojes no coherentes

Un problema asociado a la metaestabilidad es la operación de sistemas con dos o más señales de reloj no coherentes entre sí

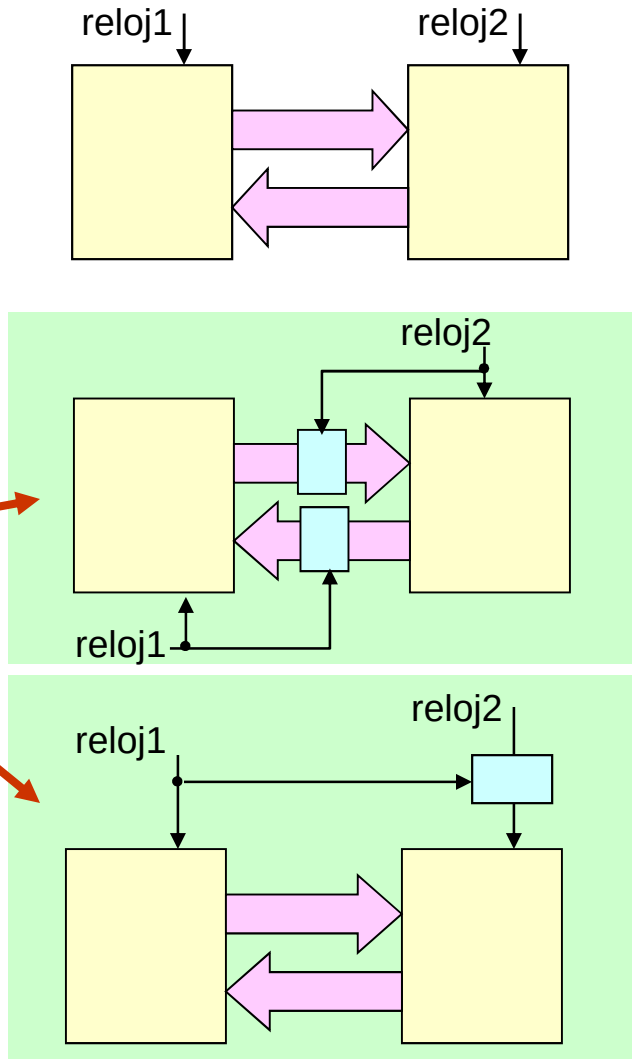
En el área de circuito donde un subsistema que opera con un reloj debe intercambiar datos con otro que opera con otro reloj la metaestabilidad es inevitable

En este caso las posibles soluciones son:

- sincronizadores en todas las líneas de intercambio
- adaptar uno de los relojes (el más lento) para hacerlo coherente con el otro (el más rápido)

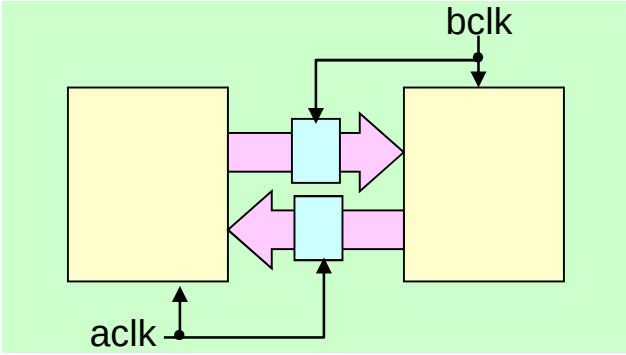
No siempre estas son las mejores soluciones, ya sea por la latencia que introducen, por el uso de recursos, u otras

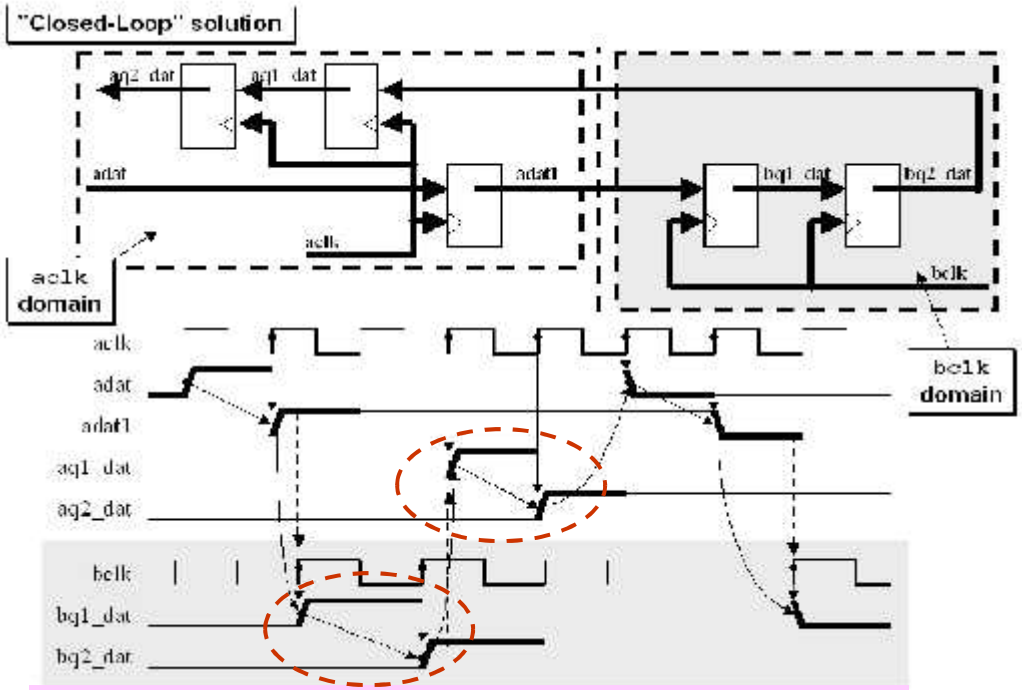
Discutir las restricciones de uso de cada uno de estos circuitos



