



Módulo 13:

Buses de comunicación

Contenidos del módulo 13

- Qué es una transacción
 - Estados de una transacción
 - Modos: single data o burst
 - Jerarquías de memorias y problemas asociados de Coherencia de datos (las memorias caché)
- Buses inter-chip:
 - ISA, MCA, SCSI
 - PCI
 - PCI Express.
- Buses intra-chip:
 - AMBA
 - AVALON.

Qué es una memoria?

- Una memoria es un recurso donde es posible almacenar información. Por ejemplo, en un microcontrolador una memoria sirve para almacenar Instrucciones y Datos
 - Una memoria puede ser de lectura solamente (ROM)
 - O servir para escribir y leer valores (RAM)
- A su vez, según su comportamiento, puede ser
 - No volátil, si la información que contiene no se borra al cortarse la alimentación: FLASH, E2ROM, NVRAM, FRAM
 - Volátil, si la información se pierde al cortarse la alimentación: (estática) SRAM, (dinámica) SDRAM, DDR2/3, QDR, ZBT
- Existen muchas tecnologías de memoria, y permanentemente se innova
- A diferencia de un procesador la memoria no elabora los datos, y devuelve lo mismo que se le escribe

Qué es un procesador?

- Un procesador es un elemento circuital capaz de elaborar datos siguiendo una secuencia de instrucciones.
- Esa secuencia de instrucciones puede incluir acciones de decisión en base al valor de los datos o a eventos externos.
- A diferencia de una memoria, un procesador posee la capacidad de elaborar los datos, y cambiar su valor.
- Un procesador actúa sobre su entorno a través de sus interfases de entrada/salida, que pueden ser desde simples bits hasta complejos canales de comunicaciones.
- Incluso el procesador más avanzado sólo puede resolver instrucciones sumamente elementales, por lo que su performance está relacionada a cuántas acciones puede resolver en un dado tiempo, lo que está directamente relacionado a la potencia de las instrucciones, la cantidad de instrucciones por segundo, y a la precisión de los datos que procesa

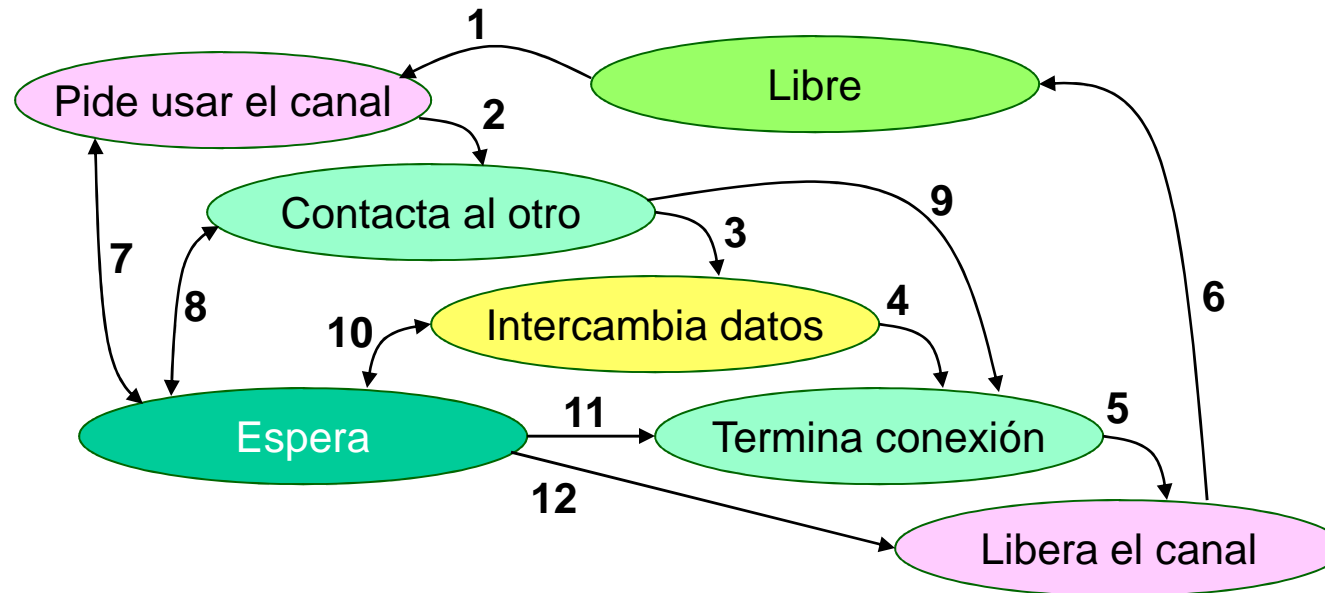
Qué es una transacción?

- Cada intercambio de información entre un procesador y la memoria o la interfase de entrada/salida, ya sea de lectura o escritura, conforma una transacción. En general una transacción requiere varias etapas (o estados):
 - Solicitar el uso del canal de comunicación.
 - Establecer una conexión virtual con la contraparte.
 - Identificar a la información que se desea transferir.
 - Transferir esa información entre proveedor y consumidor a una tasa acorde.
 - Cerrar el proceso de transferencia liberando la conexión virtual.
 - Liberar el uso del canal de comunicación.
- Cada una de estas etapas puede ser mínima o nula, o sumamente compleja, según cuál sea el mecanismo de transacción disponible. Esto depende de criterios de costo y performance

Estados de una transacción

- Una transacción conforma una única acción, pero al recorrer varios estados su ejecución puede sufrir alteraciones:
- Puede quedar en suspenso:
 - Solicitar el uso del canal de comunicación y que esté ocupado.
 - Si hay una conexión vigente y la otra parte no puede responder en tiempo.
 - Por tiempos de espera excesivos al transferir información.
- Puede ser abortada:
 - Al solicitar usar el canal si pasa cierto tiempo sin que se permita ese acceso.
 - Al querer establecer una conexión virtual si la otra parte no responde.
 - Al identificar la información que se desea transferir y que no sea compatible.
 - Si al transferir información entre proveedor y consumidor no puede completarse esa transferencia o da error.
 - Por un evento que fuerce a liberar la conexión virtual y/o el uso del canal antes de completar la transferencia.

Estados de una transacción



Una buena idea puede ser pensar en una llamada telefónica

Ejercicio:

- Identificar cada uno de los estados y las causa de activación de las posibles transiciones, con posibles ejemplos
- Perfeccionar el diagrama, si corresponde, con otras transiciones



Jerarquías de procesadores

- En un sistema con más de un procesador, de ser necesario el uso de recursos comunes se hace necesario definir criterios por los cuales asignar prioridades.
- Por eso al definir qué es una transacción se mencionó “Solicitar el uso del canal de comunicación.”
- En general el arbitraje determina que el procesador no favorecido deba esperar para el uso del recurso, lo que genera desaprovechamiento de tiempos
- El grado de acoplamiento entre procesos, o la posibilidad de definir procesos en que el procesador en espera pueda aprovechar esos tiempos muertos, permite infinidad de estrategias.
- En el caso de procesos fuertemente acoplados, justifica buscar mecanismos de replicación (o imágenes) de ese recurso compartido, lo que genera la necesidad de sincronizar esa información.

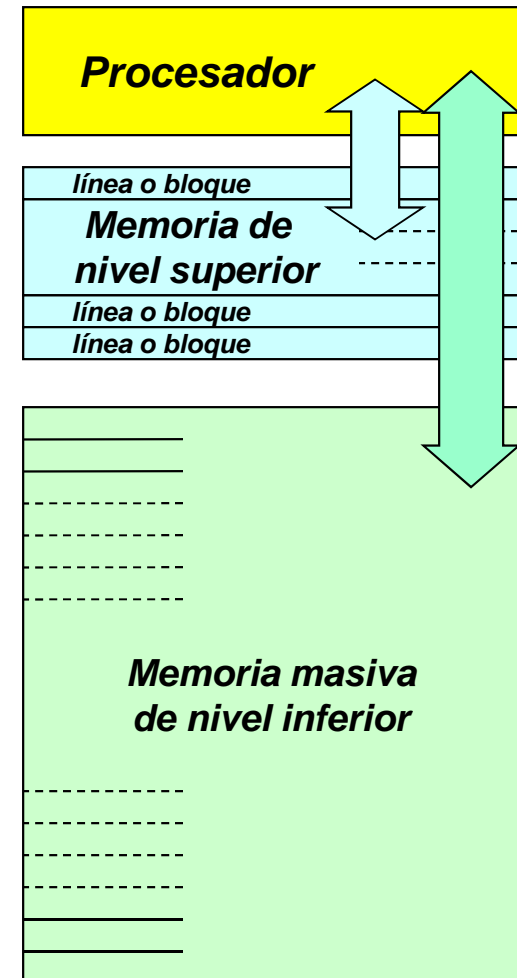
Jerarquías de memorias

- En un sistema donde se busca alta performance, dos requerimientos pueden colisionar entre sí:
 - La cantidad de información a procesar (magnitud de datos)
 - La necesidad de procesar esa información a la mayor velocidad posible, referido al mencionar la definición de transacción en “Transferir esa información entre proveedor y consumidor a una tasa acorde”.
- Una posible solución a este problema es el empleo de jerarquías de memoria:
 - Memorias masivas, con capacidad de almacenar gran volumen de datos, de velocidad intermedia
 - Memorias rápidas, con no tanta capacidad de almacenamiento
 - Y un mecanismo de intercambio de contenidos entre ambas memorias

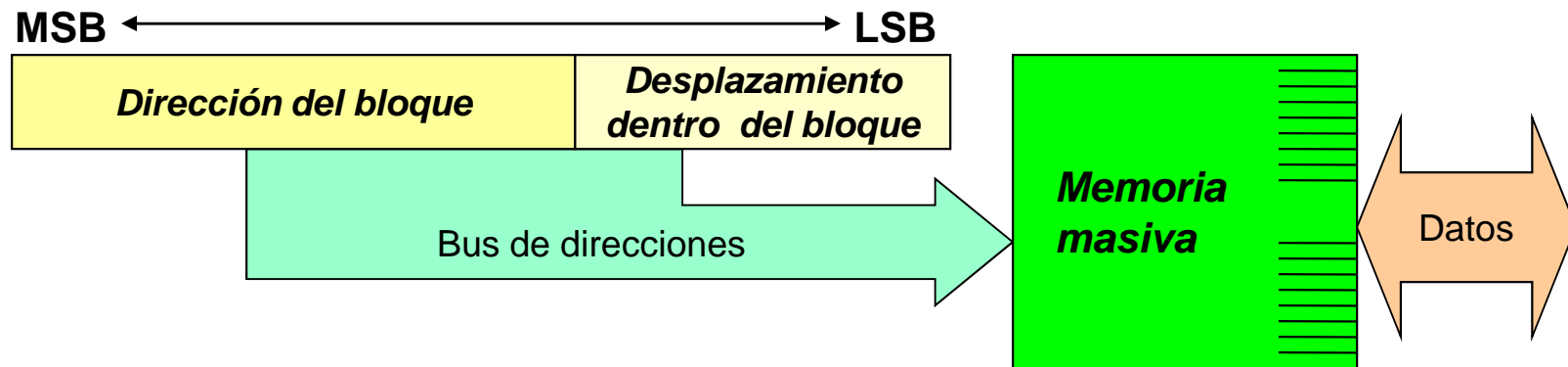
Jerarquías de memorias: las memorias “cache”

En un sistema de procesamiento suele darse en el uso de la memoria los fenómenos de:

- ❑ **localidad temporal:** si se referencia a un elemento en un instante, es muy probable que se volverá a referenciarlo en corto tiempo
- ❑ **localidad espacial:** si se referencia a un dado elemento, es muy probable que los ubicados próximos a él serán referenciados pronto
- ❑ Esto lleva a la conveniencia de organizar una **jerarquía de memoria:**
 - ❑ Una memoria pequeña y rápida (“cache”) organizada por bloques o líneas (nivel superior)
 - ❑ Una memoria masiva de acceso más lento (nivel inferior de la jerarquía)

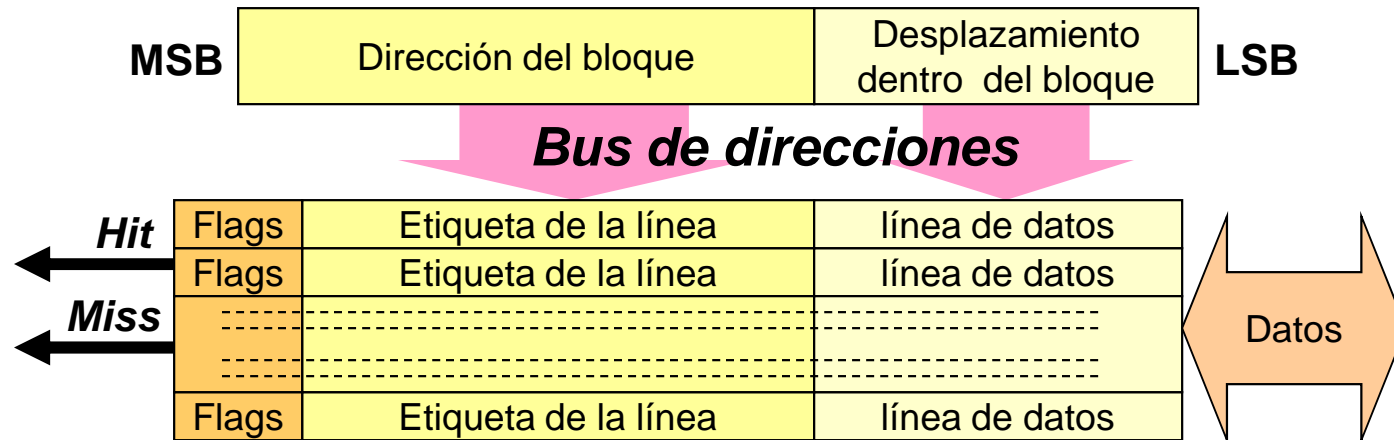


Jerarquías de memorias: las memorias “cache”



- ❑ La memoria de primer nivel y la memoria masiva están segmentadas en bloques (o “lineas”), donde debe diferenciarse entre la dirección de un bloque (base), y la de los distintos elementos dentro del bloque (desplazamiento).
- ❑ La memoria masiva es convencional, donde la posición de un dado elemento está asociada al valor del bus de direcciones (dirección del bloque más desplazamiento)

Jerarquías de memorias: las memorias “cache”



- ❑ En una cache, cada línea de datos está asociada a una línea de memoria masiva no sólo por su contenido sino también por una *etiqueta* que identifica esa relación, y ciertos *flags* que definen su estado de coincidencia y/o validez
- ❑ El direccionamiento de líneas en la cache puede ser de tipo asociativo (aunque existen alternativas), donde dada la dirección absoluta de un bloque se detecta si el mismo está almacenado en la cache (HIT) o no (MISS).
- ❑ Si en una lectura se da un HIT, los datos pueden ser leídos desde la cache, si se da un MISS los datos de la línea deben ser traídos de la memoria principal.

Jerarquías de memorias:

Lectura de las memorias “cache”

- ❑ El objetivo de una jerarquía de memoria es mejorar la performance, en una memoria cache ésta depende de cuán seguido se dan HITs, cuándo se dan MISS, y de cómo mover líneas entre la cache y la memoria principal.
- ❑ En la **lectura** de una línea, si ella está en la cache (HIT) se gana tiempo por ser una memoria rápida. Sinó (en un MISS), mientras la línea se lee de la memoria masiva se actualiza la línea de cache; el acceso será lento.
- ❑ Ante un MISS puede ser necesario descartar alguna línea de la cache (Ej: una línea inválida o la usada hace más tiempo -LRU-), y esta elección de la línea a reemplazar puede ser sobre el total, o sólo entre un dado subconjunto.
- ❑ Cuanto mayor es una línea mas probable es un HIT, pero también es posible almacenar menos líneas en un dado tamaño de cache, y ante un MISS mayor es el tiempo de reemplazo de una línea desde la memoria principal a la cache

Jerarquías de memorias: Escritura en memorias “cache”

❑ Para **escribir** la memoria existen dos opciones: **Write through** y **WriteBack** (o **CopyBack**)

❑ **Write through:** en un HIT, cada dato se escribe en paralelo en la cache y en la memoria masiva; en un MISS sólo se escribe la memoria masiva sin actualizar la cache. El ciclo de escritura es lento, pero la memoria masiva posee siempre datos coherentes (consistentes) con los de la cache.

❑ **WriteBack o CopyBack:**

❑ en un HIT sólo se actualiza la línea de cache, a la que se marca como *dirty*

❑ en un MISS hay dos acciones usuales:

❑ se escribe la memoria masiva sin actualizar la cache

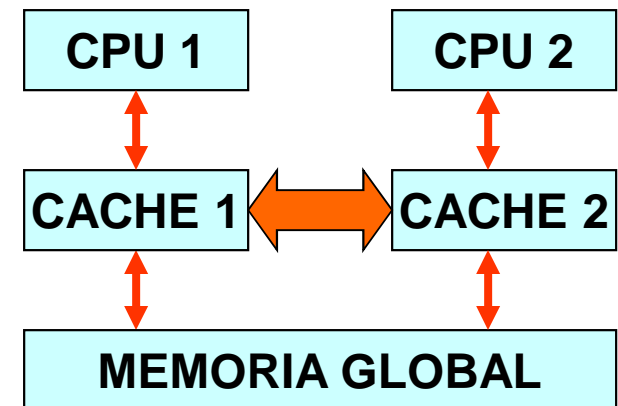
❑ o se lee la línea de memoria a cache y luego sólo se actualiza la cache (considerando la posible localidad espacial).

❑ Cada línea es transferida a la memoria masiva recién cuando debe ser reemplazada por una nueva línea.

❑ Cada ciclo de escritura es rápido, pero si una línea “dirty” debe ser reemplazada se necesita transferir la línea completa desde la cache a la memoria masiva.

Jerarquías de memorias: Coherencia en memorias “cache”

- Si varios master usan una misma memoria existe el problema de la coherencia, es decir que distintos master usen distintos valores para el mismo dato
 - Si la CPU1, usando *WriteThrough*, modifica su Cache1 y la memoria global, y la CPU2 tiene esa línea de memoria en su CACHE2, esta línea debe ser recargada desde memoria global
 - Si la CPU1, usa *WriteBack* y modifica su CACHE1 y la CPU2 quiere usar esa línea de memoria, antes de cargar CACHE2 la línea de CACHE1 debe ser cargada a memoria global
- La solución de conflictos de coherencia debe ser realizada a gran velocidad



Al usar switches los problemas de coherencia son aun mas complejos, por haber varias transacciones simultáneas



Buses paralelos (o system buses)

- ❑ A diferencia de las interfases seriales, que comunican sistemas entre sí, la mayoría de los buses paralelos tienen por fin comunicar dispositivos dentro de un sistema.
- ❑ El continuo incremento en la performance de los procesadores y de las aplicaciones que requieren movimiento masivo de información (P.Ej: gráficos) ha llevado a la necesidad de definir buses con cada vez mayor Throughput.

- ❑ **PC/XT e ISA** (Industry Standard Architecture): IBM
- ❑ **EISA** (Extended Industry Standard Architecture): Compaq. Incorpora la idea de transacción "burst"
- ❑ **NuBus**: Apple
- ❑ **MCA** (Micro Channel Architecture): IBM. Con un modo burst llamado Streaming Data Procedure
- ❑ **VL Bus** (Video Electronics Standards Association Local bus)
- ❑ **PCI** (Peripheral Component Interconnect): Intel

BUS	FECHA	DATOS (bits)	RELOJ (MHz)	THROUGHPUT (Mbytes/seg)
PC/XT	1981	8	4.77	4.77
PC/AT (ISA)	1985	16	8.33	16.66
EISA	1988	32	8.33	33
NuBus	1993	32	20	80
MCA	1987	32	10	40
	1994	64	20	160
VL Bus	1994	32	25-66	264
	1994	64	25-66	528
PCI	1992	32	0-33	132
	1995	32-64	0-33-66	528

Algo de historia sólo para comprender el porque de los buses más modernos

Porqué tantos buses? Qué caracteriza a un bus?

Gráficos cada vez más complejos y Aplicaciones multimedia:

- Interfases GUI (windows/xwindows)
- Animaciones (image synthesis)
- Captura y reproducción de Video y Audio con calidad profesional
- Teleconferencias

Redes cada vez más veloces:

- ATM, FDDI, SONET/SDH, Ethernet 100Mbits/segundo, Ethernet 1Gbps

Dispositivos de almacenamiento de datos cada vez con mayor capacidad

- Discos magnéticos, y discos ópticos (CD, DVDs, BLUE RAY)

Aplicaciones orientadas a transferencias de bloques de datos cada vez mayores y más veloces!

Ancho de banda

- Cantidad de información que se puede transportar por unidad de tiempo.

Capacidad de operación multi-Master en tiempo real

- Poseer tiempos de atención predecibles, y prever fenómenos de deadlock

Características eléctricas

- Longitud eléctrica, carga tolerable, voltaje de operación y consumos

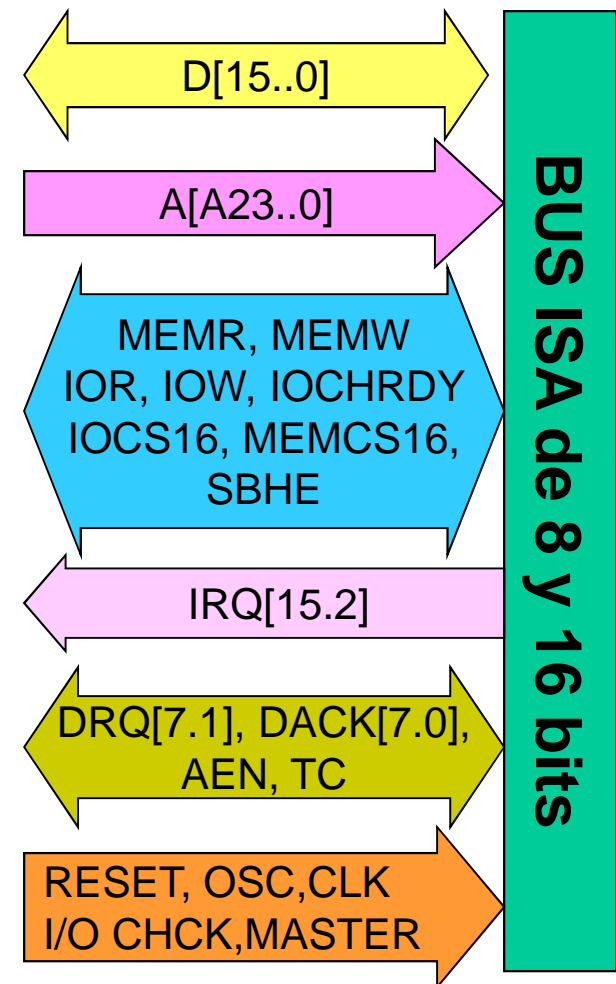
Características mecánicas

- Tipo de conectores, tamaño de plaquetas.

Costo de realización

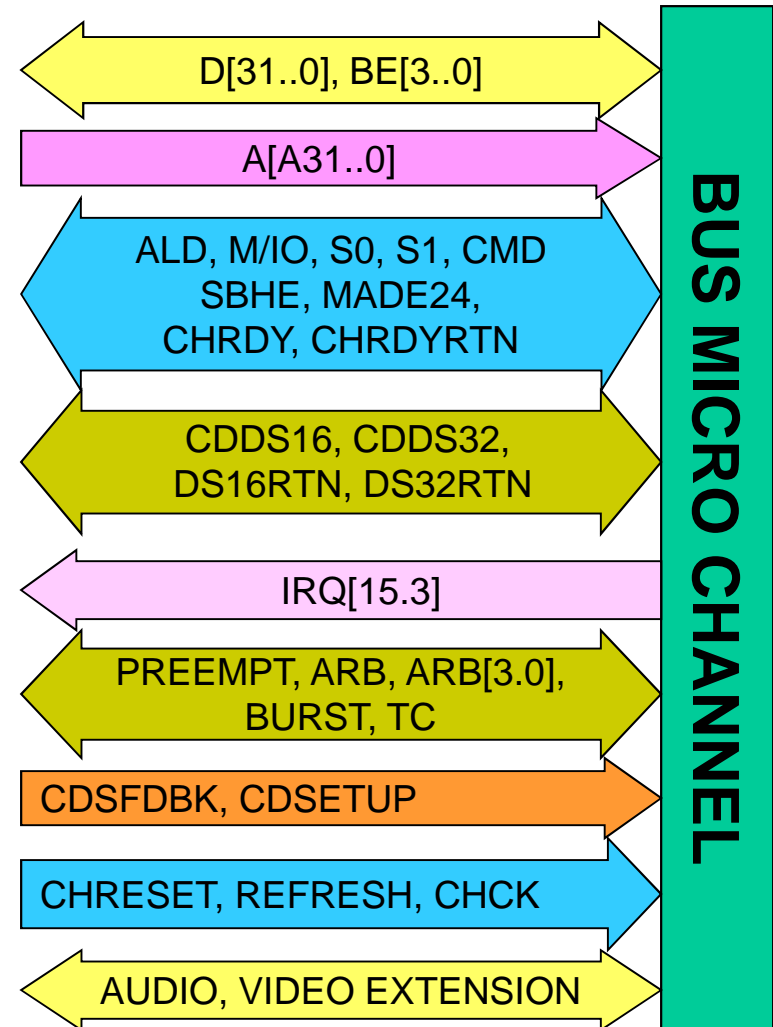
ISA 8 y 16 bits

- ❑ Bus síncrono (operación definida por CLK), con una línea para aceptar procesos lentos.
- ❑ Velocidad de transferencia máxima de 8.3 Mbytes/seg (16 bits @ 8.33MHz @ 2 ciclos por transferencia)
- ❑ Cada transferencia usa dos buses independientes, uno para direcciones y otro para datos,.
- ❑ Los movimientos de bloques de datos se realizan con una sucesión de movimientos individuales de datos
- ❑ Es posible el uso de DMA para acelerar los procesos de transferencia
- ❑ En la expansión a 16 bits un periférico distinto al HOST puede tomar el control del bus durante el DMA
- ❑ Tiene una versión llamada **PC-104** pensada **para sistemas embedded**, con distintas especificaciones eléctricas y mecánicas



Micro-channel (MCA)

- ❑ Operación cuasi-asincrónica
- ❑ Con un bus de datos de hasta 32 bits, accesible de a byte, según definen cuatro líneas BE[3..0]
- ❑ Bus de direcciones separado de 32 líneas
- ❑ Direcccionamiento asociado a la ubicación física del periférico en el bus, con autoidentificación del periférico (lo que luego será Plug&Play)
- ❑ Con mayor facilidad para sistemas con más de un master (HOST, DMA y de periféricos)
- ❑ Aparición del modo *burst* indicado mediante una línea especial
- ❑ Con extensiones especiales para video y audio (mediante líneas dedicadas)

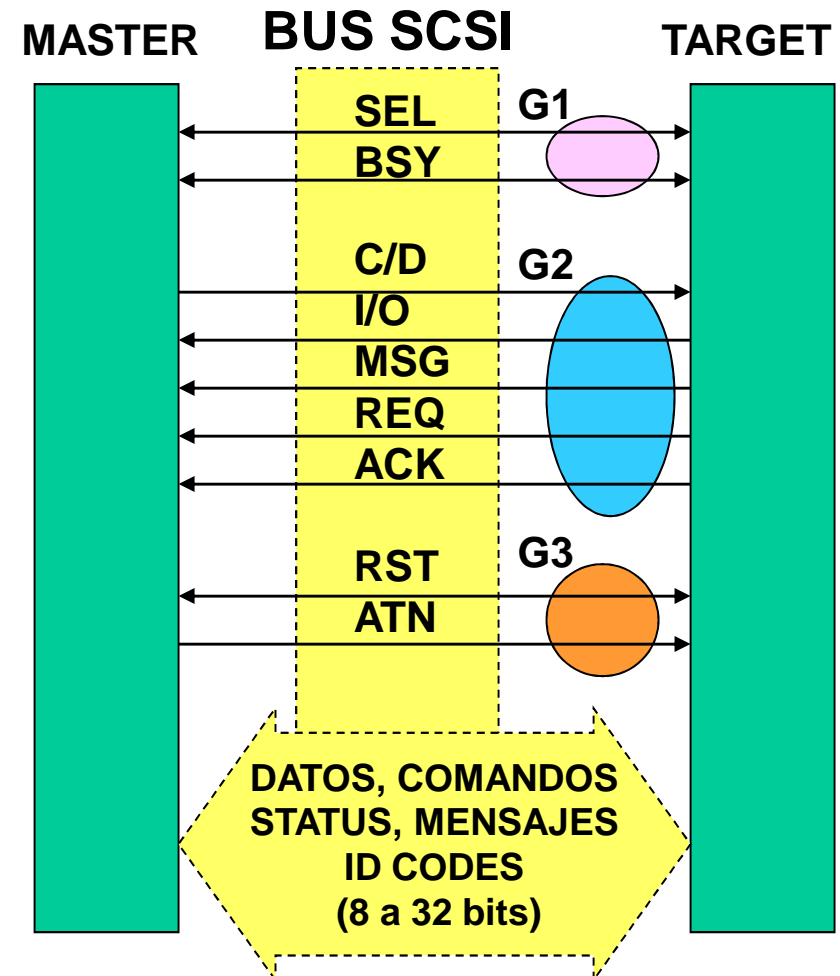


Bus SCSI: Small Computer System Interface

- ❑ Desarrollado en conjunto entre NCR y SHUGART entre 1981 y 1982
- ❑ Estandarizada mediante la norma ANSI X3.131-1986
- ❑ Con datos de 8 bits (16 bits en SCSI-2 y 32 bits en la opción **Wide_SCSI**).
- ❑ Bus síncronico de 1 MHz (en **Fast_SCSI** aumenta a 10MHz).
- ❑ La combinación (**Fast_SCSI+Wide_SCSI**) permite transferencias por bloques de hasta 40 Megabytes/segundo.
- ❑ Con especificación eléctrica tipo single-ended (para buses de hasta 6mts) o diferenciales (para buses de hasta 25 metros de longitud).
- ❑ Emplea transferencias controladas por “*handshake*” y sólo por bloques, en vez de direccionamiento físico.
- ❑ Soporta hasta 8 dispositivos (incluyendo el HOST)
- ❑ Posibilidad de múltiples MASTERS (o *initiators*) y múltiples destinatarios (o *targets*)

Bus SCSI

- ❑ Toda la actividad en el bus SCSI ocurre como una de las posibles siguientes fases:
 - ❑ *Bus_Free, Arbitration, Selection, Reselection, Command, Data, Status, Message.*
- ❑ Con señales de control:
 - ❑ **Grupo1:** BSY, SEL. Permite a un iniciador seleccionar un target
 - ❑ **Grupo2:** C/D, I/O, MSG, REQ, ACK. Selección de fases y handshake
 - ❑ **Grupo3:** RST, ATN. Definen las condiciones del bus



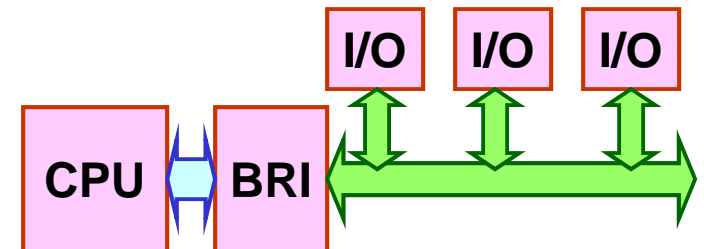
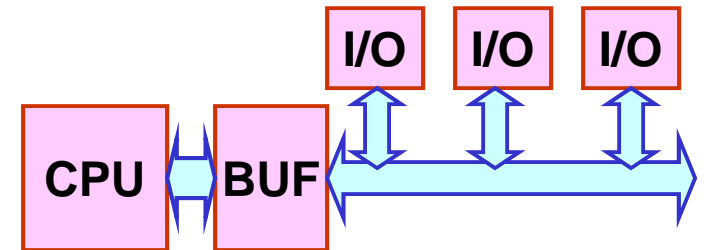
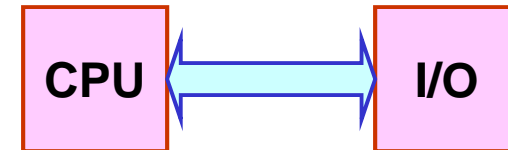
Notar las similitudes con CAN!



Alternativas de bus local

Alternativas:

- Conexión directa: caso de ciertas placas de video, que eran conectadas directamente al bus de la CPU. Es un sistema ad-hoc dependiente del tipo de procesador, de la placa de host y del periférico
- Conexión mediante buffers: esencialmente similar a la anterior, sólo que la existencia de buffers posibilita mayor carga eléctrica, aunque incorpora retardos.
- Conexión mediante un bridge: en este caso la CPU, además de conectarse a sus periféricos propios, se conecta a periféricos estándar a través de un bus abierto, con capacidad multimaster.



El bus PCI

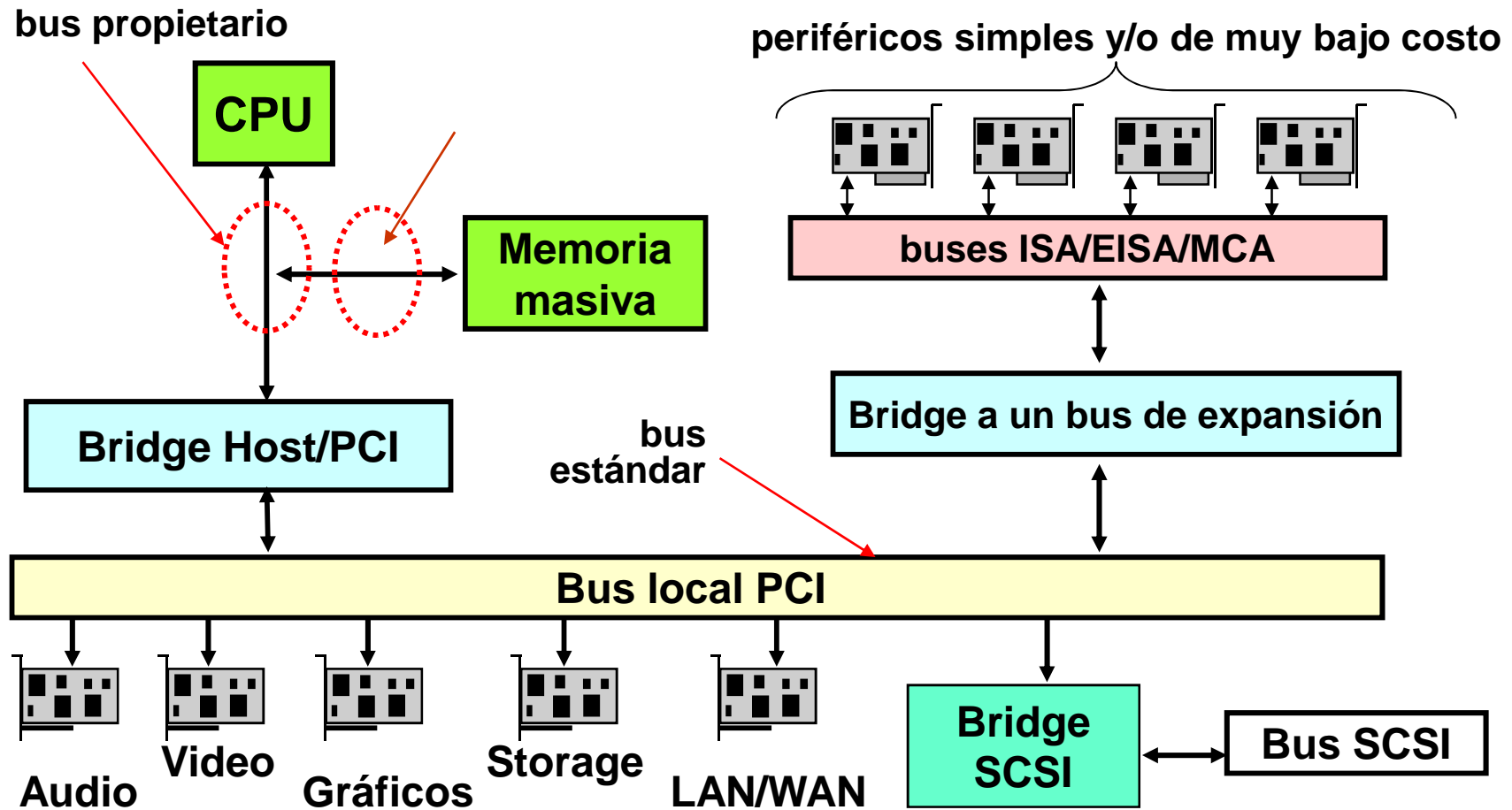
- ❑ Definido por **INTEL** en Junio de 1992 (v1.0), el nombre PCI deriva de ***Peripheral Component Interconnect***
 - ❑ Es un bus independiente del tipo de procesador
 - ❑ Detección y Configuración automática
 - ❑ Tres espacios de datos:
 - ❑ MEMORIA, I/O y CONFIGURACION
 - ❑ Todas las transacciones son de bloques (*burst transfer*)
 - ❑ Soporta hasta 256 buses PCI interconectados entre sí
 - ❑ Soporta hasta 256 dispositivos funcionales por bus PCI
 - ❑ Cada dispositivo físico puede tolerar hasta 8 dispositivos funcionales (o lógicos) distintos (*Multi-Function devices*)
 - ❑ Cada dispositivo lógico puede estar asociado con hasta 6 bloques de direcciones de memoria y/o entrada-salida

Conceptos básicos de PCI

- ❑ La **Revisión 2.0** (Abril 1993) define un clock de 33MHz, y la **Revisión 2.1** (Junio 1995) lo pasa a 66 MHz
- ❑ Con una extensión de ancho de datos de hasta 64 bits
- ❑ Elevada performance: picos desde 132 MByte/s (32 bits @ 33MHz) hasta 528 MByte/s (64 bits @ 66MHz)
- ❑ Accesos veloces: la latencia de un acceso de escritura *random* es de 60 ns a 33MHz (dos ciclos de clock)
- ❑ Validación de datos: mediante chequeo de paridad
- ❑ Arbitraje hidden que elimina tiempos de latencia
- ❑ Baja cantidad de pines: un bus PCI de 32 bits requiere sólo de 45 a 49 pines
- ❑ Es un bus “verde” (de bajo consumo eléctrico)

- ❑ **Iniciador de una transferencia (*Initiator* o *Master*)**
 - ❑ Capaz de iniciar transacciones
 - ❑ Puede transferir datos sin que el HOST lo interrumpa
- ❑ **Destinatario de una transferencia (*Target* o *Slave*)**
 - ❑ Puede enviar o recibir datos en forma pasiva
 - ❑ NO PUEDE iniciar una transacción
- ❑ **Agente:** Cualquier dispositivo presente en el bus PCI
- ❑ **Bridge Host/PCI**
 - ❑ Encargado de trasladar las transacciones entre el Local Bus del procesador HOST y el bus PCI. Usualmente con otras funcionalidades
- ❑ **Otros Bridges entre el bus PCI y otros buses**
 - ❑ PCI/ISA: para emplear periféricos estándar de baja performance
 - ❑ PCI/PCI: entre dos buses PCI