Uso de sincronizadores en todas las líneas de intercambio

- El uso de sincronizadores en todas las líneas de intercambio es una solución posible cuando dos sistemas con relojes distintos interactúan entre sí
- En el ejemplo se supone que el lado A genera una señal *adat* y debe verificar que ese valor ha sido copiado del lado B, para lo cual recibe un “feedback” llamado *aq2_dat*
- Este tipo de sincronización requiere muchos flipflops, y una cuidadosa evaluación del valor de TcoM

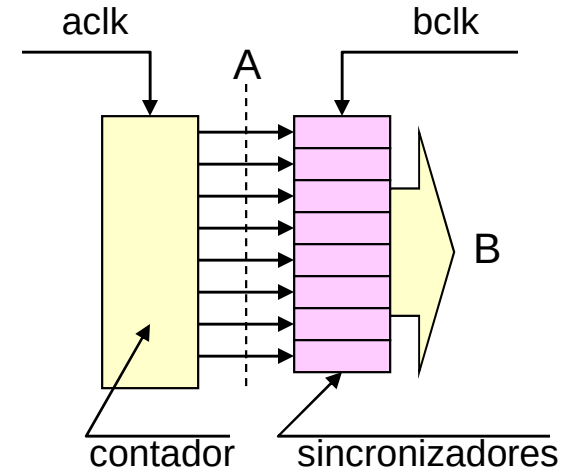


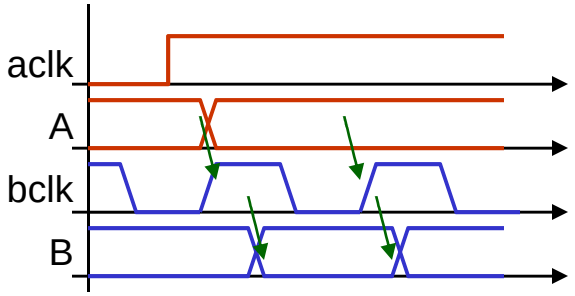


http://sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

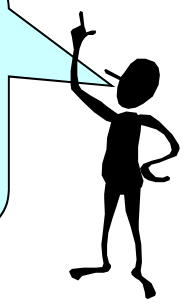
Riesgos del uso de sincronizadores con señales simultáneas

- Ejemplo: pasar el valor de un contador sincrónico que opera en el dominio del reloj *aclk* para su uso en una área con dominio de reloj *bclk*
  - En A la dispersión de las salidas del contador se debe sólo a los diferentes Tco de los flipflops
  - Si el flanco de *bclk* sucede en ese momento, algunas señales de A serán capturadas en su valor actual y otras en el previo
  - En B eso se traduce en que el *skew* entre salidas puede llegar a ser de un ciclo de *bclk*
- Soluciones:
  - Tratar que sólo cambie una señal a la vez:
    - Transcodificación One Hot (tipo demux)
    - Transcodificación Gray
  - Una señal Strobe demorada que agregue latencia pero evite la captura en el momento del cambio
  - Un circuito de handshake sincronizado



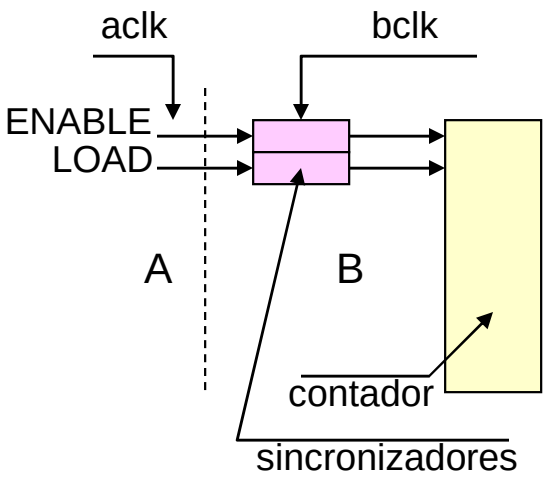


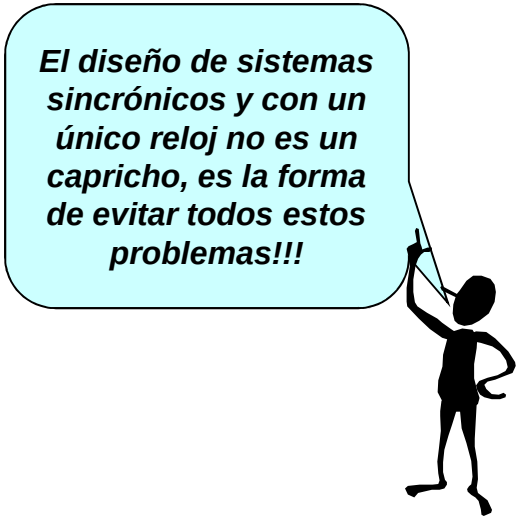
El diseño de sistemas sincrónicos y con un único reloj no es un capricho, es la forma de evitar todos estos problemas!!!



Riesgos del uso de sincronizadores con señales simultáneas

- La codificación OneHot puede ser útil para transferir ciertas acciones de control entre diferentes dominios.
- Ejemplo: desde el lado A se desea controlar las señales ENABLE y LOAD de un contador en el lado B
  - Si la señal ENABLE=1 y LOAD=1 se transfiere de A a B, puede que el contador no se cargue, que se cargue y cuente, etc... (porque se copie 00, 01, 10 o 11), aunque se usen dos sincronizadores
  - Una única señal de control que sea (ENABLE AND LOAD) consolida y unifica la acción de control
- Pero si hay muchas señales de control esta decodificación ONE HOT produce una expansión exponencial de combinaciones
- Los mismo sucede en señales de control secuenciales y excluyentes, que pueden terminar superpuestas o generar o regiones vacías luego de ser sincronizadas





Trascodificador Binario a GRAY

- Dados códigos de N bits, una secuencia con codificación binaria es aquella donde códigos sucesivos corresponden a valores numéricos sucesivos (000, 001, 010, 011, 100, 101, 110, 111, 000,...). En este caso, al pasar de un código a otro puede cambiar más de un bit (por ejemplo, al pasar desde 011 a 100, o desde 111 a 000 cambian todos).
- En una secuencia con codificación GRAY estos 2^N códigos están ordenados de otro modo, de modo que al pasar de un código al siguiente sólo cambie un bit (000, 001, 011, 010, 110, 111, 101, 100, 000, ...).
- Para convertir una secuencia binaria a GRAY los pasos son:

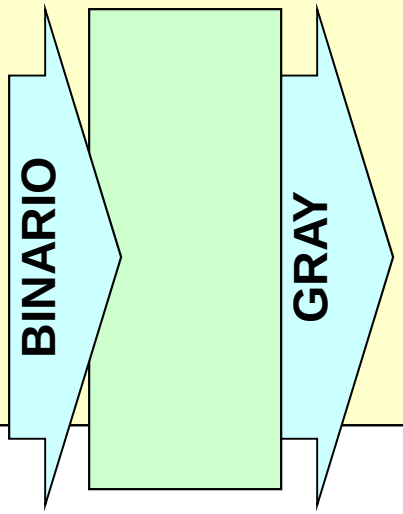
  - ***el bit MSB coincide en GRAY y binario***
  - ***cada bit restante GRAY se obtiene como el XOR del bit de igual peso y el inmediato superior del código binario***

- Los códigos GRAY son muy usados cuando en base a un código desea hacerse una decodificación ausente de “glitches” por la existencia de adyacencia (distancia de Hamming=1) entre distintos valores.

Trascodificador Binario a GRAY

```
ENTITY bin2gray IS
 GENERIC (ancho : INTEGER := 8);
 PORT(
  bin : IN BIT_VECTOR (ancho DOWNTO 1);
 gray : OUT BIT_VECTOR (ancho DOWNTO 1));
END ENTITY bin2gray;

ARCHITECTURE a OF bin2gray IS
BEGIN
 gray (ancho) <= bin (ancho);
 gray (ancho-1 DOWNTO 1) <=
   bin(ancho-1 DOWNTO 1)
   XOR bin(ancho DOWNTO 2);
END a;
```



Binario	GRAY
0000=0	0000=0
0001=1	0001=1
0010=2	0011=3
0011=3	0010=2
0100=4	0110=6
0101=5	0111=7
0110=6	0101=5
0111=7	0100=4
1000=8	1100=C
1001=9	1101=D
1010=A	1111=F
1011=B	1110=E
1100=C	1010=A
1101=D	1011=B
1110=E	1001=9
1111=F	1000=8

Procesos básicos de interfase: el proceso de HANDSHAKE

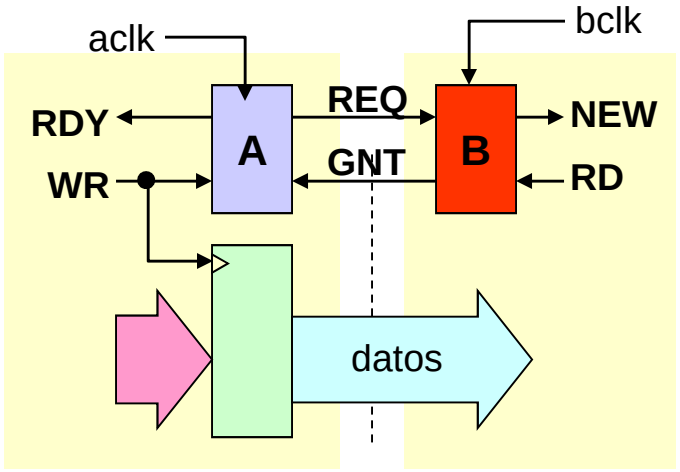
Si dos sistemas deben intercambiar datos en distintos dominios se necesita un mecanismo que asegure que:

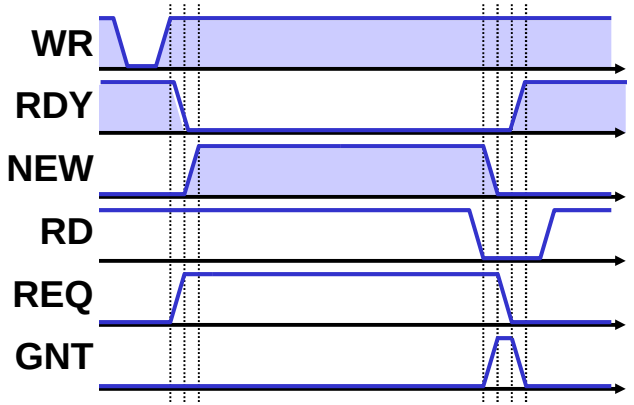
- el transmisor pueda enviar los datos (es decir que pueda chequear que el receptor pueda recibirlos)
- que el transmisor los envíe y el receptor los reciba
- que el receptor los lea

Esta interacción se llama ***handshake*** y necesita de dos líneas **REQ** y **GNT** *con sincronizadores* además de líneas locales (**RDY/WR** y **NEW/RD**) y para los datos .