Arquitectura de un sistema PCI



Conceptos relacionados con PCI: Sistemas abiertos y cerrados

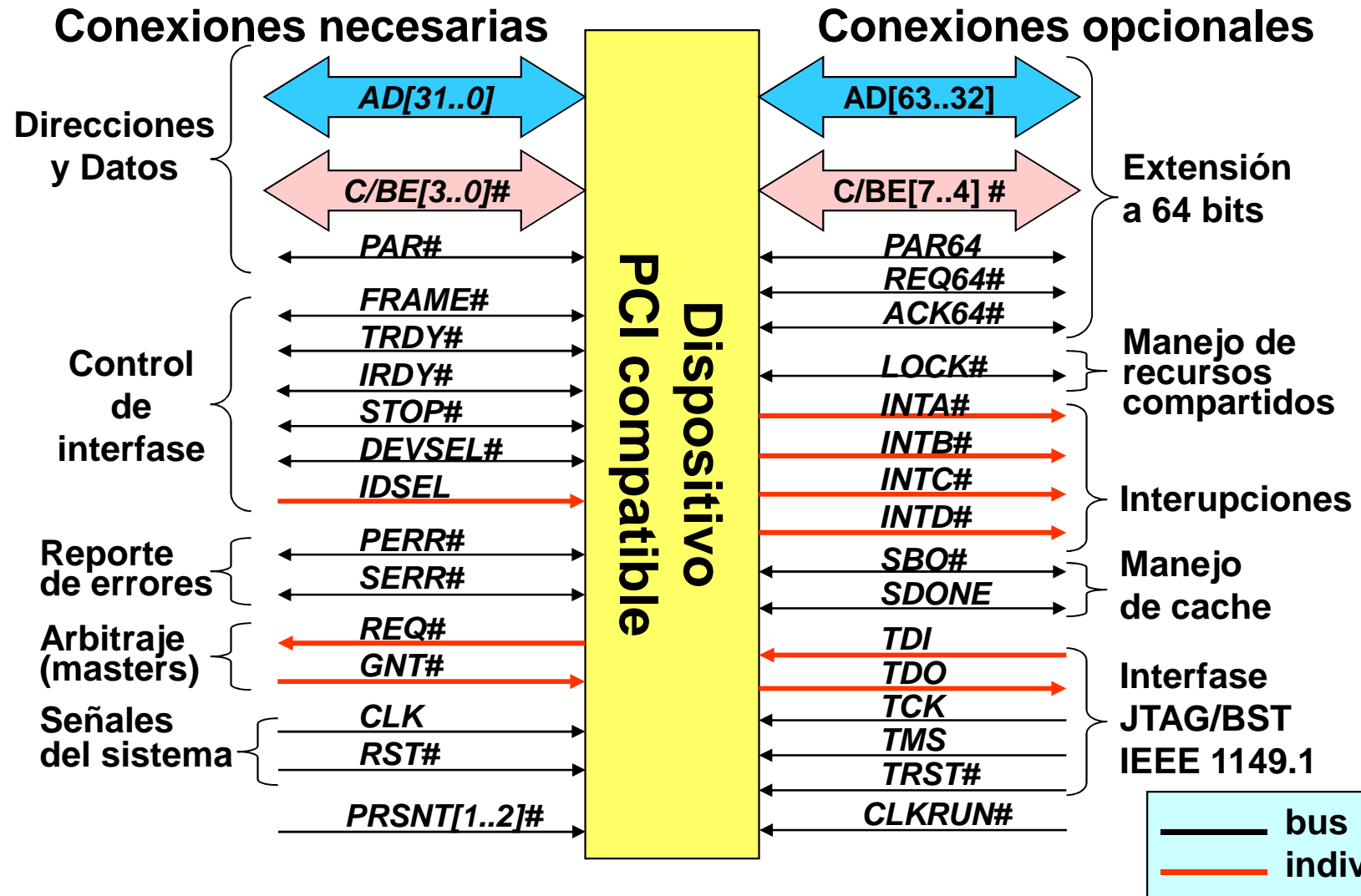
- **Sistemas abiertos**
- El caso típico de una PC vendida en forma de “partes para armar”.
 - Es necesario que componentes ya existentes o a diseñar en el futuro por proveedores independientes puedan operar juntos
 - El diseñador del bus desconoce *a priori* qué es lo que será instalado en su sistema
 - Se requiere estricta compatibilidad con la especificación PCI
- **Sistemas Cerrados**
- Es el diseñador quien elige qué es lo que será conectado en el sistema
 - No se requiere compatibilidad estricta con la especificación del bus PCI, con lo que pueden relajarse exigencias y así bajar costos
- Dónde es posible usar sistemas cerrados?
 - En equipos especiales (P.Ej: instrumental)
 - Intercomunicación entre plaquetas, o dentro de una plaqueta
 - Sistemas PCI con menos carga eléctrica
- Ventajas de los sistemas cerrados?
 - Se pueden disminuir las exigencias temporales. No es necesario operar a 33 MHz, y a menor velocidad, pueden aumentarse los valores de t_{CO} y t_{SU}
 - El dispositivo a usar puede ser más lento y más económico
 - Es común el uso de sistemas cerrados operando a 25-MHz

Conceptos relacionados con PCI: “reflected-wave switching”

- ❑ En PCI no se usan resistencias para adaptar la impedancia del bus.
- ❑ Al llegar la onda incidente a los extremos de alta impedancia del bus, se generan ondas reflejadas que aumentan el valor efectivo que la onda incidente ya había definido en las líneas del bus
- ❑ Recién al llegar estas ondas al driver ayudan a éste a llegar a su valor final, y se extinguen parcialmente en la baja impedancia del driver
- ❑ Esto determina un menor consumo eléctrico del driver, y la posibilidad de integrarlo en un ASIC-VLSI
- ❑ Para la correcta operación del bus es necesaria la existencia de diodos enclavadores que limiten las sobretensiones de hasta 11Volts que pueden aparecer
- ❑ En PCI los datos se capturan en la subida de CLK, lo que hace crítico el análisis de los tiempos de propagación. La norma define rigurosamente el largo de las pistas y sus características eléctricas.

❑ Este uso de las ondas reflejadas por desadaptación es llamado **“reflected-wave switching”**

El bus PCI



Terminología eléctrica

- ❑ En un bus PCI las señales tienen uno entre cinco posibles comportamientos:
 - ❑ **IN:** entrada estándar
 - ❑ **OUT:** salida siempre activa tipo TotemPole o PushPull
 - ❑ **T/S:** línea bidireccional, *tri-state*
 - ❑ **O/D:** salida *OpenDrain*. El sistema provee un pullup. Este tipo de líneas no es adecuado para líneas de alta velocidad de conmutación por la constante de tiempo creada por la resistencia y la capacidad parásita del bus
 - ❑ **S/T/S:** *Sustained TriState*. señal Tri-State. Quien pone esa salida en '0' debe volverla a '1' por un ciclo de CLK antes de pasar a Tri-State; un nuevo agente debe esperar un ciclo de CLK con la señal en Tri-State antes de excitarla. Requiere que el sistema central provea un pullup de valor elevado que mantiene la línea en "1", aunque la tarea de dejar la línea estacionada (*parked*) en ese valor es del agente.

Ciclos de una transacción

Una transacción describe un proceso completo de intercambio de datos entre un Master y un Slave, y se compone de las siguientes etapas:

- ❑ **PEDIDO DEL BUS**: un agente con capacidad de operar como Master solicita al árbitro del sistema el manejo del bus y queda a la espera que le sea asignado
- ❑ **DEFINICIÓN DEL COMANDO Y FASE DE DIRECCIONAMIENTO**: una vez en control del bus el Master realiza un ciclo donde define qué comando será ejecutado y la dirección inicial desde o hacia donde se realizará la transferencia
- ❑ **FASE DE INTERCAMBIO DE DATOS**: una vez definida la dirección se realizan una o más transacciones sucesivas de movimiento de datos
- ❑ **EL CIERRE DE UNA TRANSACCIÓN**: cuando la transacción termina, ya sea porque se cumplieron sus objetivos, ya sea por otras múltiples condiciones, se cierra la transacción

Señales básicas del sistema

☐ **CLK: - PCI Clock - Input**

- ☐ Es una entrada común a todos los dispositivos en el bus PCI. Puede valer desde 0 a 33MHz.

☐ **RST#: - RESET - Input**

- ☐ Fuerza todos los registros de configuración, máquinas de estado y drivers de salida a un estado predefinido (en general Tri-State).

Bus de datos y direcciones

AD[31..0]: Address & Data, T/S

AD [31..0] Bus multiplexado de direcciones y los datos.

Una transacción se compone de una (*) **fase de direccionamiento** seguida por una o más **fases de datos**.

- ❑ La **fase de direccionamiento** es el ciclo de CLK donde las líneas **AD[31..0]** son empleadas para transportar una dirección.
- ❑ En la (o las) **fases de datos** las líneas AD[31..0] son usadas para mover datos entre el INITIATOR y el TARGET, pudiendo transferirse hasta 32 bits en cada ciclo de reloj.

Por ello, **PCI** soporta de modo inherente las transacciones por bloques (o “bursts”).

(*) En el caso especial de direccionamientos de 64 bits (**DAC-Dual Address Cycle**) la **fase de direccionamiento** ocupa dos ciclos de clock.

AD[31..0]: la fase de direccionamiento

La **fase de direccionamiento** es el ciclo de CLK donde se activa **FRAME#**; allí las líneas **AD[31..0]** son empleadas para transportar una dirección.

- ❑ En accesos de memoria las líneas **AD[31..0]** corresponden a direcciones de bytes, pero como los datos están alineados a DWORDs, al inicio de una transacción el valor de **AD[1..0]** sólo puede ser 00 (direccionamiento lineal) o 10 (caso especial cuando se usan memorias cache). En este último caso, si el **TARGET** no puede aceptar este tipo de operación debe rechazar el comando (*TARGET Disconnect*)
- ❑ En accesos de I/O las líneas **AD[31..2]** referencian a un DWORD en el espacio de I/O, en tanto los cuatro posibles valores de **AD[1..0]** indican el byte menos significativo al cual el MASTER pretende acceder, y en base a él es que el TARGET activará **DEVSEL#**.
- ❑ Junto con **C/BE[3..0]#** este método permite fraccionar el espacio de I/O y acceder a puertas desde 8 a 32 bits. No tiene sentido que a través de **C/BE[3..0]#** se intente acceder a un byte ubicado en posición menos significativa a la descrita por **AD[1..0]**

Bus de datos y direcciones

- ❑ En la (o las) **fases de datos** las líneas **AD[31..0]** son usadas para mover *datos* entre el INITIATOR y el TARGET.
- ❑ AD[7..0] transporta el LSB (byte menos significativo) y AD[31..24] el más significativo (MSB)
 - ❑ Los datos a escribir se consideran estables y válidos cuando se activa la línea de control **IRDY# (Initiator Ready)**, los datos a leer cuando se activa la línea de control **TRDY# (Target Ready)**.
 - ❑ La transferencia se realiza en el ciclo de reloj donde tanto **IRDY#** como **TRDY#** están activos
 - ❑ Esta interacción entre **IRDY#**, **TRDY#** y **CLK** configura un proceso de HandShake que permite controlar el ritmo con el que el Initiator y el Target suministran y aceptan datos

C/BE[3..0]#: BusCommand & ByteEnables - T/S

- ☐ En la **fase de direccionamiento** estas líneas definen el comando a ejecutar.
- ☐ En la **fase de datos** son usadas para indicar cuál de los 4 bytes presentes en **AD[31..0]** transporta información válida (C/BE[0]# se refiere al LSB y C/BE[3]# al MSB).
- ☐ En una transferencia de escritura el MASTER debe poner valores estables en los bytes no usados, y ese valor será usado para el cálculo de paridad. Lo mismo vale para un TARGET durante una lectura

PAR: Paridad - T/S

- ☐ Cómputo de paridad sobre las líneas AD[31..0] más C/BE[3..0]# (también de los bytes de datos no usados). Su valor hace que el número de “1”s en AD[31..0] más C/BE[3..0]# más PAR sea par.
- ☐ PAR es OBLIGATORIO y se presenta al bus con un ciclo de retardo.
- ☐ El MASTER genera PAR para ciclos de direccionamiento y ciclos de escritura, en tanto el TARGET genera PAR en los ciclos de lectura

FRAME#: Cycle Frame, S/T/S en el Master, e IN en el Target

- ❑ Generada por el **MASTER** sirve para señalar el inicio Y la duración de una transacción.
- ❑ Para detectar que le ha sido asignado el control del bus, un **MASTER** debe verificar que **FRAME#** e **IRDY#** estén inactivas y **GNT#** activa en un mismo flanco activo de clock.
- ❑ **FRAME#** es desactivada por el **MASTER** cuando está listo a realizar la última fase de datos.

IRDY#: Initiator Ready. S/T/S (Master), IN (Target)

- ❑ Generada por el MASTER.
- ❑ En un ciclo de escritura indica que el MASTER está colocando datos válidos en el bus. **IRDY#** y **TRDY#** activas en un flanco activo de CLK indican la culminación de un ciclo de escritura.
- ❑ En un ciclo de lectura indica que el MASTER está listo para aceptar los datos que el TARGET coloque en el bus

TRDY#: Target Ready. IN (Master), S/T/S (Target)

- ❑ Generada por el TARGET
- ❑ En un ciclo de lectura indica que el TARGET está colocando datos válidos en el bus
- ❑ En un ciclo de escritura la activación de **TRDY#** estando **IRDY#** en un flanco activo de CLK activada indica la culminación de una fase de datos.

STOP#: IN (Master), S/T/S (Target)

- Es el medio por el cual el TARGET pide al INITIATOR detener la transacción en progreso en el ciclo corriente

DEVSEL#: IN (Master), S/T/S (Target)

- señal activada por un TARGET cuando identifica la dirección en el bus como propia. En el bus, DEVSEL# activa indica que un dispositivo ha sido seleccionado (lo que será útil para la decodificación sustractiva)

IDSEL#: IN (Master), IN (Target)

- línea punto a punto (***sistema-->agente***) usada como “chip select” durante transacciones de lectura y escritura en el espacio de configuración.

Señales de arbitraje (sólo para MASTERS)

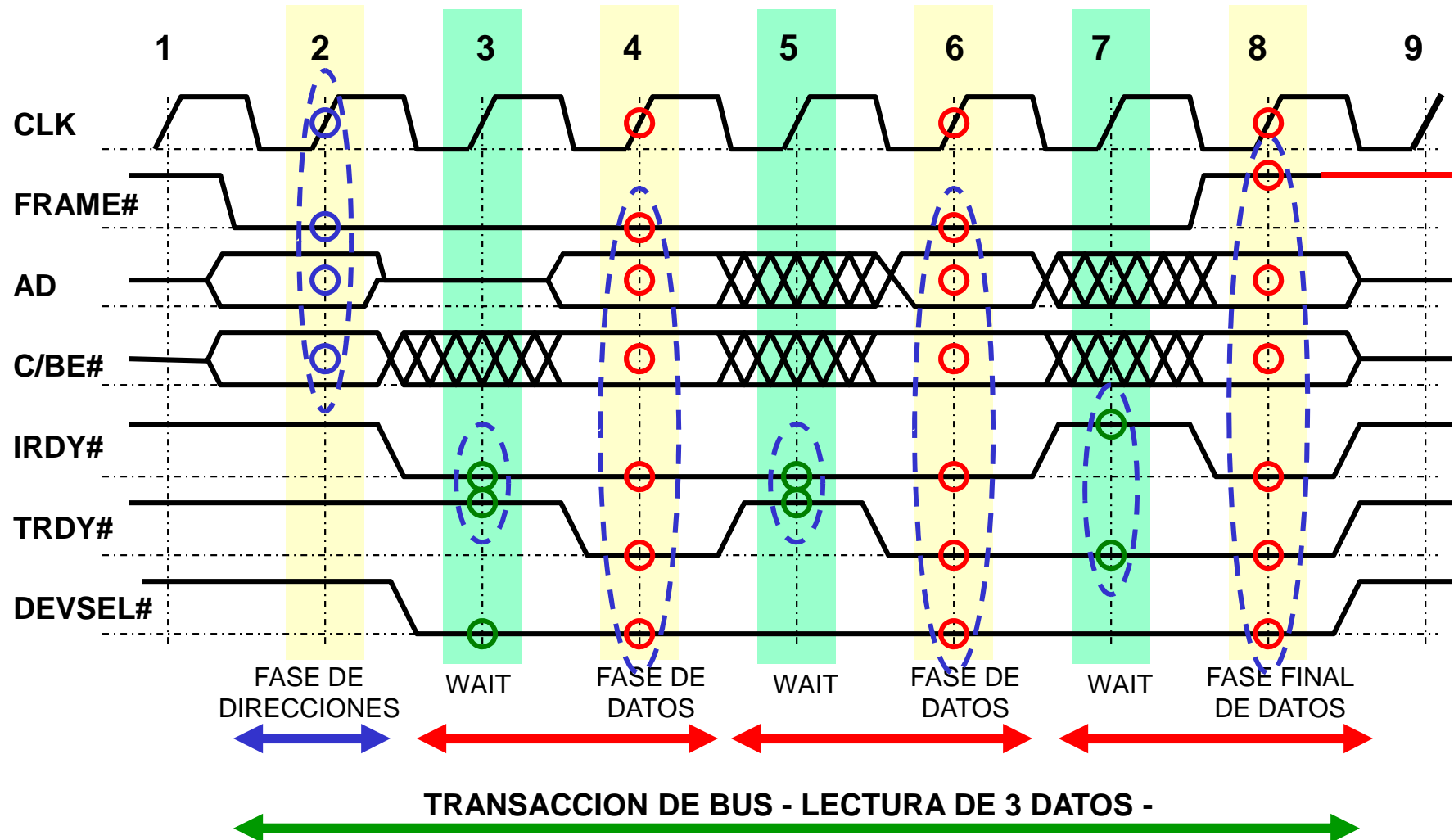
REQ#. Request, T/S

- Indica al árbitro que el agente desea hacer uso del bus.
- Es una señal punto a punto (**agente --> árbitro**)
- Cada MASTER tiene su propia REQ#, la que debe ser pasada a tri-state cuando RST# se activa

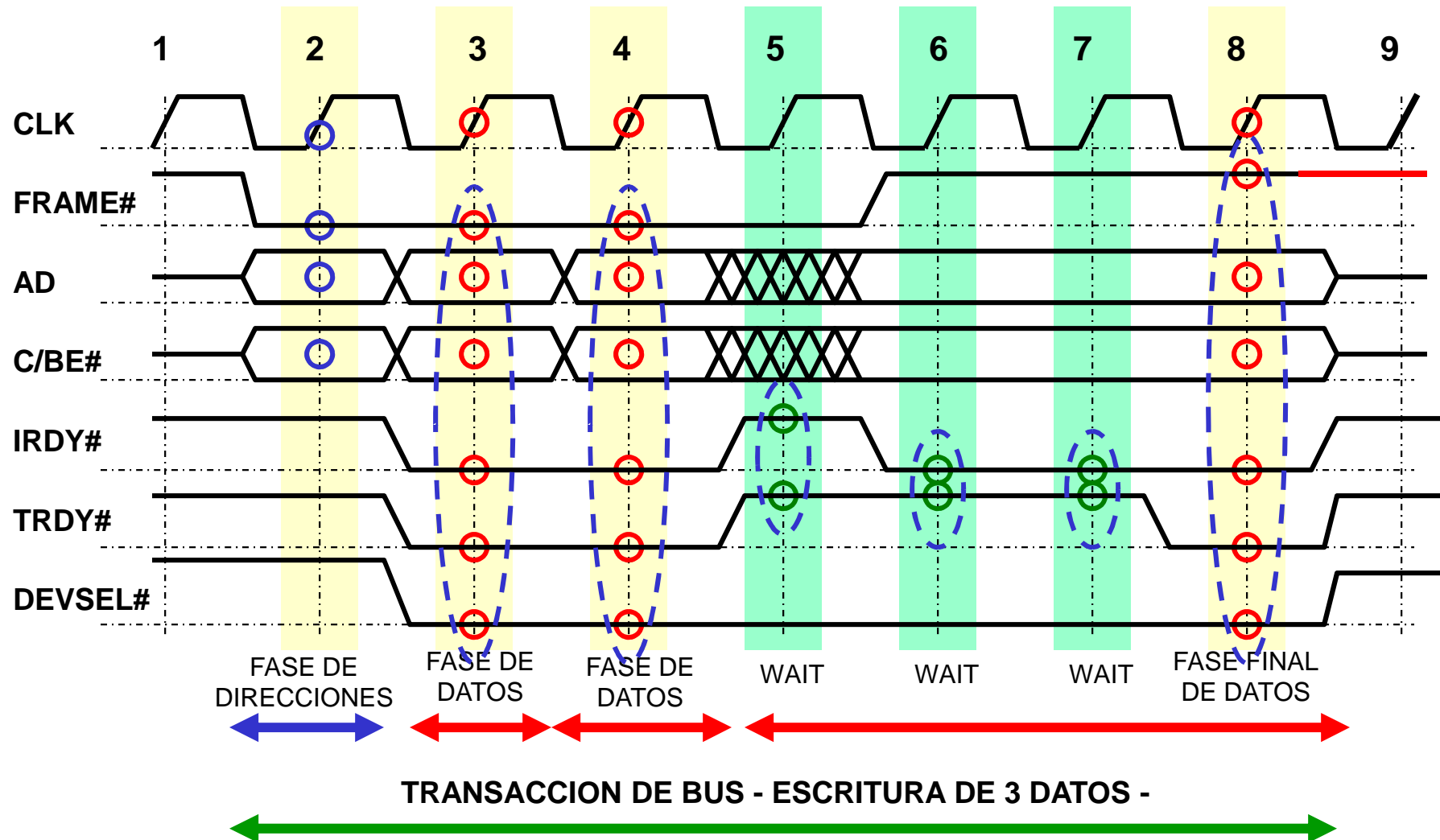
GNT#. Grant, T/S

- Generada por el árbitro, es el medio por el cual concede el derecho a uso del bus a un agente.
- Es una señal punto a punto (**árbitro-->agente**)
- Cada MASTER tiene su propia GNT#, cuyo valor debe ignorado cuando RST# se activa

Transacción elemental de lectura



Transacción elemental de escritura



La latencia del bus

- PCI está diseñado para presentar baja latencia de acceso, en forma predecible.
- Esta latencia está definida inicialmente por el arbitraje, y luego por los MASTERS y los TARGETS
- Cada MASTER tiene limitada la posesión del bus en condiciones de tráfico elevado y para ello tiene un Latency Timer. En función de éste, de las líneas FRAME# y GNT#, y del comando en curso, debe liberar el bus siguiendo ciertas reglas
- MASTERS y TARGETS están también limitados en la cantidad de ciclos de WAIT que pueden añadir a una transacción

La latencia del bus

Latencia del MASTER es la cantidad de WAITs hasta que el MASTER activa IRDY# luego de haber activado FRAME#, y para subsecuentes transferencias

Todos los MASTER deben activar IRDY# dentro de los 8 CLK de haberlo hecho con FRAME#, y si desactivan IRDY# los tiempos entre sucesivas activaciones tampoco deben superar los 8 CLK

Latencia del TARGET es la cantidad de WAITs antes que el TARGET active TRDY# (o STOP#)

- Latencia inicial es el número de CLK desde que se activa FRAME# hasta que lo hace TRDY# (fase inicial de datos) o STOP# (Retry y Target Abort); está limitada a 16 ciclos de CLK.
- Latencia subsecuente es el número de CLK desde que IRDY# y TRDY# se activan en un ciclo de datos hasta que IRDY# y TRDY# (o STOP#) lo hacen en el siguiente; está limitada a 8 CLK

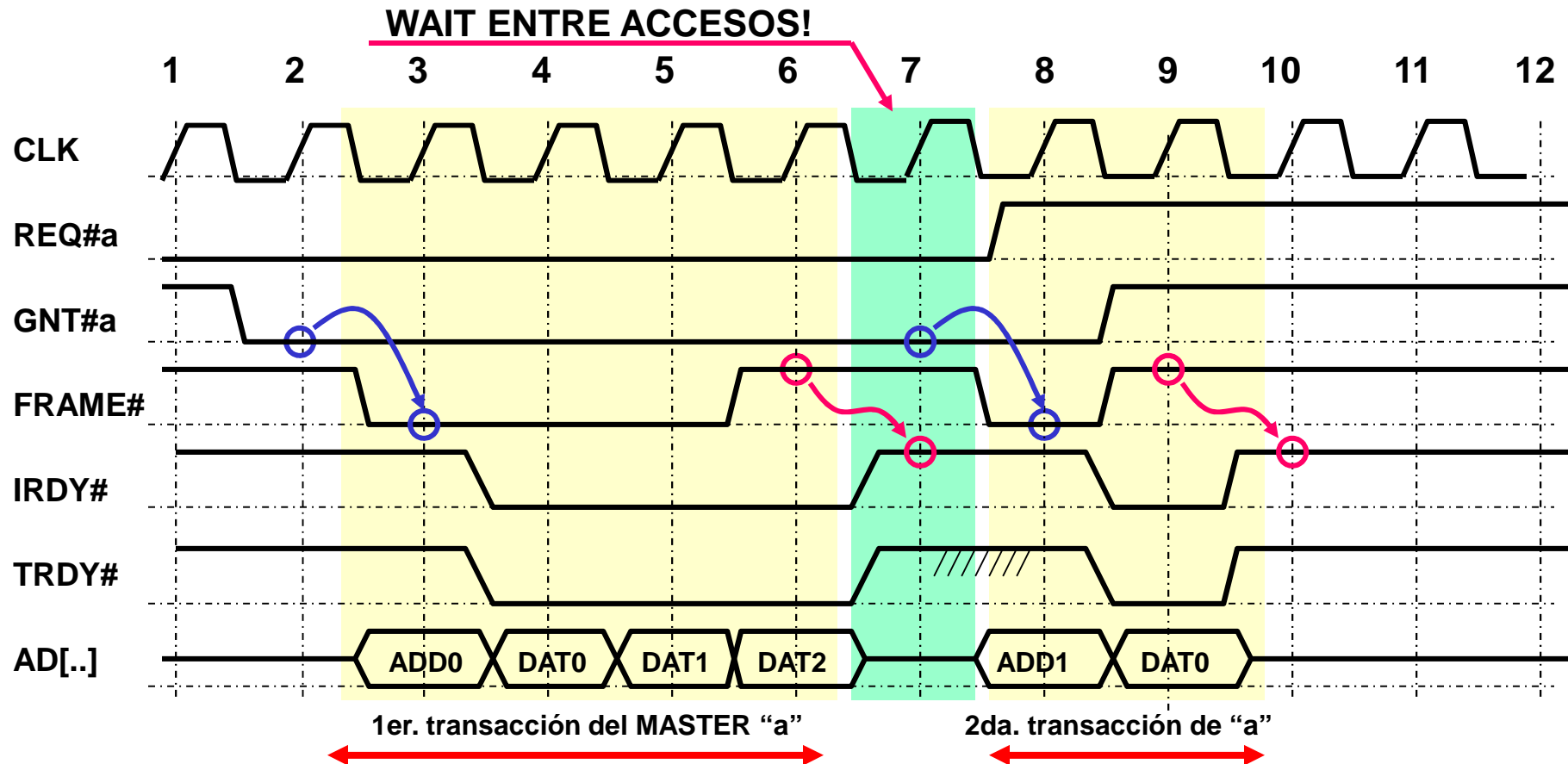
Address/Data stepping

- ❑ En algunas aplicaciones de PCI no existen exigencias de velocidad, pero sí de bajo ruido, acoplamiento entre líneas (crosstalk), o EMI (generación de interferencia electromagnética).
- ❑ Uno de los elementos más conflictivos surge de la conmutación simultánea de las 32 líneas AD[..], que puede provocar fuertes picos instantáneos de corriente de alimentación
- ❑ Se llama “stepping” a extender el proceso de inserción de direcciones y datos a lo largo de varios ciclos, mediante ciclos de WAIT para el cambio gradual de las líneas AD[..] de a bloques. Por ejemplo, dividiendo AD[] en 4 pasos, en cada paso y en el caso peor sólo conmutarán 8 buffers.

Transacciones BACK to BACK

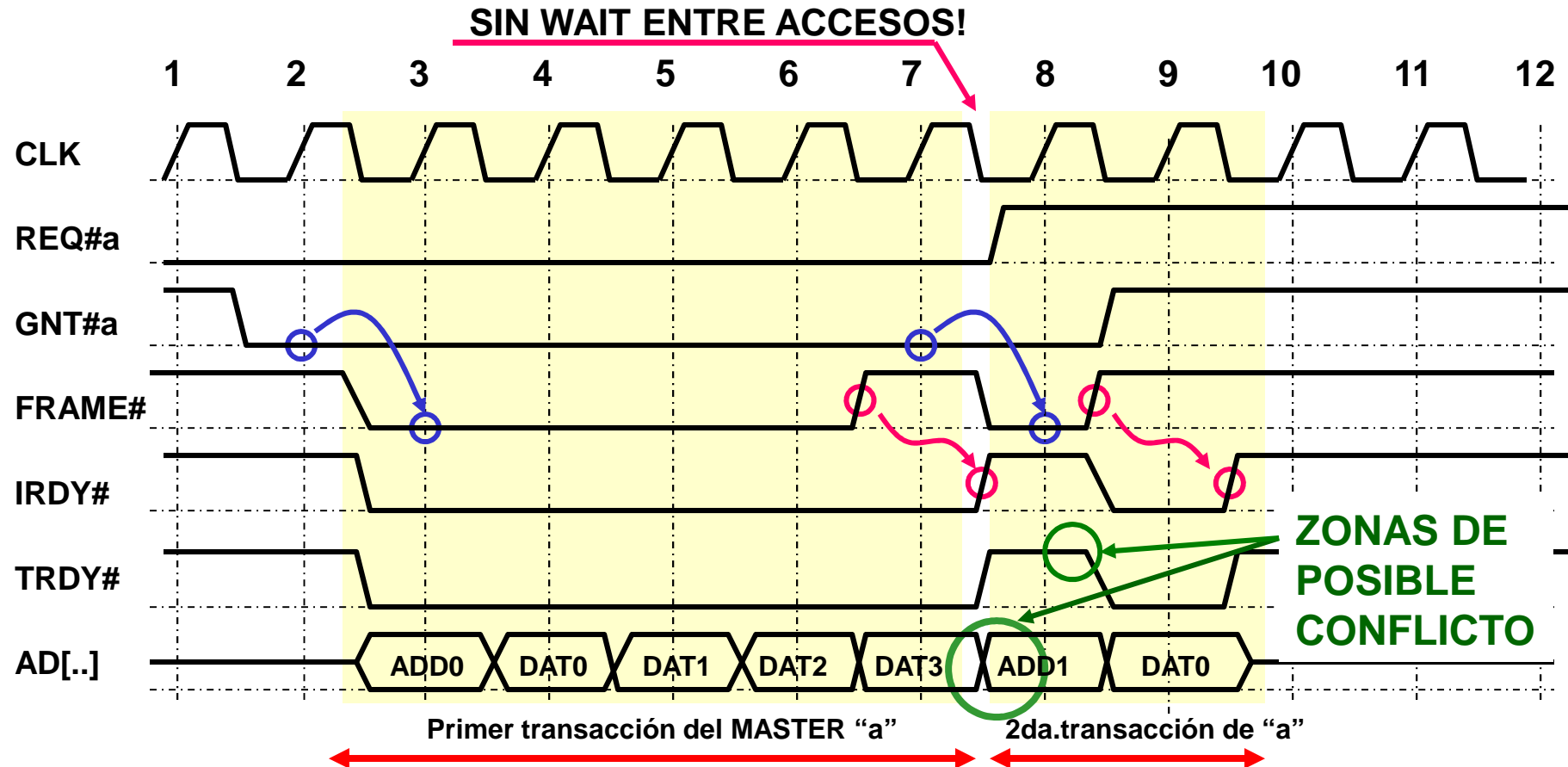
- En ciertos casos un MASTER tiene pendientes de realizar varias transacciones.
- Para ello no libera el bus mediante **REQ#** al terminar una de ellas, a la espera que el árbitro le siga dando control del bus para realizar las aún pendientes en sucesión.
- Si entre transacción y transacción el MASTER inserta un ciclo de WAIT este tipo de transacciones sucesivas es llamada **BACK to BACK**.
- Si el MASTER no inserta este ciclo de WAIT se generan las llamadas transacciones **FAST BACK to BACK**. En este caso es posible que el/los TARGET de transacciones sucesivas puedan colisionar entre sí en ciertas líneas, lo que obliga a recaudos especiales.

Transacción normal Back-to-Back



Sin perder dominio del bus (GNT# sigue en '0') el MASTER lo retoma en el mismo ciclo en que lo libero para un segundo acceso a quizás otro TARGET. Durante el periodo 7-8 el primer TARGET ha tenido tiempo de pasar TRDY# y DEVSEL# (y STOP# o PERR#) a S/T/S por lo que otro acceso no provoca colisiones

Transacción FAST Back-to-Back



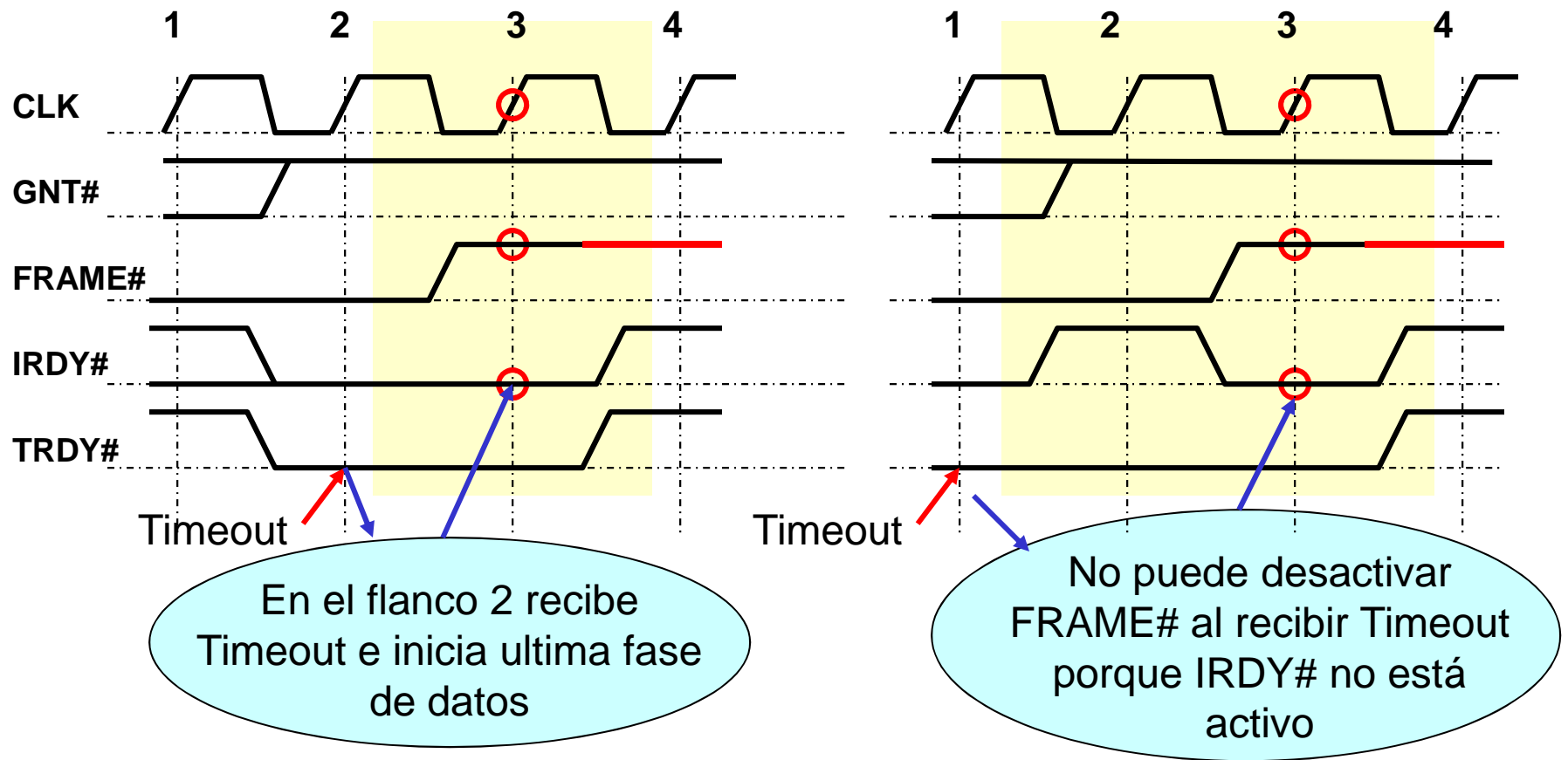
Sin haber perdido el dominio del bus (GNT# sigue en '0') el MASTER retoma el bus en el mismo ciclo en que lo libero para un segundo acceso a quizás otro TARGET. Acá la única forma de evitar colisiones es acceder al mismo TARGET o que los tiempos de respuesta de distintos TARGET garanticen la ausencia de conflictos (bus contention) en AD[], TRDY#, DEVSEL#, PERR# y STOP#.