La latencia de la sincronización de REQ->NEW asegura que los datos estén estables al ser leídos por B, y *no necesiten sincronizadores*

Las latencias **REQ->NEW** y **GNT->RDY** limitan la tasa máxima de transferencia





- Discutir las funcionalidades de las distintas señales, dibujar las máquinas de estado A y B
- Diseñar el circuito en VHDL y simularlo considerando WR activa en el flanco+ y RD activa en el flanco-

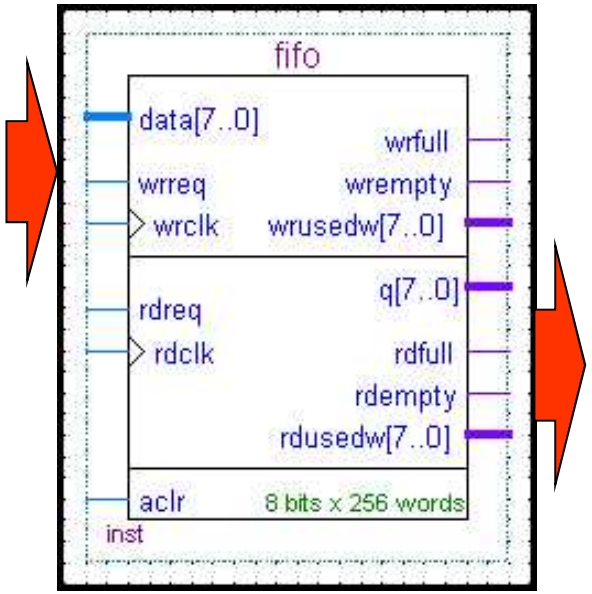
Procesos básicos de interfase: memorias compartidas

- Cuando los datos a intercambiar en la interfase entre dos sistemas son abundantes el esquema de handshake puede ser ineficiente por la alta latencia de cada transferencia
- Una alternativa es el empleo de memorias compartidas, que de hecho son un sistema más, intercalado entre ambos interlocutores
- Las memorias pueden ser de flujo de datos ***unidireccional*** o ***bidireccional***
  - El caso típico de una memoria de datos de flujo UNIDIRECCIONAL es un buffer tipo FIFO (First In First Out)
  - El caso típico de una memoria de datos BIDIRECCIONAL es una RAM Dual Port
- Para que estas memorias operen correctamente se requieren líneas de control
  - En el caso de un FIFO hacia el lado transmisor suele ser mandatoria la señal **FULL** y hacia el lado receptor la señal **EMPTY**
  - **EMPTY** también puede ser útil al transmisor para saber que el receptor leyó los datos
  - Y un FIFO típico también posee señales adicionales que indican el grado de ocupación de la memoria para optimizar su aprovechamiento

Analizar las LPMs disponibles para implementar FIFOs en VHDL

Procesos de interfase: memorias FIFO

- Las FIFO resultan ideales para “amortiguar” una interfase entre sistemas con distintas tasas de producción y consumo de datos, y dos posibles dominios de reloj, definidos por WRCLK y RDCLK
- En caso de un reloj común, una solución simple usa períodos de reloj alternados para conmutar entre la lectura/escritura de una RAM estándar (single port)
- Si se usa una RAM dual-port la sincronización depende si el reloj es común (WRCLK == RDCLK) o si WRCLK y RDCLK son diferentes
  - Si WRCLK==RDCLK una máquina sincrónica debe arbitrar las posibles situaciones (NADA, READ, WRITE, WRITE+READ)
  - Si WRCLK!=RDCLK el análisis de la situación (WRITE+READ) presenta mayor complejidad, en los casos límite (FIFO lleno o vacío)

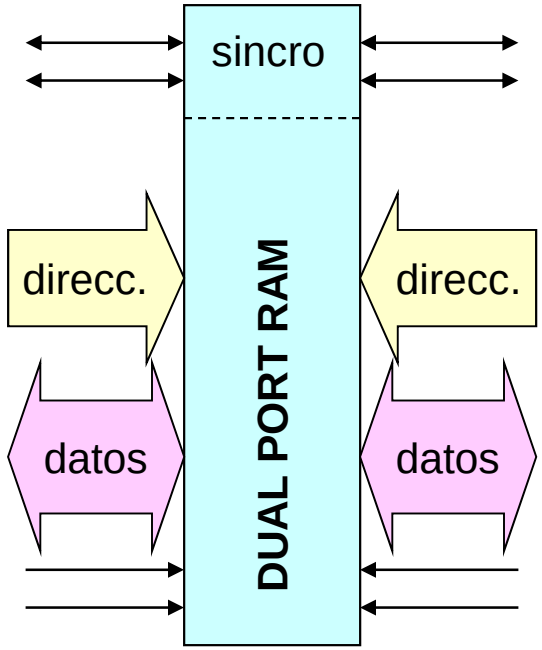


Es claro que las tasas instantáneas son las que pueden ser distintas!!! Las tasas medias de producción y consumo deben ser iguales!!!

Procesos de interfase: memorias compartidas bidireccionales

En las memorias bidireccionales la sincronización del flujo de datos es más compleja, pudiendo emplear:

- **FLAGS**: son bits de control que pueden ser escritos por un actor y leídos por otro. Por ejemplo, un proceso de Handshake requiere dos flags, uno por cada actor
- **SEMÁFOROS**: son bits de control que pueden ser escritos y leídos por ambos actores y que pueden ser usados para reservar un dado recurso. El uso de semáforos requiere de circuitos tipo TEST&SET
- **ARBITROS**: si ambos actores pueden escribir sobre las mismas áreas de memoria es importante crear circuitos que definan los “derechos de acceso” en cada momento. Estos derechos pueden ser asignados con prioridades fijas o variables (por ejemplo, tipo Round Robin)



Analizar las LPMs disponibles para implementar Dual Port RAMs en VHDL

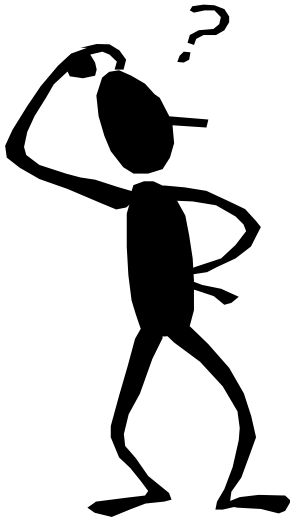
Manejo de memorias compartidas: los semáforos

- Los semáforos se usan cuando se requiere el uso de un recurso durante un tiempo prolongado. Si el recurso es único, son semáforos binarios, y si es múltiple es un semáforo contador, que indica cuántos recursos están libres
- Posibles aplicaciones:
  - Controlar el uso de un área de memoria compartida para garantizar la coherencia de datos
  - Administrar el permiso de acceso a un único recurso común de hardware (binario). Por ejemplo, una tarjeta de memoria, una impresora
  - Administrar un pool limitado de recursos (binario). Por ejemplo, un conjunto de buffers de memoria para almacenamiento temporario de datos
  - Sincronizar tareas.
- Siempre es posible que un recurso pedido, por alguna razón no quede disponible (Ej: por error quien lo tiene no lo libera) y que ello produzca bloqueos, por eso siempre conviene definir Timeouts y acciones de recuperación de errores.

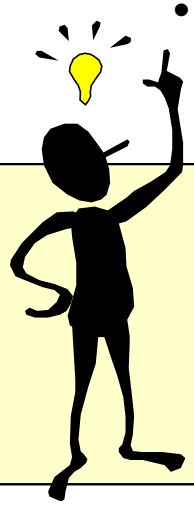
Manejo de memorias compartidas: los semáforos

- Corresponden a una construcción de programación descripta por E. W. Dijkstra en la década del 60.
- Cuando el recurso a compartir es sólo uno el estado del semáforo se describe mediante un único bit, si el recurso es múltiple está asociado a un contador UP/DOWN.
- Un semáforo soporta sobre sí tres acciones:
  - la ***inicialización***, cuando se carga con su valor máximo
  - la ***reserva de recursos***, un proceso TEST AND SET *atómico* (no interrumpible) donde:
    - si no hay recursos se avisa que no los hay
    - si hay recursos se reserva uno y se avisa que ha sido asignado
  - la ***liberación de recursos***, un proceso también atómico que repone el recurso liberado para que pueda ser nuevamente requerido
- Al sintetizar un semáforo por hardware es crítico *arbitrar* las prioridades con las que se atienden pedidos *simultáneos* de reserva de recursos para asegurar su atención *atómica*.
- Un modelo simple de un semáforo binario es el de una calle de una sola vía, donde un vehículo que desea ingresar debe reservar la calle para evitar que otro entre por el otro extremo y se genere un embotellamiento ("***deadlock***") a medio camino.
- La necesidad de reserva de recursos *atómica*, es para evitar el caso donde ambos vehículos testean simultáneamente que el camino está disponible y lo reservan pero se produce el deadlock.

Linked State Machines y Petri Nets



- En cualquier diseño de mediana complejidad es previsible la necesidad de modelar múltiples máquinas de estado que operan concurrentemente, y que interactúan entre si.
- Aunque M máquinas de estado con N estados cada una pueden presentar N^M posibles casos, en general una máquina no necesita operar conociendo los detalles internos completos de cada una de las otras máquinas, posibilitando el modelado de ***LSMs (Linked State Machines)***
- Desde el punto de vista de su interacción, los estados internos de una FSM pueden ser agrupados en clases de equivalencia, simplificado el modelado.