Cierre de transacciones

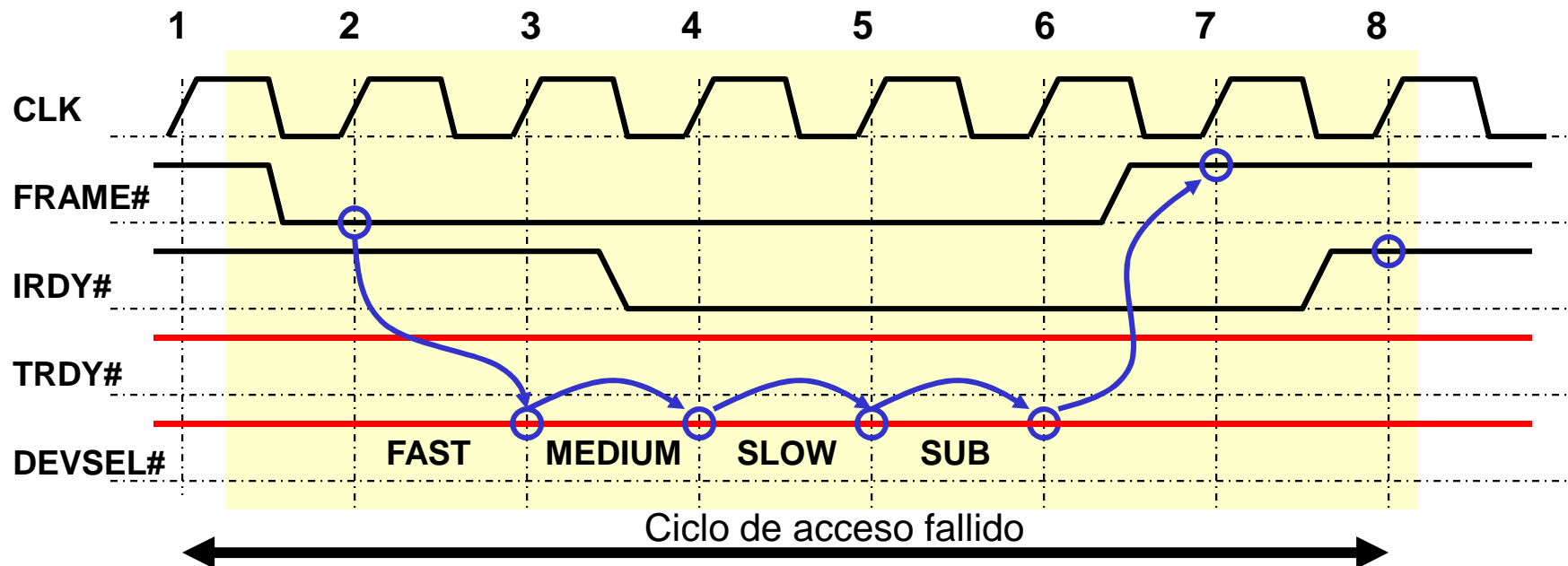
Una transacción puede ser terminada por el MASTER o por el TARGET

- el MASTER comienza el cierre una transacción desactivando FRAME# con IRDY# activada, y lo completa cuando ambas se desactivan.
- este cierre puede deberse:
 - a que la tarea fué cumplida
 - a que el Latency Timer ha llegado a su tope
 - o a la ausencia de respuesta por el TARGET dentro de un período de tiempo (*Master-Abort*)

Cierre por el MASTER por Timeout



Cierre por el MASTER por Master Abort

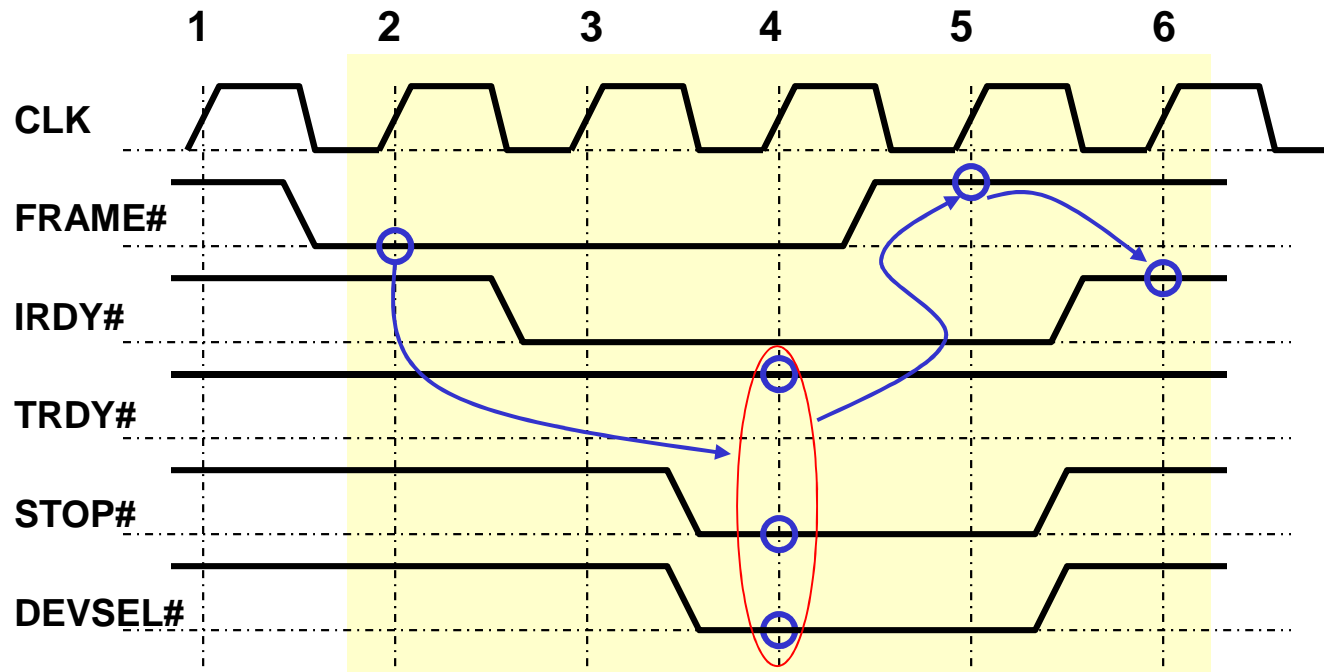


El pedido del MASTER es hecho público en (2) y el TARGET no responde mediante DEVSEL# en (3), (4), (5) ni en (6), por lo que en (7) el MASTER inicia la desconexión desactivando FRAME# y lo culmina en (8) desactivando IRDY#

Si un TARGET no puede completar una transacción puede usar STOP# para iniciar su cierre. Éste cierre puede ser del tipo:

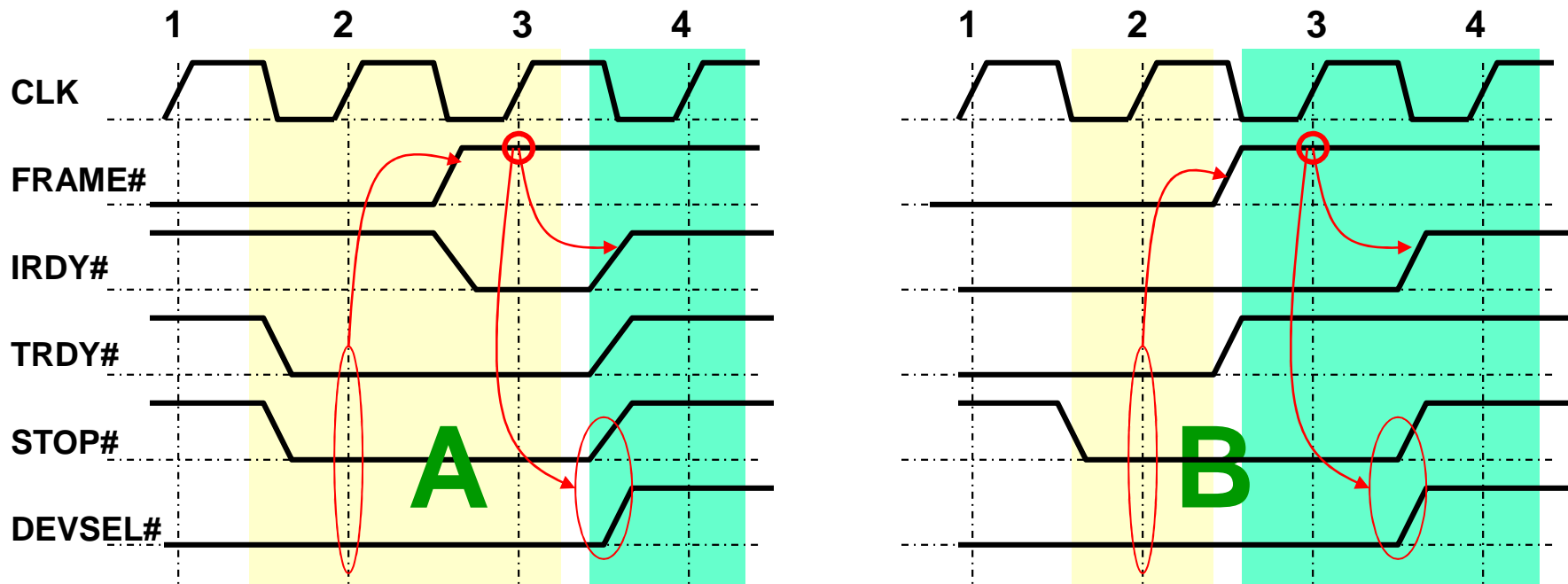
- Retry: la transacción se termina antes de transferir ningún dato porque el TARGET está ocupado, no puede satisfacer la Latencia Inicial, está reservado por otro MASTER, u otra razón. Se define activando STOP# con DEVSEL# activo, sin haber activado TRDY#.
- Disconnect: la transacción se cierra cuando ya se ha transferido algún dato porque el TARGET no puede cumplir con la Latencia Subsecuente y continuar en modo burst. Se define activando STOP# con DEVSEL# activo, activando o no TRDY#.
- Target-Abort: cuando el TARGET detecta una condición que le imposibilita cumplir la transacción. Se define activando STOP# y desactivando DEVSEL# en el mismo ciclo.

Pedido de cierre por el TARGET tipo Retry



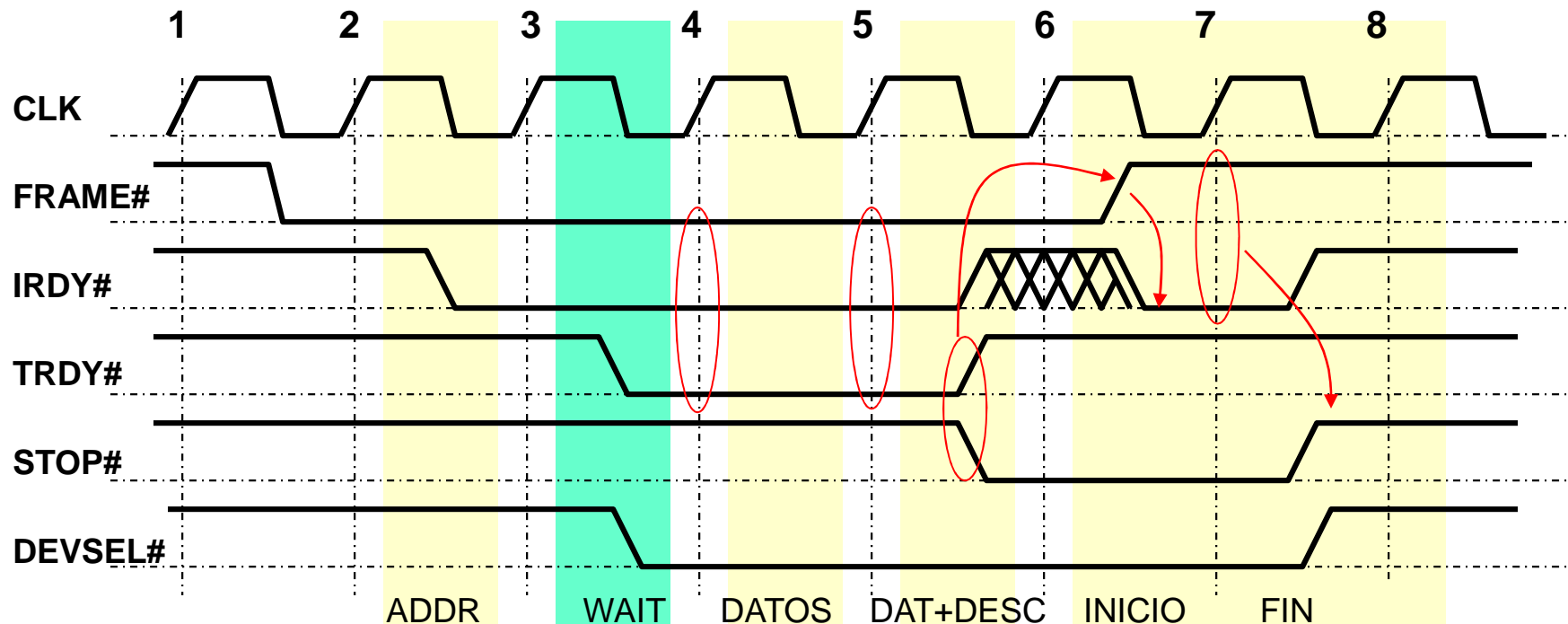
- El MASTER direcciona al TARGET en (2) mediante FRAME#.
- En (4) el TARGET se identifica con DEVSEL# pero a la vez activa STOP# para señalar que está imposibilitado para transferir datos en ese momento.
- El MASTER inicia la desconexión en (5) desactivando FRAME# y la culmina en (6) desactivando IRDY#

Cierre por el TARGET tipo Disconnect With Data



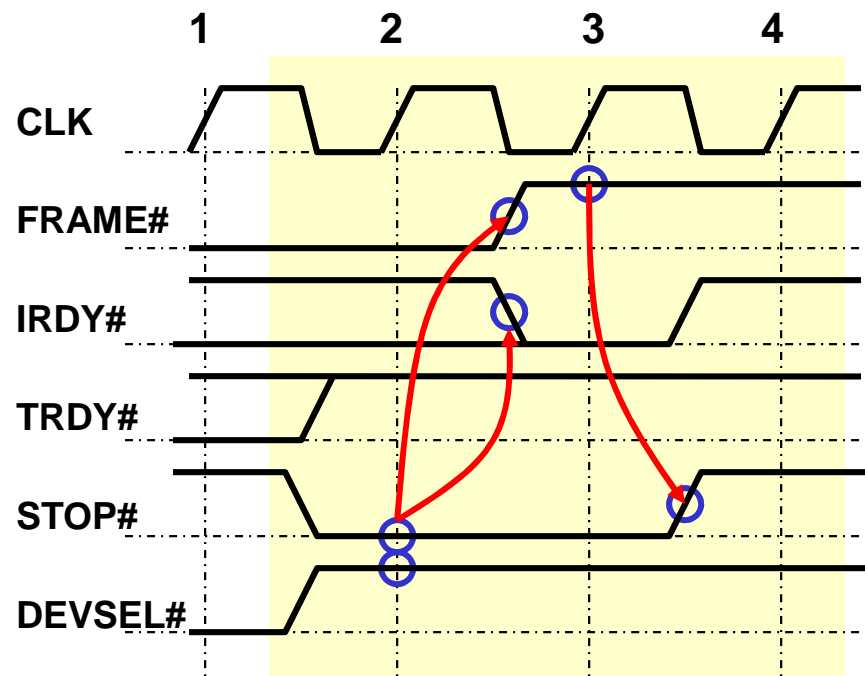
- En “A” y “B” en (2) el TARGET señala su intención de finalizar la transacción (TRDY# activa) y cerrar la conexión (DEVSEL# y STOP# activas).
- En el caso “A” el TARGET espera a que el MASTER pueda completar la transacción (activando IRDY#) y aceptar la desconexión (desactivando FRAME#).
- En el caso “B” el MASTER estaba listo por lo que en (3) inicia la desconexión.
- En ambos casos al iniciarse el cierre el TARGET desactiva STOP# y DEVSEL#

Cierre por el TARGET tipo Disconnect Without Data



- El TARGET solicita el cierre de la conexión activando STOP# en (6), pero a la vez informa que no puede recibir más datos desactivando TRDY#.
- El MASTER inicia la desconexión en (7) desactivando FRAME# y la culmina en (8) desactivando IRDY#

Cierre por el TARGET tipo Target Abort



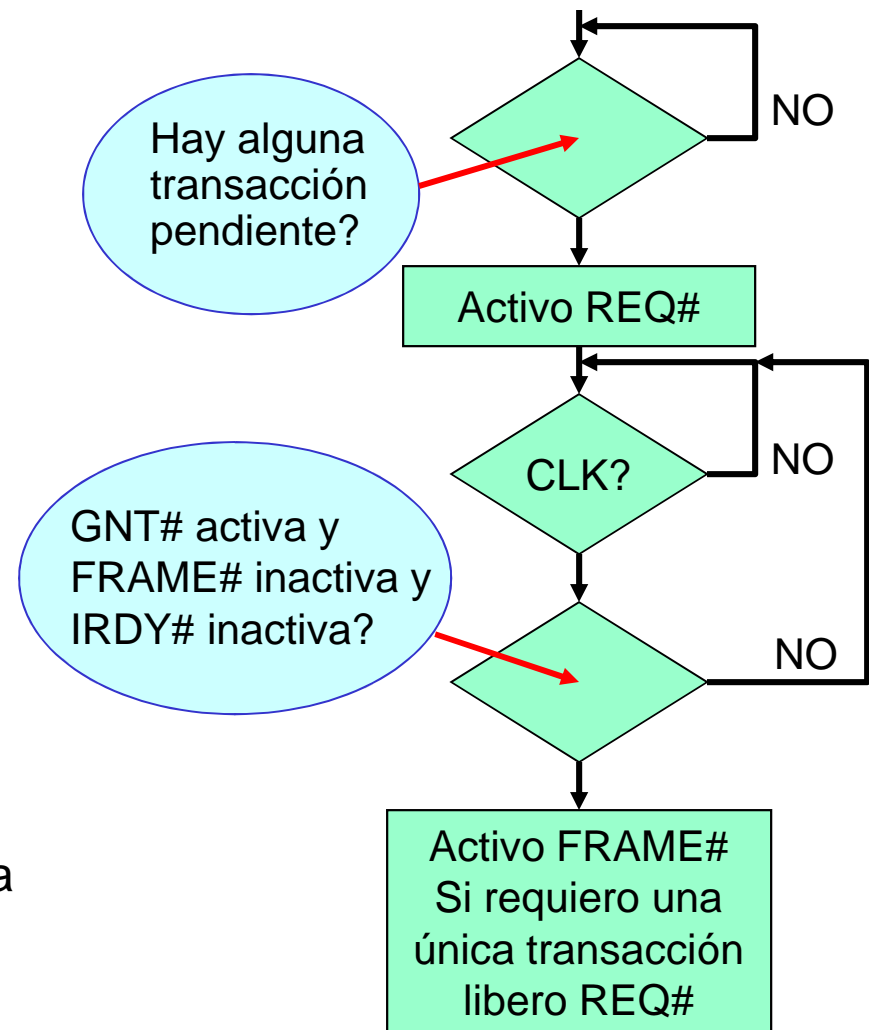
- El MASTER había direccionado al TARGET (FRAME# activo) y el TARGET se había identificado (DEVSEL# activa).
- No importa si hubo o no fase de datos previas, si el TARGET es incapaz de cumplir el pedido vigente en (1) desactiva TRDY# y DEVSEL# y activa STOP# en (2).
- El MASTER inicia la desconexión en (3) activando IRDY# y desactivando FRAME#, el TARGET libera STOP#, y la desconexión culmina en (4) al desactivarse IRDY#

El pedido del bus: algoritmos de arbitraje

- PCI no define un algoritmo de arbitraje, que debe ser definido en forma externa para satisfacer requerimientos de latencia y/o performance. La única exigencia es que en un flanco de CLK sólo esté activa una única línea **GNT#**
- Una vez iniciada una transacción (el MASTER activa FRAME#) **GNT#** puede ser desactivada sin por ello interrumpirla
- El árbitro puede remover **GNT#** a un master al que se ha dado el bus (**REQ#** y **GNT#** activas) si no activa FRAME# por 16 clocks
- Un agente debe pedir ser MASTER activando **REQ#**, sólo si necesita hacer uso del bus, y NO para bus parking. Bus parking debe ser controlado por el árbitro, quien activa la línea **GNT#** del propietario por *default* del bus.
- Las **GNT#[i]** pueden cambiar en cada CLK. Cada MASTER debe validar su **GNT#** antes de iniciar una transacción
- *No conviene remover **GNT#** del MASTER actual para habilitar el default hasta que el bus pase a IDLE.*
- *Si una transacción es terminada por un TARGET (usando STOP#) la línea **REQ#** debe desactivarse por 2 CLK para que otro agente pueda usa al bus.*