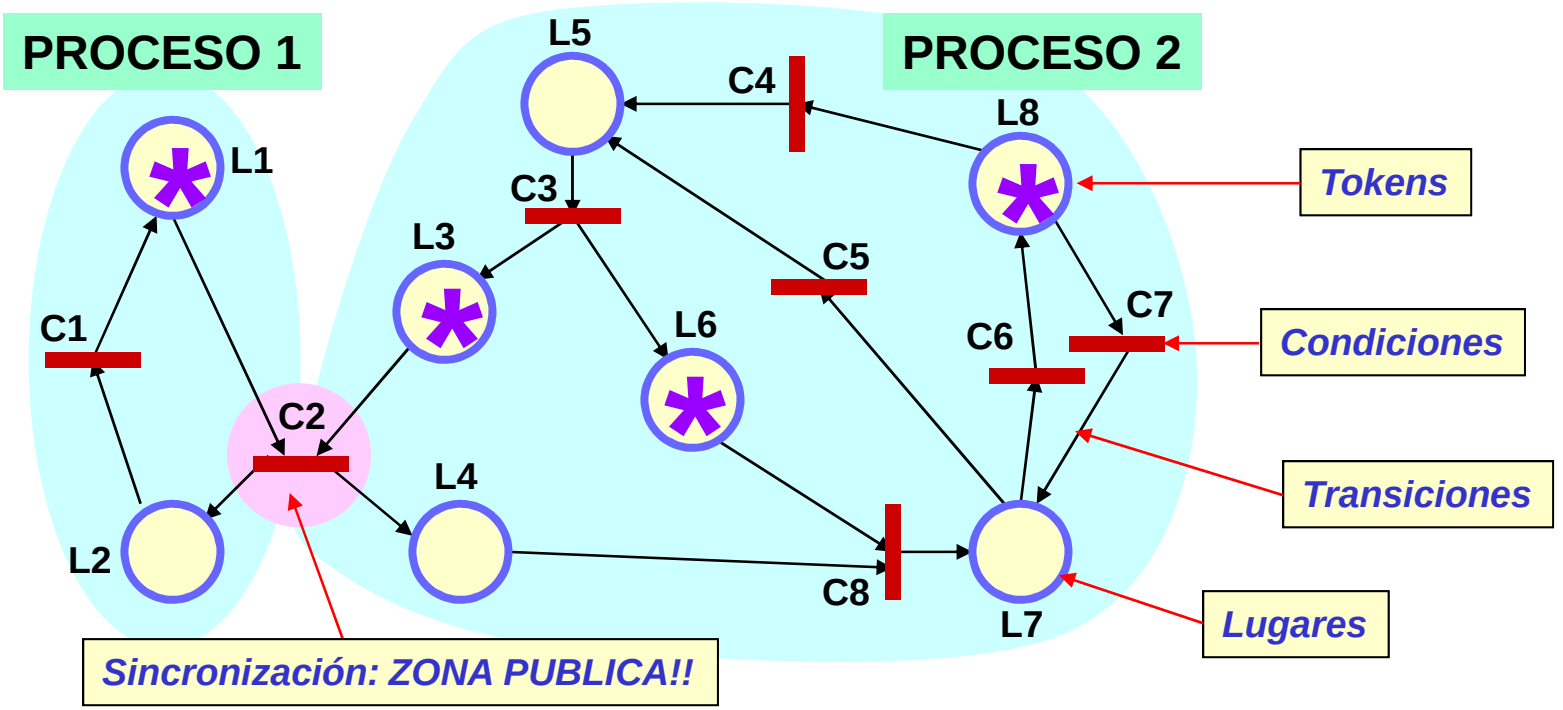
- Si en una entidad se describe una máquina de estados, aquellos detalles “públicos” de esa máquina pueden ser “publicados” en forma de ***ports*** de un nuevo tipo, definido en un ***package*** apropiado.

Linked State Machines y Petri Nets

- Las ***Redes de Petri*** (***Petri Nets*** o ***PN***) son un formalismo apto para la descripción de ***Linked State Machines***, pues permiten describir procesos que interactúan y se sincronizan, al modo "***data flow***".

  - Son grafos dirigidos, en los que los vértices son llamados "***lugares***" (y están asociados al estado de la red) y las aristas son llamadas "***transiciones***" (y están asociadas a los posible eventos en la red).
  - En una ***PN*** "***marcada***" existen uno o más símbolos llamados "***tokens***" (cuya cantidad puede ser variable) que se estacionan en ***lugares***, y se mueven de lugar a lugar siguiendo las ***transiciones que se "disparan".***
  - Una transición suele tiene asociada una ***condición de disparo***, ciertas flechas que vienen desde lugares de origen y ciertas flechas que van hacia lugares de destino
  - El disparo de esta transición requiere que en todos los lugares de origen haya al menos un ***token***, y que la condición se cumpla.

Linked State Machines y Petri Nets

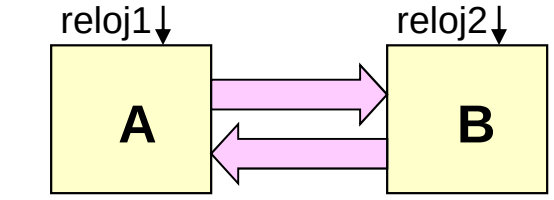


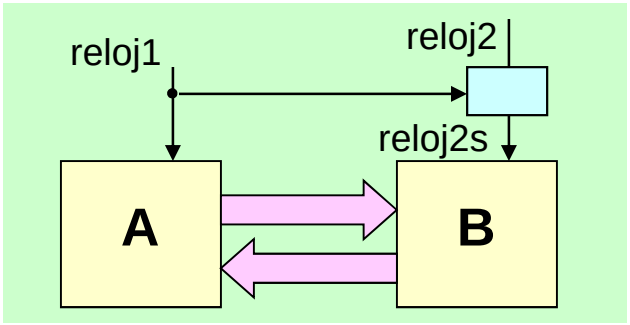
- Ejemplo que muestra dos FSM sincronizadas entre sí.
- A los fines “públicos” útiles para la sincronización, el PROCESO 2 puede modelizarse como si tuviera sólo dos estados (similar al PROCESO 1)
- Se observan transiciones donde el número de ***tokens*** aumenta (**C3**), y otras donde ese número disminuye (**C8**).