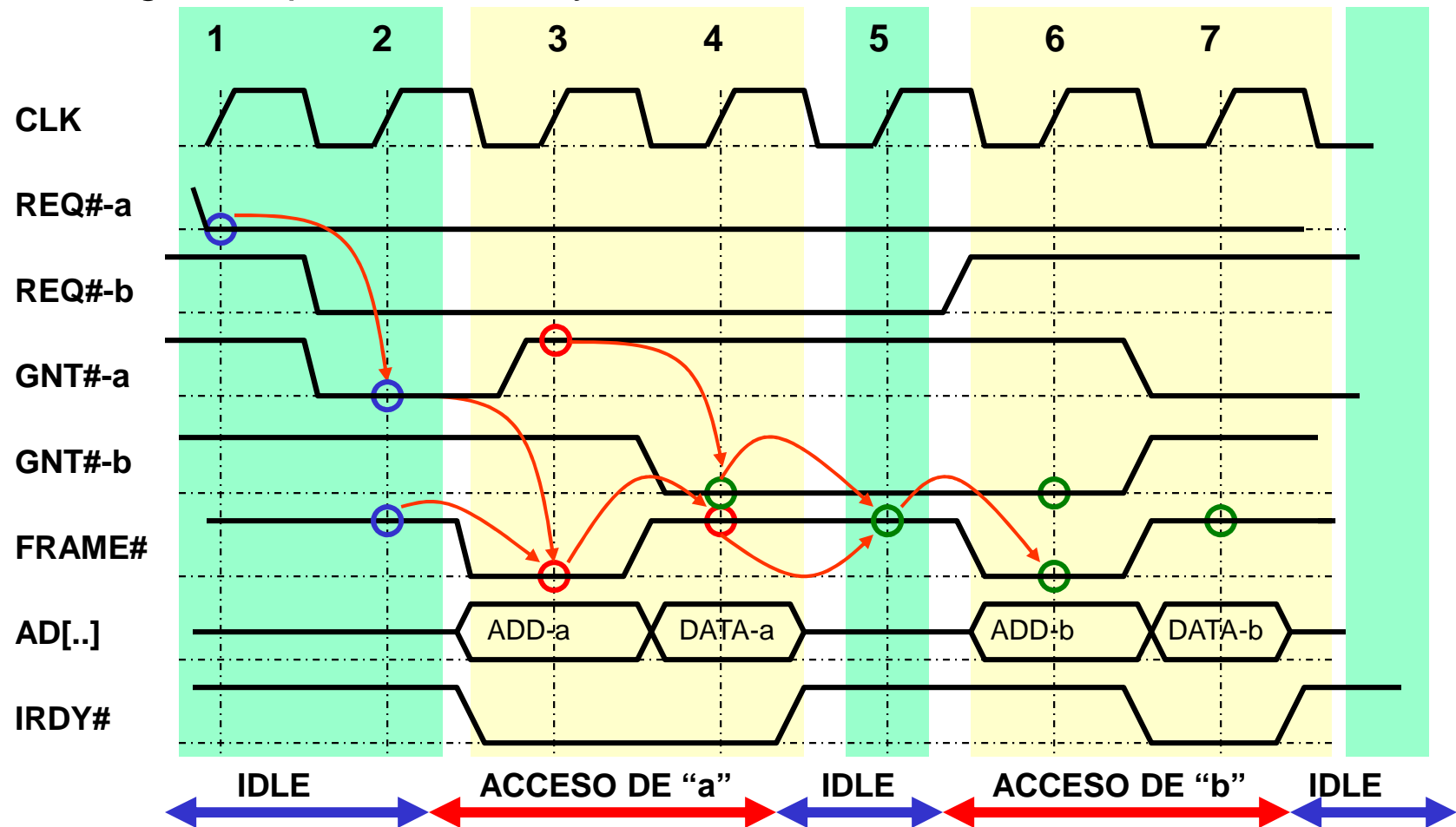
Toma del bus por un MASTER

- Un agente puede mantener **REQ#** activa todo el tiempo que necesite el bus. Si no lo hace puede forzar a rearmar el bus perdiendo tiempo
- Si un agente requiere una única transacción activa **FRAME#** y desactiva **REQ#** en el mismo ciclo, para que el árbitro tenga tiempo para arbitrar un posible nuevo MASTER pendiente
- Para tomar el control del bus un agente debe verificar que le ha sido dado el permiso (**GNT#**) y que el agente previo ha liberado el bus (**FRAME#** e **IRDY#**)



Secuencia básica de arbitraje

Dos agentes piden acceso y el árbitro les brinda en forma alternada



PERR#: Parity Error, S/T/S

- Indica errores de paridad en todos los ciclos excepto el Special Cycle.
- Es activada por el agente que recibe datos, y al ser desactivada debe ser mantenida en ese estado por un ciclo de CLK antes de pasar a Tri-State

SERR#: System Error, O/D

- Indica errores de paridad en los ciclos Special Cycle, o todo otro error que pueda ser catastrófico.
- Es activada durante un ciclo de CLK y está asociada a la generación de un NMI.

Líneas de interrupción

INTA#, INTB#, INTC#, INTD#: O/D

- Las interrupciones son líneas opcionales, activas bajas y por nivel, y su operación es asincrónica respecto a la señal de CLK
- Son salidas OPEN-DRAIN, y el sistema es el encargado de forzar un nivel alto mediante un pull-up. Una vez activado, un pedido de INT# debe permanecer en ese estado hasta ser removido por software
- Para un dispositivo mono-función sólo se habilita el uso de INTA#, en tanto los dispositivos multi-función pueden usar todas las líneas de interrupción.

Las interrupciones

- Un dispositivo ***MonoFunción*** es aquel que engloba dentro de un único dispositivo físico a un único dispositivo lógico. Un dispositivo de este tipo que requiera atención del software por interrupción deberá usar exclusivamente INTA# y programar este valor en el registro de configuración Interrupt Pin.
- Un dispositivo ***MultiFunción*** engloba dentro de un único dispositivo físico entre 2 y 8 dispositivos lógicos, y este tipo de dispositivos podrá usar hasta 4 interrupciones (INTA# a INTD#). Cada dispositivo lógico deberá programar qué pin usa en el registro de configuración Interrupt Pin.
- El hecho que sólo se definan 4 líneas no quiere decir que sólo ingresen 4 líneas al administrador de interrupciones, sino usualmente muchas más, lo que depende de cada back-plane

Las interrupciones

- Terminada la configuración, cuando el HOST conoce qué dispositivo está conectado a qué línea en qué slot y a que interrupción real del router corresponde, este número debe ser programado en el registro de configuración Interrupt Line
- A su vez, previo a la habilitación de las interrupciones, el sistema debe instalar la tabla de vectores que conectan cada número de interrupción a cada rutina de atención.
- Estas rutinas pueden estar en la BIOS ROM, en una ROM asociada a cada dispositivo, o en un driver cargado por el sistema operativo.
- En el caso que varios dispositivos compartan el mismo número REAL de interrupción el software deberá definir los esquemas de arbitraje a emplear para resolver requerimientos simultáneos.

El espacio de configuración

- Cada dispositivo PCI puede tener hasta 64 DWORDS con registros que definen ciertas características de operación. De estos 64 DWORDs, los primeros 16 DWORDS corresponden al encabezado, y los restantes a registros propios del dispositivo
- De los registros del encabezado, existen 8 registros que son imprescindibles (*mandatory*), en tanto otros son opcionales
- De los obligatorios, hay registros READ-ONLY de valor estático, registros con escritura habilitada, y otros con sólo la escritura de '0's habilitada.
- Sólo puede acceder a esos registros el HOST, quien debe poner en C/BE[..] el comando *Configuration Read* o *Configuration Write* en la fase de direccionamiento, y activar durante esta fase la línea punto a punto **IDSEL** del agente seleccionado.
- El valor de AD[10..8] en la fase de direccionamiento define cuál de los 8 posibles dispositivos internos de un MultiFunction es usado, en tanto el valor de las líneas AD[7..2] define cuál DWORD de los 64 posibles del dispositivo ha de ser leído o escrito

Los registros de Configuración

00	Device ID		Vendor ID	
01	Status Register		Command Register	
02	Class Code			Revision ID
03	BIST	Header Type	Latency Timer	Cacheline size
04	Base Address Register 0			
05	Base Address Register 1			
06	Base Address Register 2			
07	Base Address Register 3			
08	Base Address Register 4			
09	Base Address Register 5			
10	Cardbus CIS Pointer			
11	Subsystem ID		Subsystem Vendor ID	
12	Expansion Rom Base Address			
13	Reserved			
14	Reserved			
15	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Los registros de Configuración

- Vendor ID (16 bits): registro estático RO que identifica al fabricante mediante un número dado por el PCI SIG. El número 0xFFFF no puede ser usado y corresponde a un dispositivo inexistente
- Device ID (16 bits) y Revision ID (8 bits): registros estáticos RO con la identificación del dispositivo y su versión elegidos por el fabricante
- Header Type (8 bits): para definir si un dispositivo es o no multifunción
- Class Code (24 bits): registro estático RO obligatorio, dividido en tres subregistros de 8 bits: base class, subclass, y programming interface.
 - La clase base puede ser un tipo predefinido como MASS STORAGE CONTROLLER, NETWORK CONTROLLER, DISPLAY CONTROLLER, etc, 0x00 si es un PCI definido antes que se predefinieran los tipos, o 0xFF si no se acomoda a ninguno.
 - La subclase describe al tipo con más detalle. P.Ej: un MASS STORAGE puede ser SCSI, IDE, IPI, RAID, floppy u otro.
 - El tercer registro da detalles de la interfase de manejo. P.Ej: un SIMPLE COMM's CONTROLLER puede ser subclase PUERTA PARALELO e interfase ECP

Los registros de Configuración

Command Register (16 bits): registro R&W donde sólo están definidos 10 bits. Sólo tiene sentido sintetizar aquellos bits que serán usados por el dispositivo

- bit 9: Fast Back-to-Back enable (default=0). Es un bit opcional para MASTERS donde un '1' habilita la realización de transacciones Fast Back-to-Back.
- bit 8: System Error enable (default=0). Un '1' habilita al driver que genera SERR# para generar indicación de errores graves y de paridad.
- bit 7: Wait Cycle enable (default=0). Un '1' habilita el uso de address/data stepping
- bit 6: Parity Error response (default=0). Un '1' habilita la generación de PERR# ante la detección de errores de paridad. Si está en '0' el dispositivo ignora este tipo de errores
- bit 5: VGA palette snoop enable (default=0). Un '1' habilita características del manejo de palette en periféricos tipo VGA

Los registros de Configuración

Command Register (16 bits): registro R&W donde sólo están definidos 10 bits . Sólo tiene sentido sintetizar aquellos bits que serán usados por el dispositivo

- bit 4: Memory Write & Invalidate (MWI) enable (default=0). Un '1' habilita a un MASTER a generar comandos MWI. Si está deshabilitado el MASTER sólo puede usar comandos Memory Write. Debe ser habilitado en sintonía con el registro Cacheline Size según el cache del sistema
- bit 3: Special Cycle recognition (default=0). Un '1' habilita al dispositivo a decodificar mensajes de broadcast tipo Special Cycle
- bit 2: Master enable (default=0). Un '1' habilita al dispositivo a actuar como MASTER (si es que tiene capacidad para hacerlo)
- bit 1: Memory access enable (default=0). Un '1' habilita al dispositivo a reaccionar a accesos de memoria por el bus PCI
- bit 0: I/O accesss enable (default=0). Un '1' habilita al dispositivo a reaccionar a accesos de I/O por el bus PCI

Los registros de Configuración

Status Register (16 bits): registro Read & Write donde sólo es posible escribir '0's pero no '1's. Sólo tiene sentido sintetizar aquellos bits que serán usados por el dispositivo

- bit 15 - Detected Parity Error: indica que el dispositivo ha detectado un error de paridad (por más que no active PERR#)
- bit 14 - Signaled System Error: debe setearse cada vez que el dispositivo genera SERR#. Si no es capaz de hacerlo debe estar siempre en '0'.
- bit 13 - Received Master Abort. Requerido en los MASTER debe ir a '1' cada vez que el MASTER aborta una transacción por Master Abort
- bit 12 - Received Target Abort. Requerido en los MASTERS, debe ir a '1' cada vez que un TARGET aborta una transacción.
- bit 11 - Signaled Target Abort. Requerido en los TARGET capaces de abortar una transacción para indicar cuándo realizan esa acción.
- bit 10..9 - Device Select Timing: bits read only, indican cuál es el máximo tiempo en que un TARGET se identifica con DEVSEL# (0,1 o 2 ciclos)

Los registros de Configuración

Status Register (16 bits): registro Read & Write donde sólo es posible escribir '0's pero no '1's. Sólo tiene sentido sintetizar aquellos bits que serán usados por el dispositivo

- bit 8 - Data Parity Reported: Requerido sólo en los MASTER, se setea en una transacción si ese MASTER fué quien la inició, y él generó PERR# en un read o su TARGET en un write, y el bit PARITY ERROR RESPONSE del Command register está en '1'.
- bit 7 - Fast Back-to-Back Capable: Read only, un '1' indica que el TARGET soporta transacciones Fast Back-to-Back
- bit 6 - UDF Support: Fijo. Marca si el dispositivo acepta User Definable Features
- bit 5 - 66MHz capable: Fijo, un '1' marca que el dispositivo es capaz de operar con un reloj de 66 MHz.
- bit 4..0: Reserved (0)

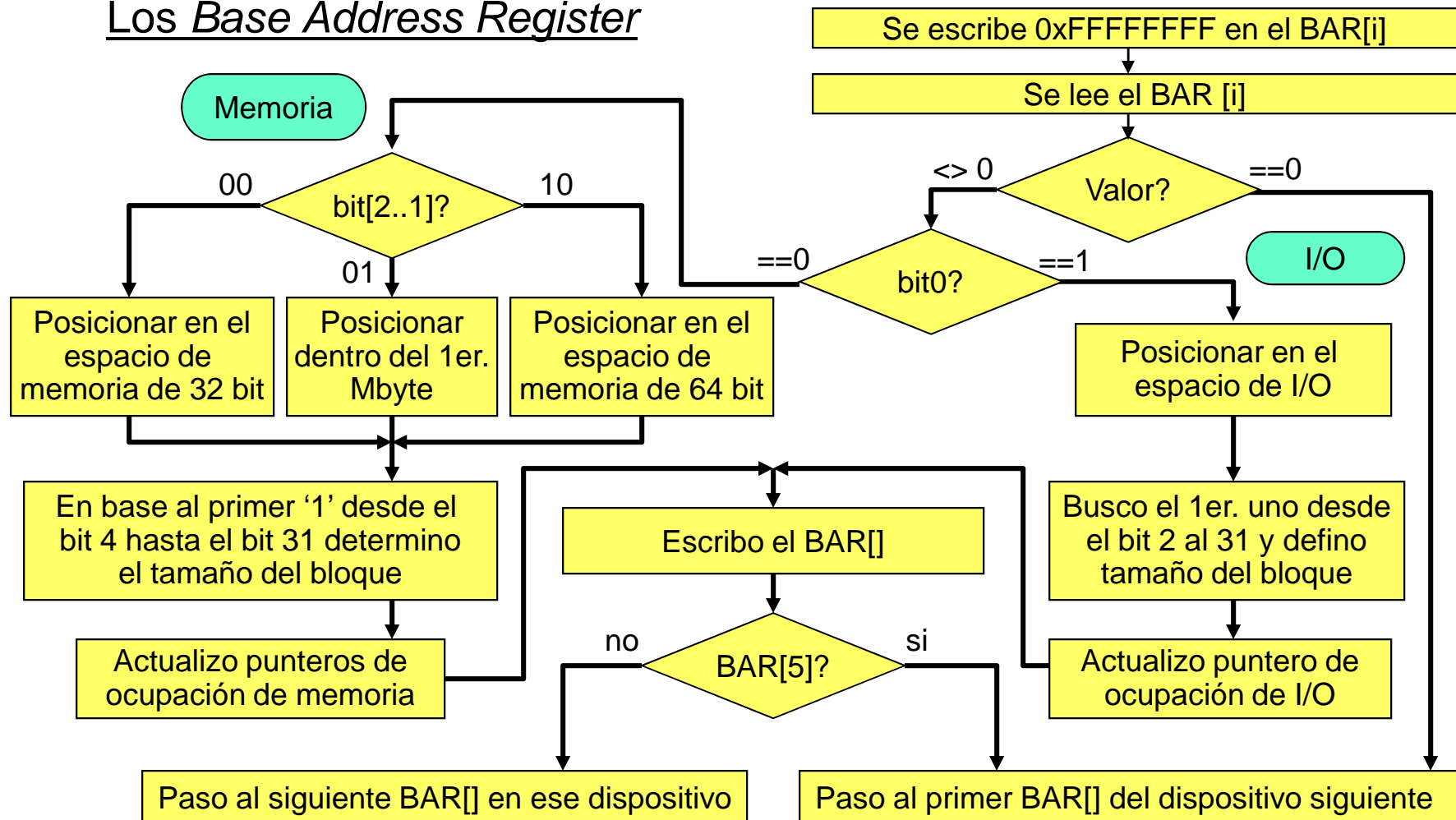
Los registros de Configuración

Los registros *Base Address Register (BARs)*

- Cada interfase PCI puede asociarse a hasta 8 dispositivos (8 bloques BAR), cada uno de ellos a hasta 6 zonas en el espacio de I/O y Memoria, cada una de ellas ocupando un bloque de direcciones en ese espacio
- En PCI es el Host quien cumple la tarea de asignar posiciones físicas a cada dispositivo, de modo incremental, al inicializar el sistema, lo que permite la relocalización física de cada periférico
- Para poder inicializar cada dispositivo el HOST debe resolver:
 - cuántos de los bloques de datos están asociados a ese dispositivo
 - a qué espacio de datos corresponde cada uno
 - y qué tamaño tiene cada bloque
- Para resolver estas preguntas, en el banco de registros de un dispositivo PCI existen 6 posibles registros de 32 bits denominados BAR[5..0]

Los registros de Configuración

Los Base Address Register



Los registros de Configuración

- Latency Timer: registro RW obligatorio en los MASTER que soportan ciclos de escritura burst con más de 2 fases de datos. Puede ser fijo si soportan ciclos de escritura de 2 o menos fases de datos, pero nunca mayor a 16.
- Cacheline size: obligatorio para MASTERS que soportan Memory Write & Invalidate y para Memory Targets que soportan direccionamiento *cacheline wrap* ($AD[1..0] == 10$).
- BIST Built-in Self test, opcional. Debe valer 0x00 si no es usado.
- Interrupt Pin: numero de pin de interrupción ($INTA\# = 0$ a $INTD\# = 3$) al que excita el dispositivo
- Interrupt Line: número REAL de interrupción asociado a la interrupción que genere ese dispositivo. En el caso de PCs tipo x86 estan definidos los valores de 0 a 15. El valor 0xFF indica “no connection”
- Otros: Max_Lat, Min_Gnt, Cardbus CIS Pointer, Subsystem ID y Subsystem Vendor ID, Expansion Rom Base Address

Exigencias temporales en PCI

- La elevada velocidad de operación del bus PCI obliga a cumplir exigencias temporales estrictas, así como de carga eléctrica y capacidad de las entradas
- Entre flanco y flanco de la señal CLK deben suceder varios eventos:
 - transición de CLK y su propagación por el bus
 - resolución interna, y propagación desde la interfase al conector del bus de los datos de salida
 - propagación de las señales a través del bus por *reflected wave switching* hacia todas las interfases
 - estabilización de esos datos con el tiempo de “setup” necesario antes de la próxima transición de CLK

Exigencias temporales en PCI

- capacidad de entrada: hasta 10 pF/línea
- carga eléctrica: no pullups y 1 carga por interfase
- tiempos:

- transición de CLK y su propagación (**skew**): **2 ns**

- estabilización de datos de salida

- **tco** de señales tipo bus: entre 2 y **11 ns**

- salida de tri-state: hasta 2 ns

- pasaje a tri-state: hasta 28ns

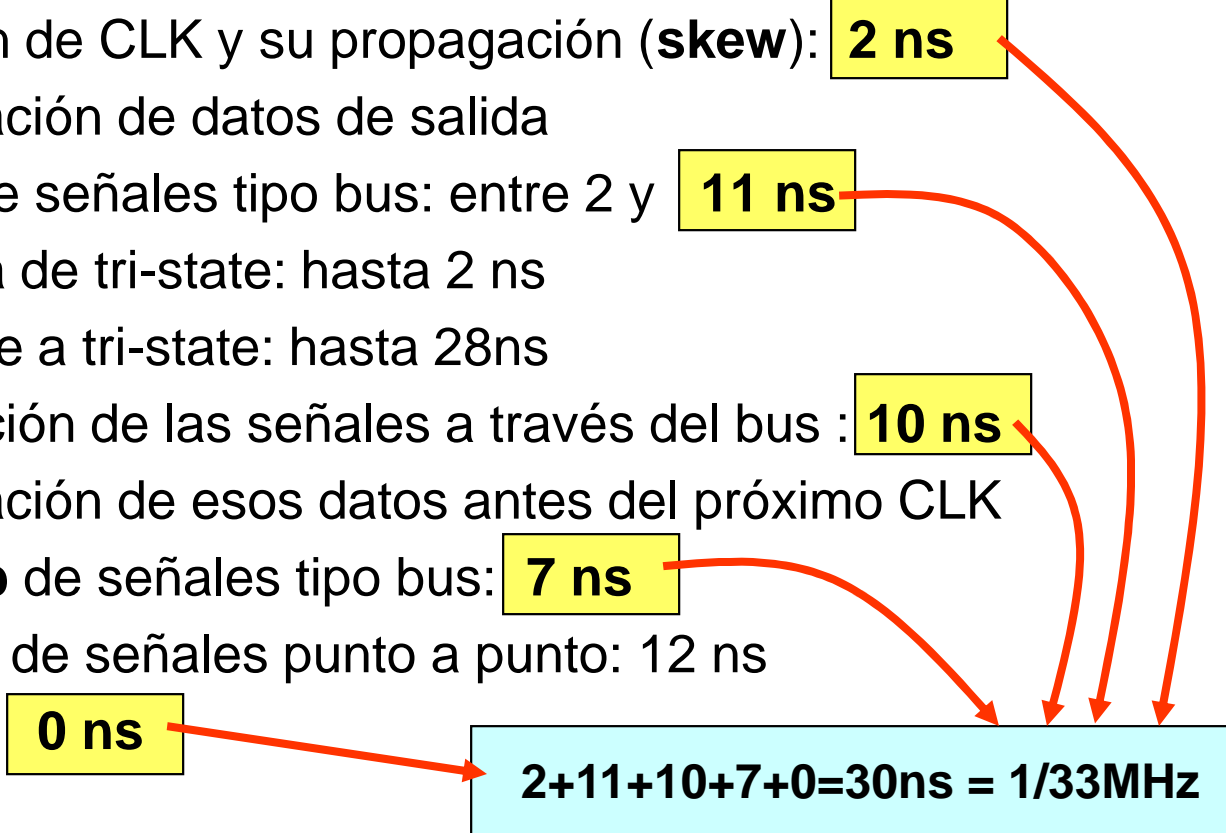
- propagación de las señales a través del bus : **10 ns**

- estabilización de esos datos antes del próximo CLK

- **setup** de señales tipo bus: **7 ns**

- setup de señales punto a punto: 12 ns

- **hold**: **0 ns**


$$2+11+10+7+0=30\text{ns} = 1/33\text{MHz}$$

Un procesador real para SoC: el caso ARM

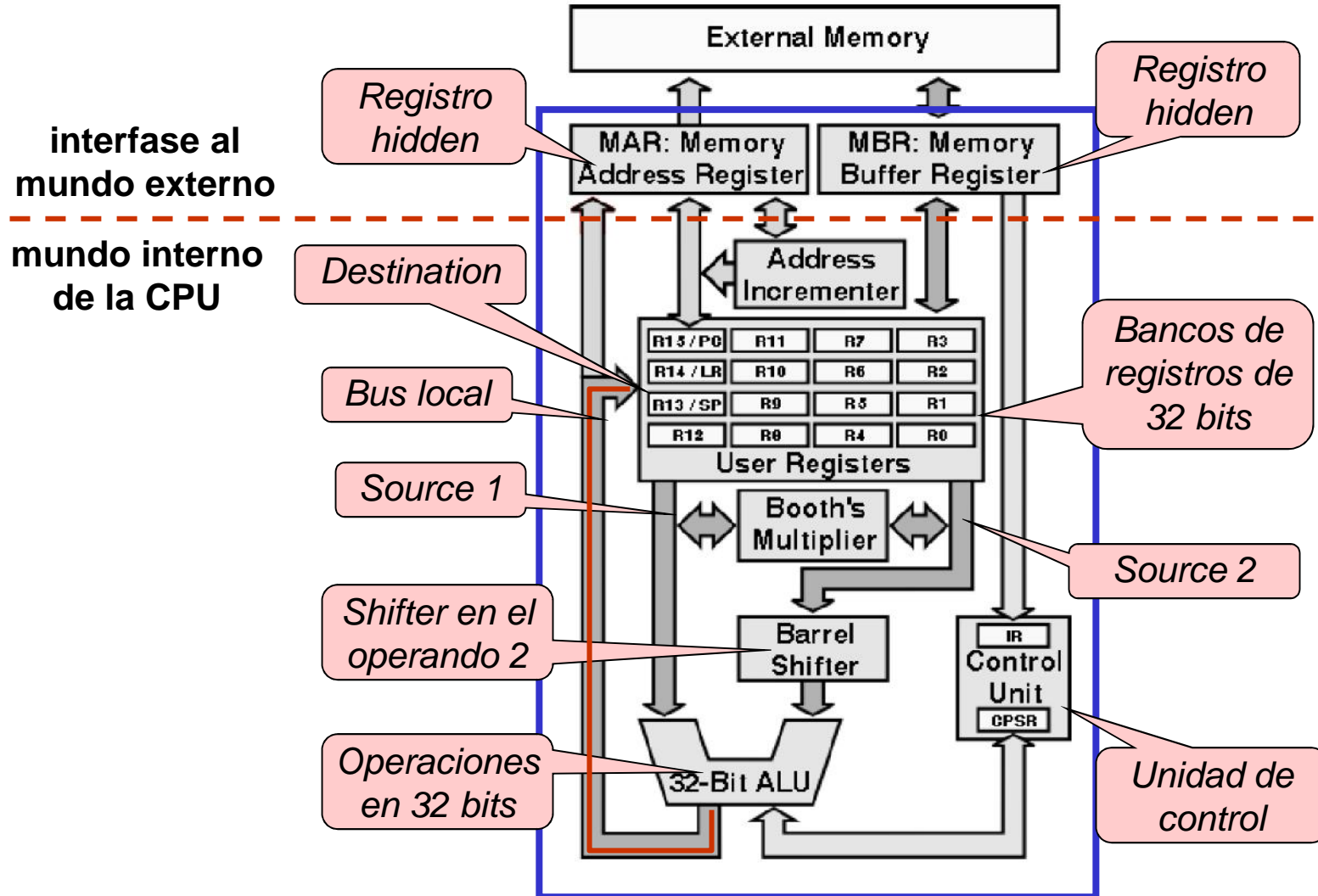
ARM corresponde a una arquitectura RISC de 32-bit, siendo la más usada en la actualidad, y su simplicidad la hace conveniente para aplicaciones de “sistemas en un chip” (*System On Chip*, o **SoC**) de bajo consumo de potencia. Por eso es predominante en sistemas móviles como PDAs, teléfonos celulares, reproductores de video o música, o periféricos inteligentes.

Los **ARM** poseen distintas variaciones de una arquitectura básica similar, incluyendo los **ARM7**, **ARM9**, **ARM11**, y desde hace poco los **Cortex**. La arquitectura ARM no es propia de un fabricante de chips, sino de una empresa que diseña IP, puede ser usada con licencia, y por eso existen procesadores ARM/Cortex fabricados bajo licencia de ARM por empresas como Atmel, Freescale, NEC, NVIDIA, NXP, Samsung, Sharp, ST Microelectronics, Texas Instruments, y otros.

Características básicas de las arquitecturas ARM

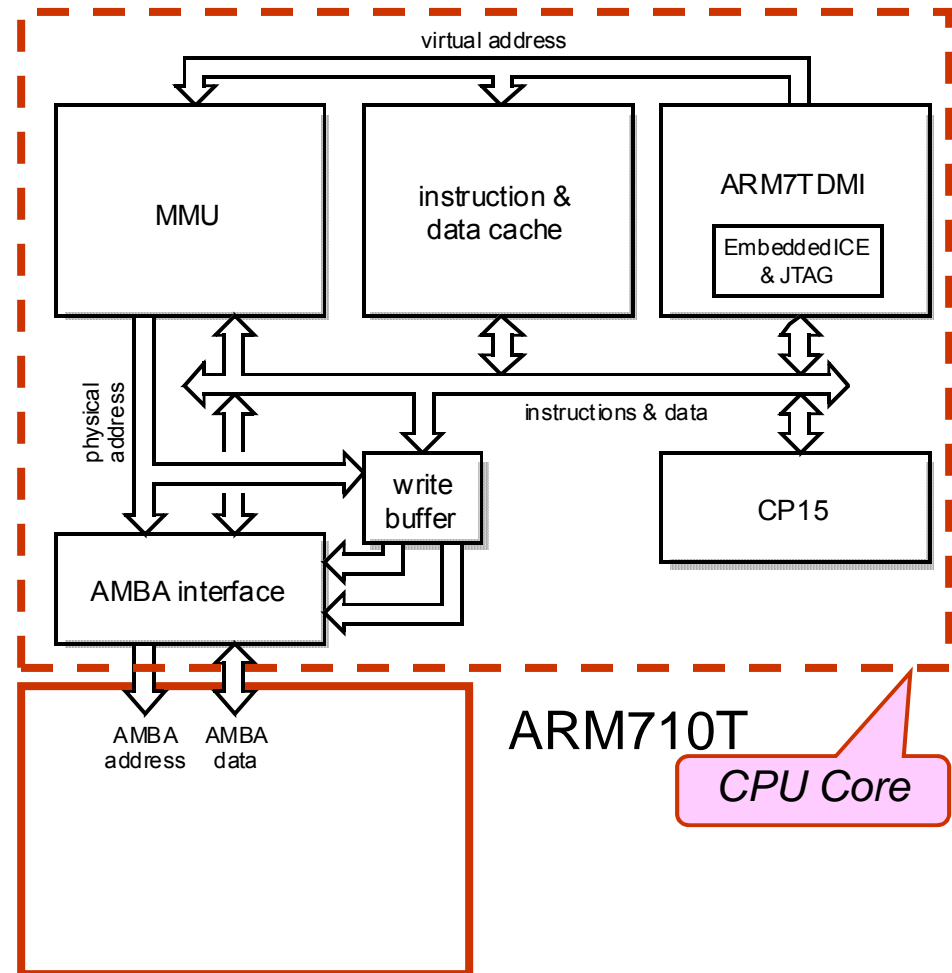
- Diseñados para aplicaciones pequeñas de alta performance y bajo consumo (es decir, NO para un centro de cómputo)
- RISC 32 bits
- Control simultáneo de la ALU y el shifter en cada instrucción de procesamiento de datos
- Juego de Registros con funcionalidades uniformes
- Arquitectura Load/Store (las operaciones sólo actúan sobre registros, en memoria sólo se lee y escribe)
- Modos de direccionamiento simple, de 3 hasta direcciones (*source1, source2, destination*)
- Capacidad de auto incremento y auto decremento
- Load y store de múltiples datos mediante una única instrucción (*block moves*).

Arquitectura del *Processor Core* de los ARMv4



Qué es AMBA?

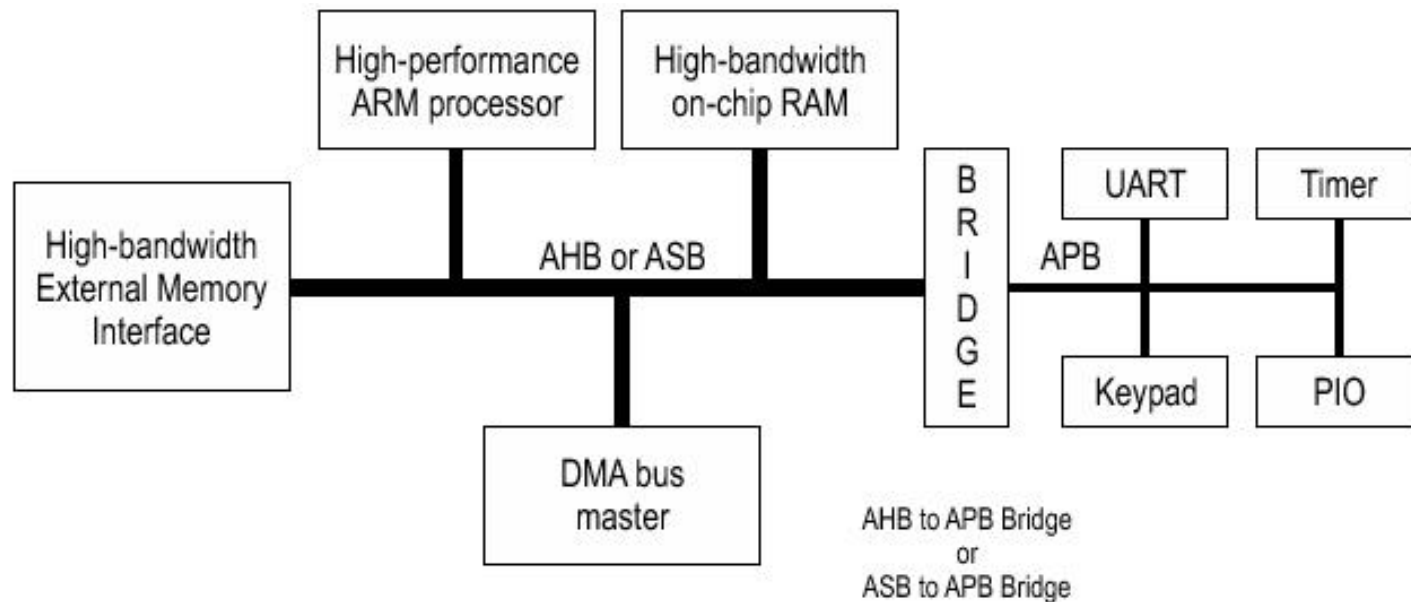
- La CPU Core ARM se conecta con la memoria y los periféricos mediante una interfase estandarizada
- ARM llamó a esta interfase AMBA, por Advanced Microcontroller Bus Architecture
- Esta característica permite que los distintos fabricantes de procesadores basados en ARM puedan definir y ofrecer periféricos propios
- La interfase AMBA define varios métodos posibles



Qué es AMBA?

- Advanced Microcontroller Bus Architecture es una norma de comunicaciones on-chip, independiente de la tecnología que se usa para implementarla, pues sólo define el protocolo del bus
- Originalmente definía tres diferentes tipos de bus: AHB (Advanced High-performance Bus), ASB (Advanced System Bus), y APB (Advanced Peripheral Bus).
- Pero en la versión 3 aparece AXI (Advanced)
- Con la aparición del microprocesador Cortex (ARMv7) se definió una versión del AHB llamada AHB Lite, para reducir la complejidad y costo
- Existen distintas generaciones de especificaciones AMBA, que han surgido con el avance tecnológico y la ampliación del tipo de aplicaciones de los procesadores ARM en sus distintas variantes
- Es una norma en permanente expansión y evolución

Sistema típico usando la versión inicial de AMBA



AMBA AHB

- * High performance
- * Pipelined operation
- * Multiple bus masters
- * Burst transfers
- * Split transactions

AMBA ASB

- * High performance
- * Pipelined operation
- * Multiple bus masters

AMBA APB

- * Low power
- * Latched address and control
- * Simple interface
- * Suitable for many peripherals

La especificación AMBA 2 define dos protocolos de interfase:

AMBA 2 AHB Interface

- La interfase AMBA 2 AHB está orientada a facilitar la interconexión eficiente entre masters en un sistema con reloj común, tolerando datos de hasta 128 bits.

AMBA 2 APB Interface

- La interfase AMBA 2 APB está orientada al manejo de transacciones de bajo ancho de banda y elevada latencia, tal como sucede con registros de configuración y periféricos. Es una interfase compacta y de bajo consumo de energía que opera en forma independiente del AMBA 2 AHB, con quien se conecta a través de un bridge.

AMBA 2 AHB Lite:

AMBA 3

AMBA 3 agrega 4 protocolos para cubrir requerimientos de procesamiento intensivo, canales de baja velocidad, y también de test y debug.

- **AMBA 3 AXI Interface:** Para manejar elevado tráfico de datos a alta velocidad, a través de hasta 5 canales de transacciones con temporización variable entre ellas, y transacciones fuera de orden. El sistema utiliza pipeline, soporta transacciones READ y WRITE simultáneas, así como dispositivos de elevada latencia inicial
- **AMBA 3 AHB Interface:** permite la interconexión muy eficiente en un simple de simple frecuencia. Posee canales unidireccionales con pipeline, compatible con AMBA 2 AHB-Lite.
- **AMBA 3 APB Interface:** para transacciones de bajo ancho de banda, compatible con AMBA 2 APB.
- **AMBA 3 ATB Interface:** define el mecanismo para realizar el trace de actividades de la CPU en forma no intrusiva, para tareas de debug

AMBA 4 agrega 3 nuevos protocolos por sobre los de AMBA 3

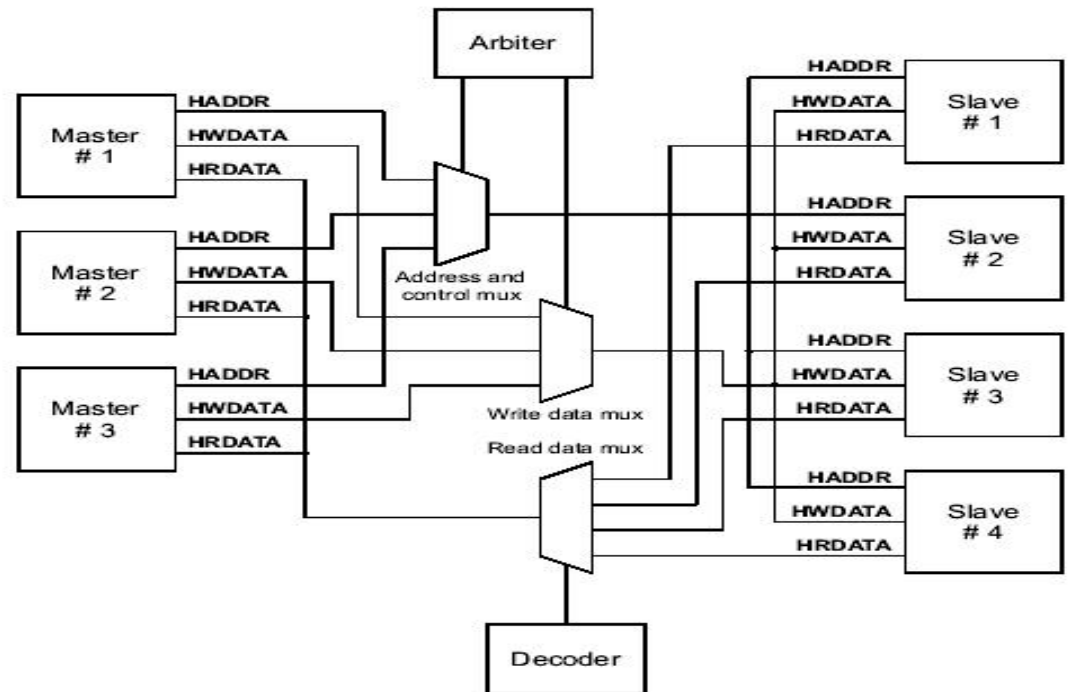
- **AXI4:** es una mejora de AXI3 para optimizar el aprovechamiento de sistemas multimaster. Soporta transferencias burst de mayor longitud, y administración de QoS (Quality of Service), entre otros
- **AXI4-Lite:** es un subset de AXI4, orientado al manejo de registros simples, con bursts de longitud unitaria, de tamaño fijo, y sin capacidad de control de acceso exclusivo (lock)
- **AXI4-Stream:** ideal para implementación en FPGA, está orientado a transferencias unidireccionales de datos con reducido uso de recursos de cableado. Estos recursos pueden ser compartidos para soportar más de un stream de datos sobre el mismo medio físico, con diferentes tamaños de datos