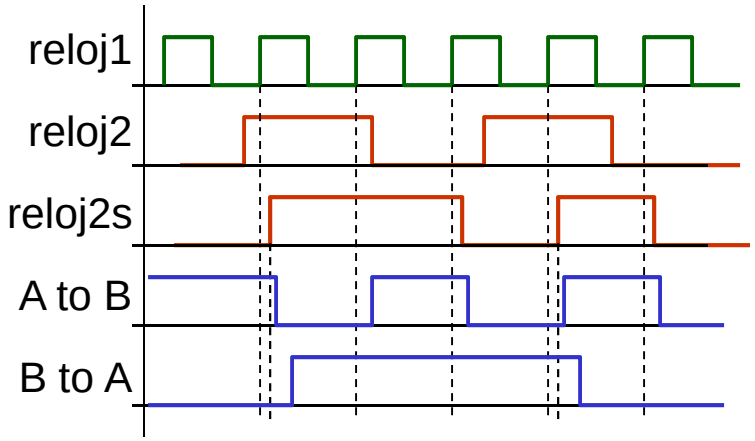
Fuente: BAKER

Problemas básicos de interfase: sincronización del reloj

- La sincronización de la señal de reloj de un dominio con el reloj de otro dominio puede servir si la frecuencia del *reloj2* es inferior a la del *reloj1* (F2 < F1/2)
  - En este caso tanto el '1' como el '0' de *reloj2s* están sincronizados con *reloj1*
- Pero debe tenerse en cuenta que el flanco activo de *reloj2s* tendrá un retardo Tco respecto al flanco activo de *reloj1*, por lo que:
  - En las salidas de A hacia B debe asegurarse que se satisface *Thold*
  - Las salidas de B hacia A tendrán un retardo adicional a considerar para cumplir *Tsetup* del lado A





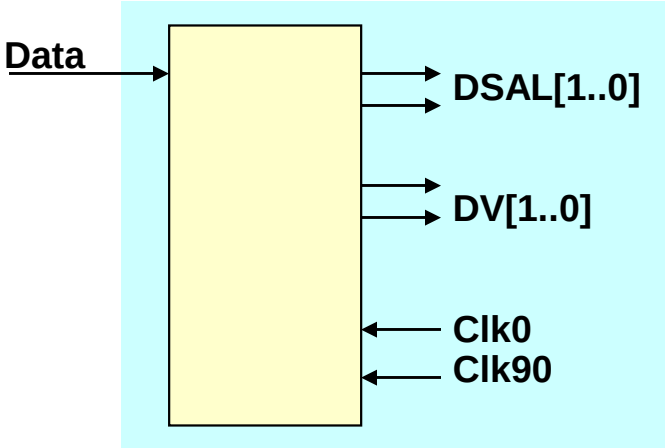


Recuperación asincrónica de datos

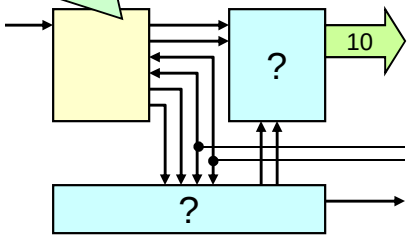
Si se dispone de un reloj local y no se desea usar un VCO, como la velocidad a la que ingresan los datos no es coherente con ese reloj (puede ser más rápida o más lenta) en cada ciclo de reloj local pueden suceder 3 eventos:

- Que haya ingresado un único bit válido de datos: este es el caso más habitual cuando los relojes de transmisión y recepción son muy similares
- Que no haya ingresado un bit válido de datos: esto se da esporádicamente si el reloj de recepción es más rápido que el de transmisión
- Que hayan ingresado dos bits de datos: esto se da esporádicamente si el reloj de recepción es más lento que el de transmisión

La nota de XILINX **XAPP224** muestra una solución si se dispone de un reloj local con fase 0 y 90 grados



Si los datos que salen son pasados de serie a paralelo (P.Ej: a 10 bits) analizar el diseño del deserializador!

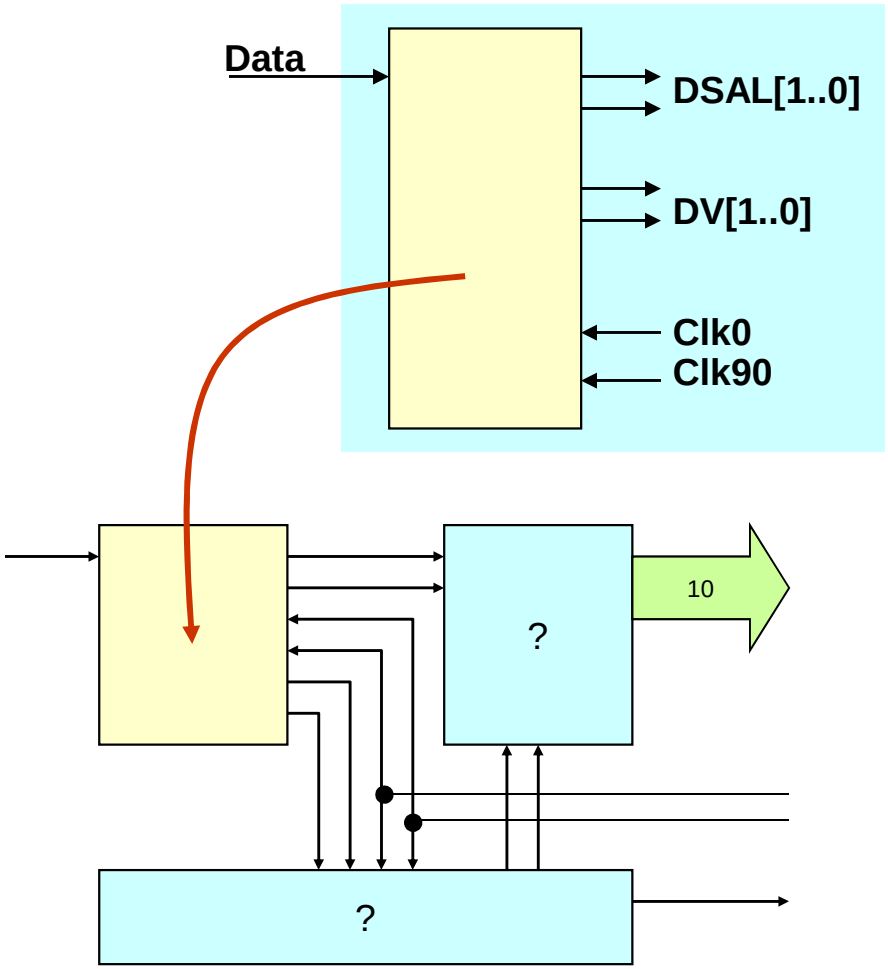


- ***Dados datos que ingresan en formato manchester diferencial agruparlos en paquetes de 10 bits***
- ***Realizar el diseño en VHDL y simularlo***

Recuperación asincrónica de datos

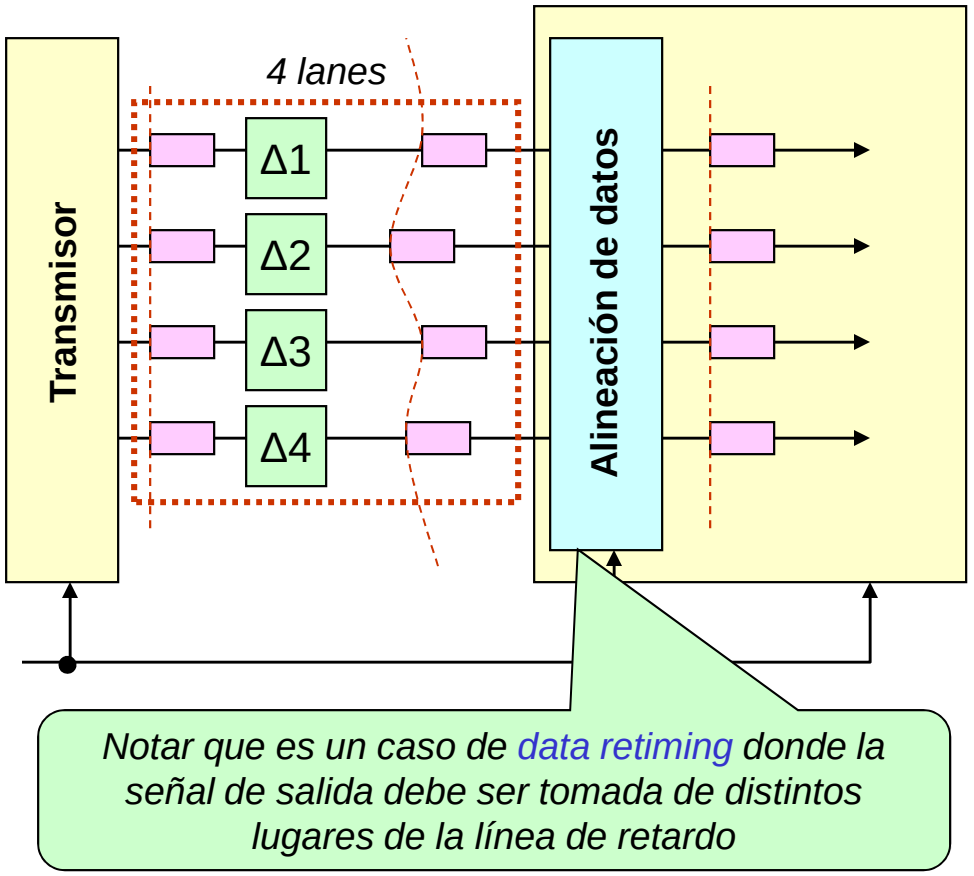
Discusión y ejercicio:

Cómo implementaría el circuito completo del recuperador más el deserializador?



Alineación de datos sincrónicos multi-lane

- En enlaces digitales sincrónicos de alta velocidad pueden transmitirse datos en paralelo por varios canales
- Estos datos pueden arribar al receptor desfasados entre sí, debido a varias causa (distintas capacidades parásitas o longitud de pistas de circuito impreso)
- En ese caso debe muestrearse cada dato en el momento más oportuno, y aplicarle una demora distinta de modo de lograr que los datos salgan de módulo de alineación con identica temporización
- La nota de XILINX **XAPP225** describe una posible solución



Ref: ***http://direct.xilinx.com/bvdocs/appnotes/xapp225.pdf***