Componentes de un bus AHB

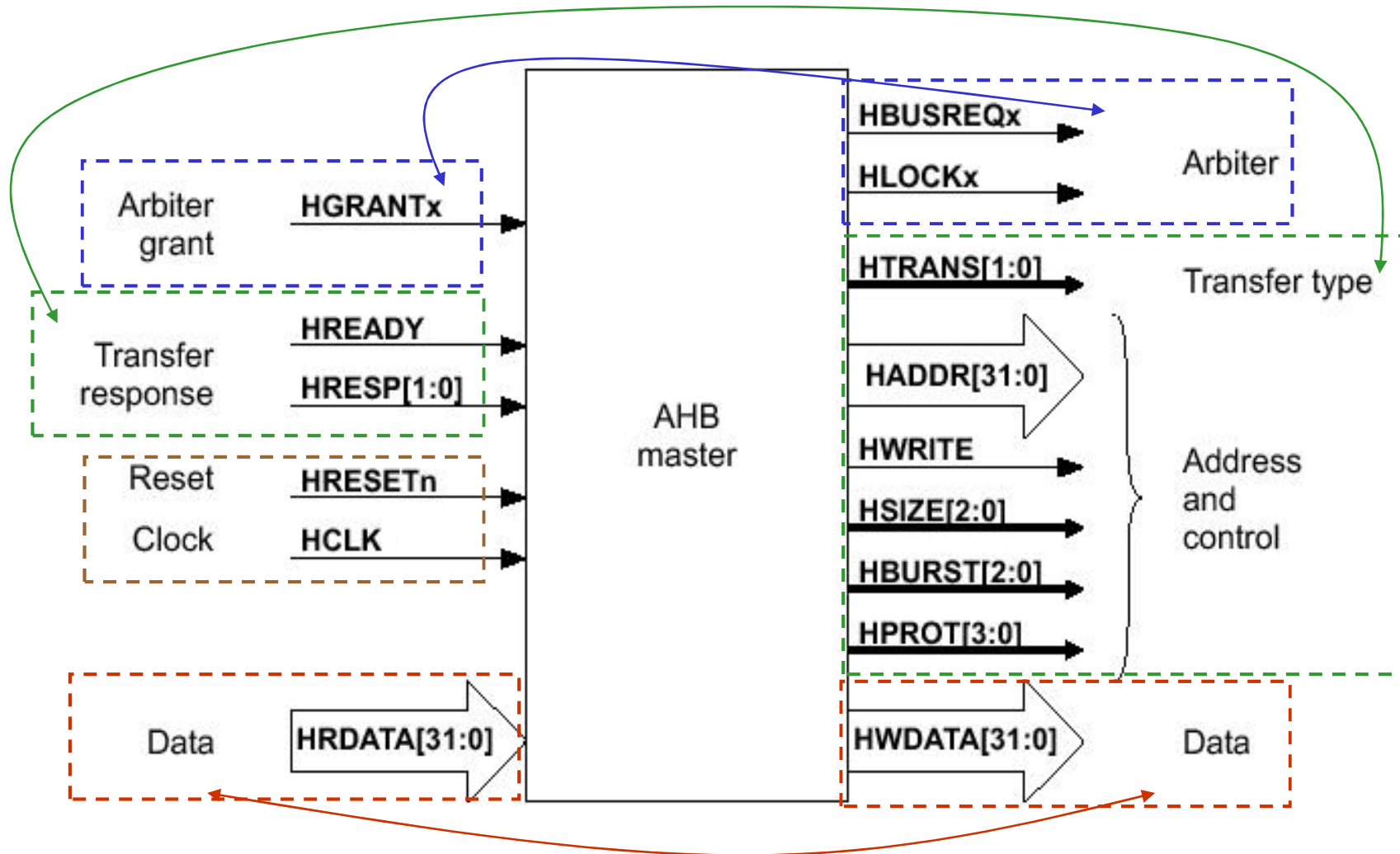
- AHB es un bus de alta performance, con capacidad multimaster (hasta 16), que puede realizar transferencias burst o por sub-bloques (Split transfers).
- Su implementación es mediante canales unidireccionales (no tri-state).
- Los agentes visibles en un bus AHB son:
 - AHB master: es el único capaz de iniciar una transacción read/write, y sólo un master puede estar activo a la vez.
 - AHB slave: es quien responde a la transacción read/write, y es identificado mediante su dirección (Address mapping)
 - AHB arbiter: determina cuál master es el que está tiene el derecho (grant) de uso del bus, según un mecanismo de arbitraje no definido en la norma.
 - AHB decoder: identifica y selecciona a los esclavos en una transacción.

Topología y señales básicas AHB

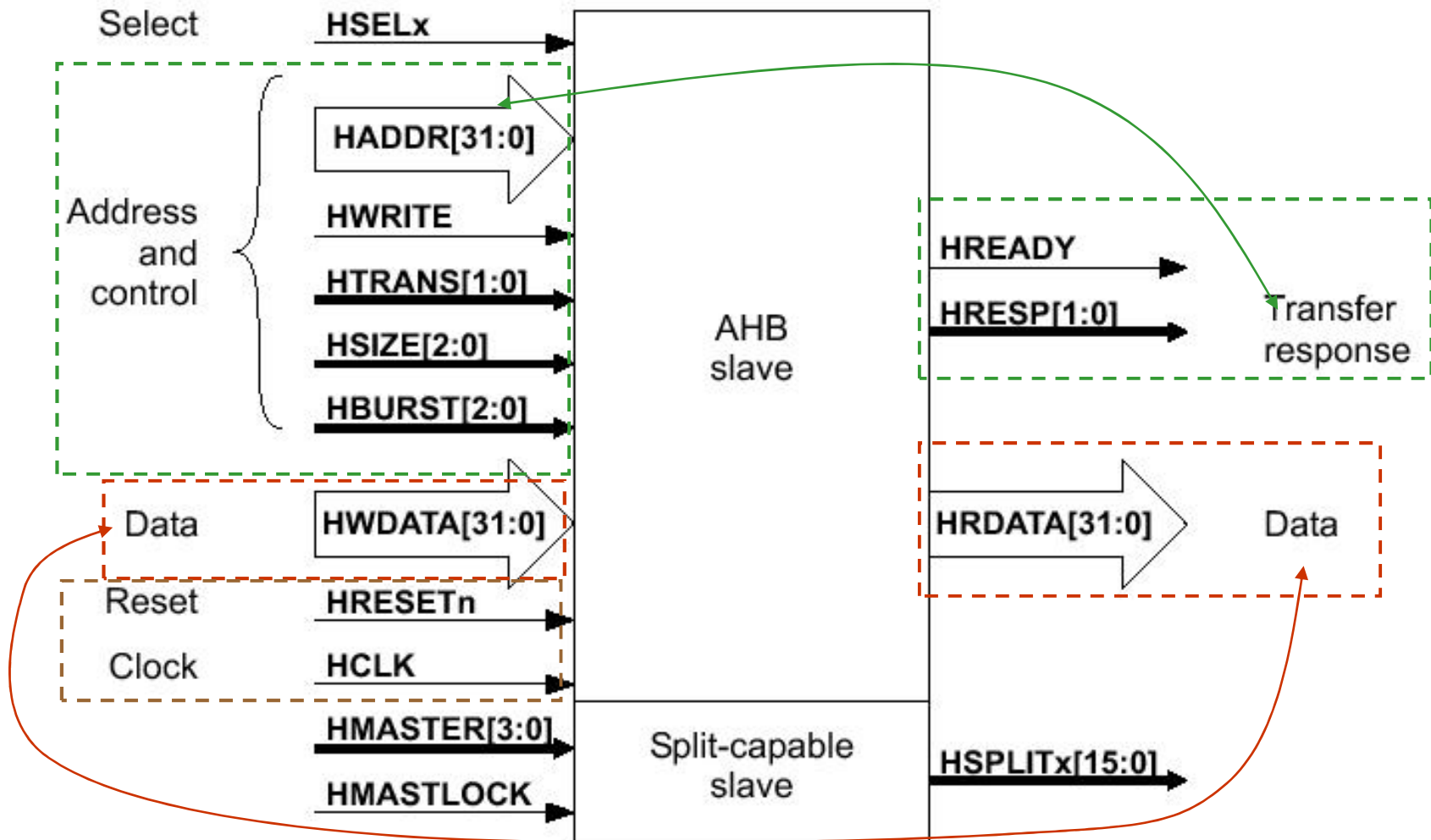
- HRESETn: Activa baja
- HADDR[31:0]: Bus de direcciones de 32-bits
- HWDATA[31:0]: Bus de escritura de datos, *from master to slave*
- HRDATA[31:0]: Bus de lectura de datos, *from slave to master*
- HREADY: Validación de transferencia, del esclavo
- HWRITE: Alto => write, Bajo => read
- HSIZE[2:0]: Tamaño de la transferencia



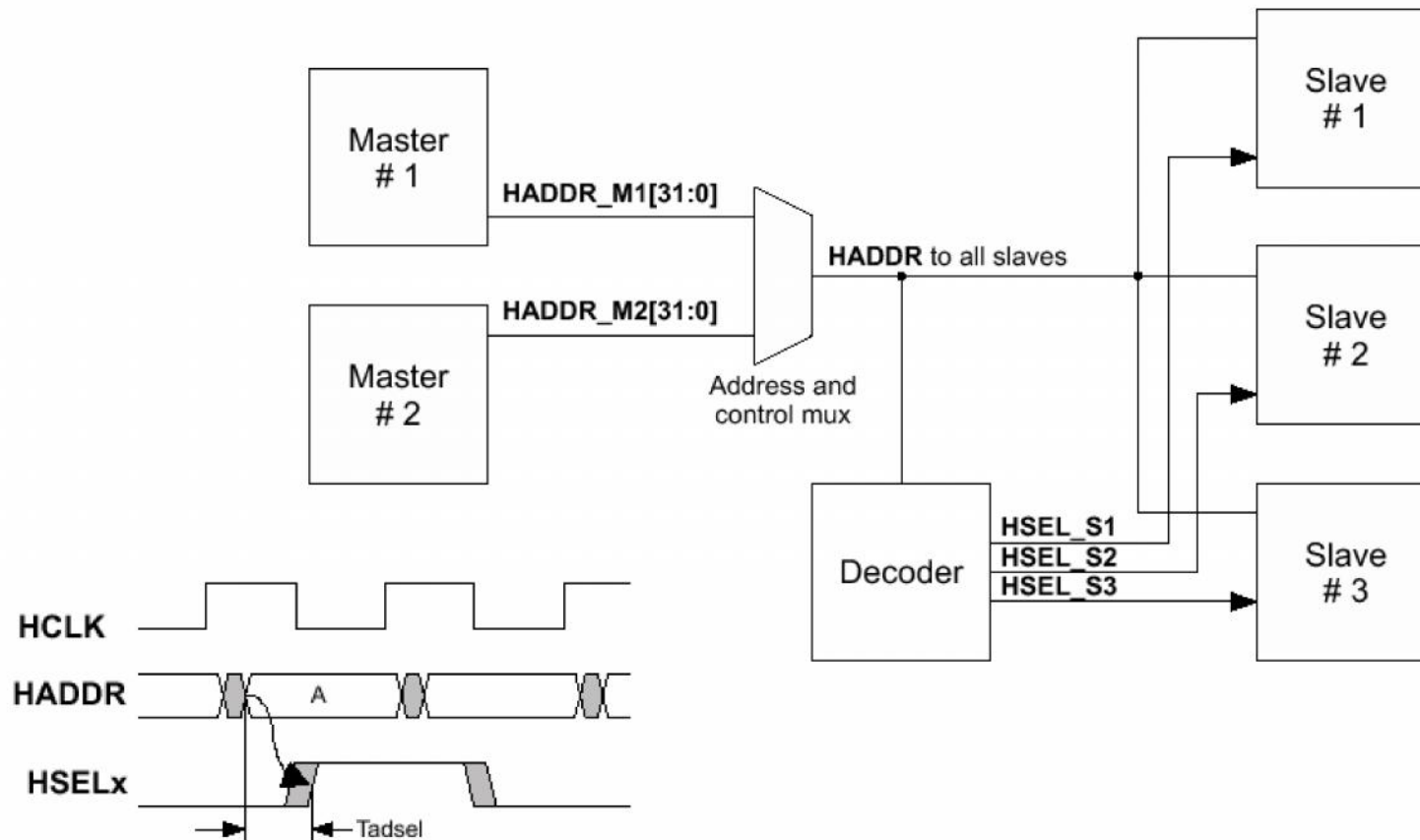
Esquema del Master AHB



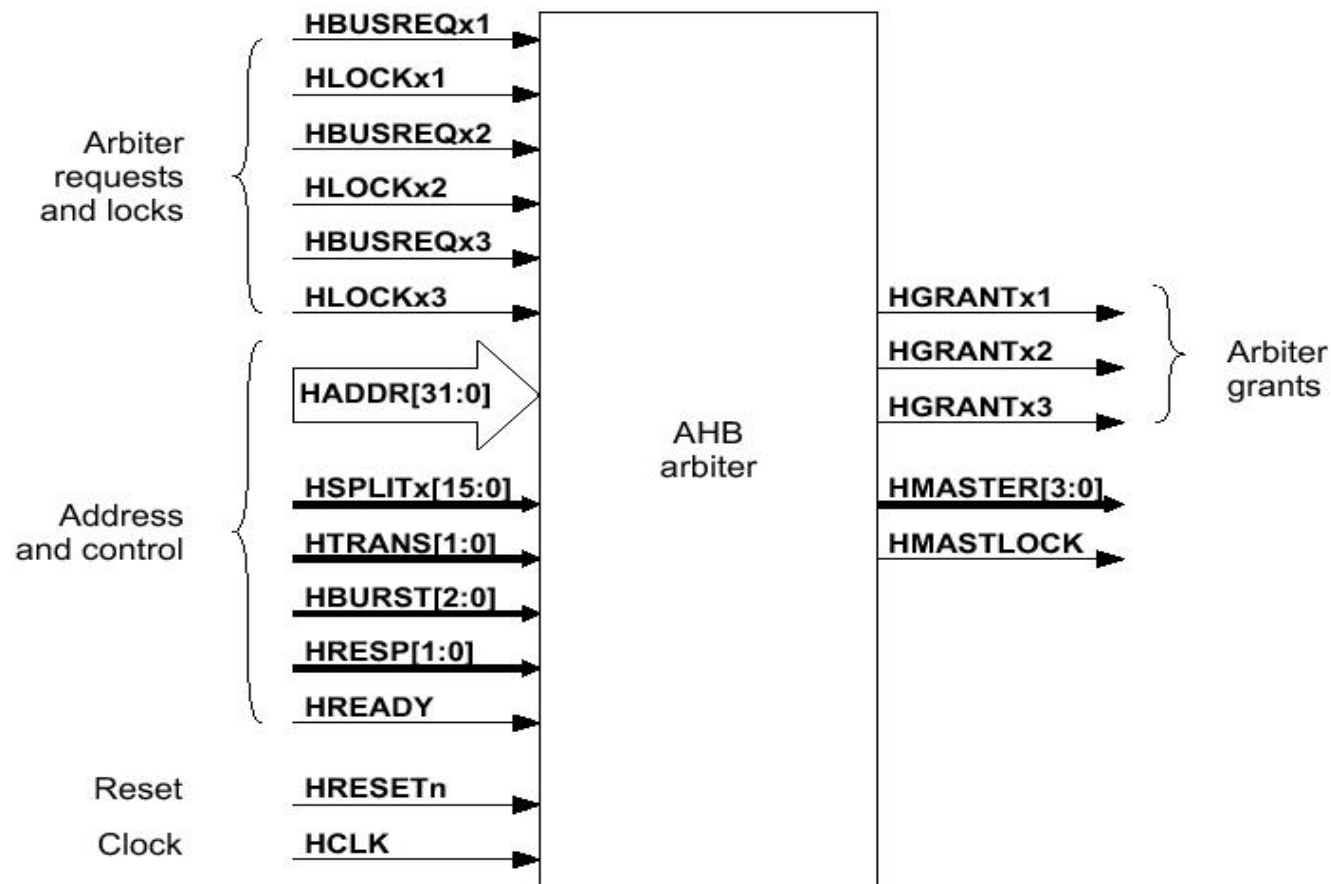
Esquema del AHB Slave



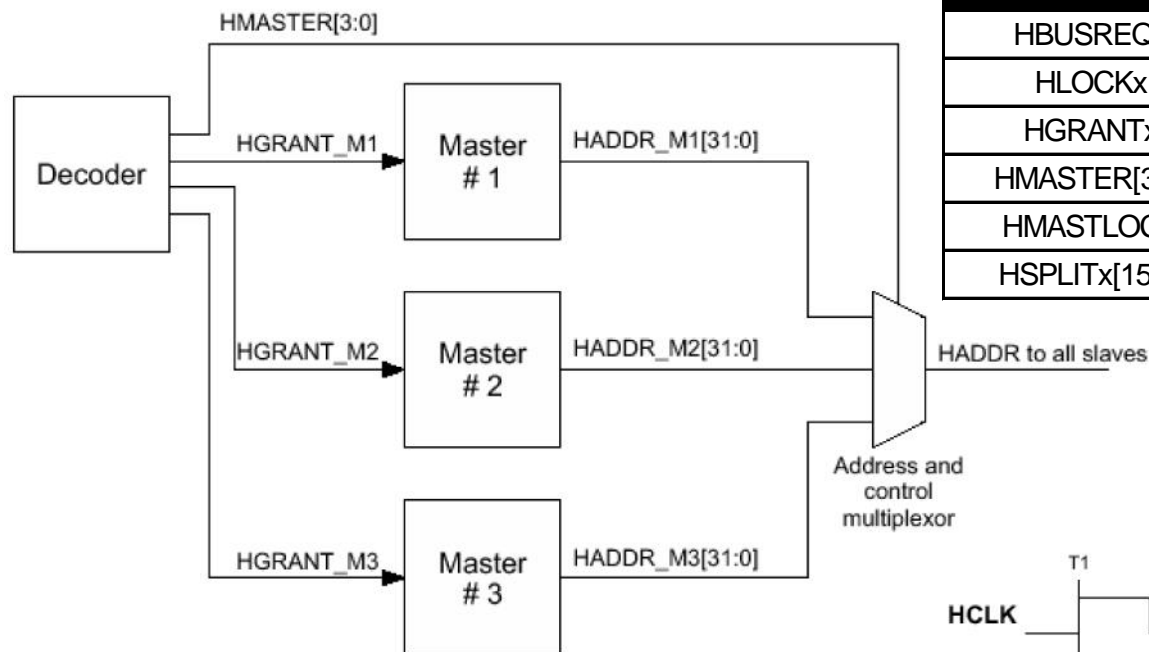
Decodificador de esclavos AHB



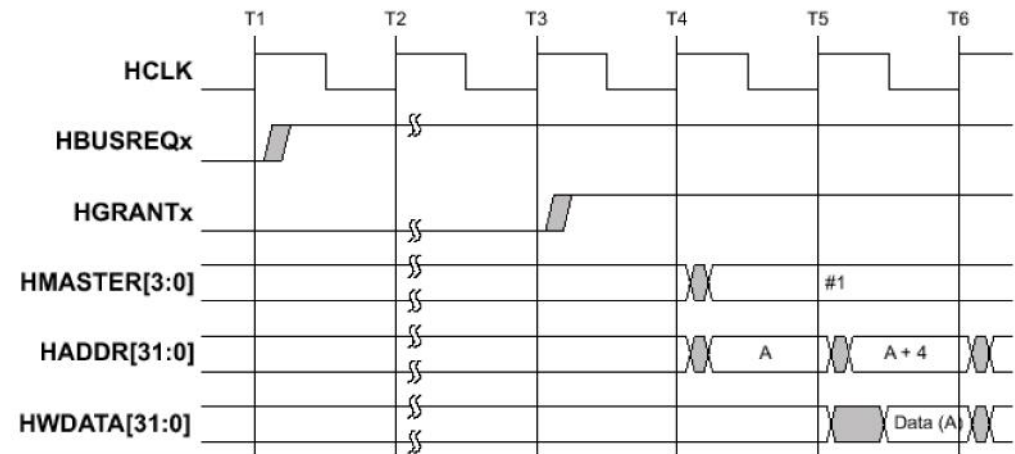
Árbitro AHB



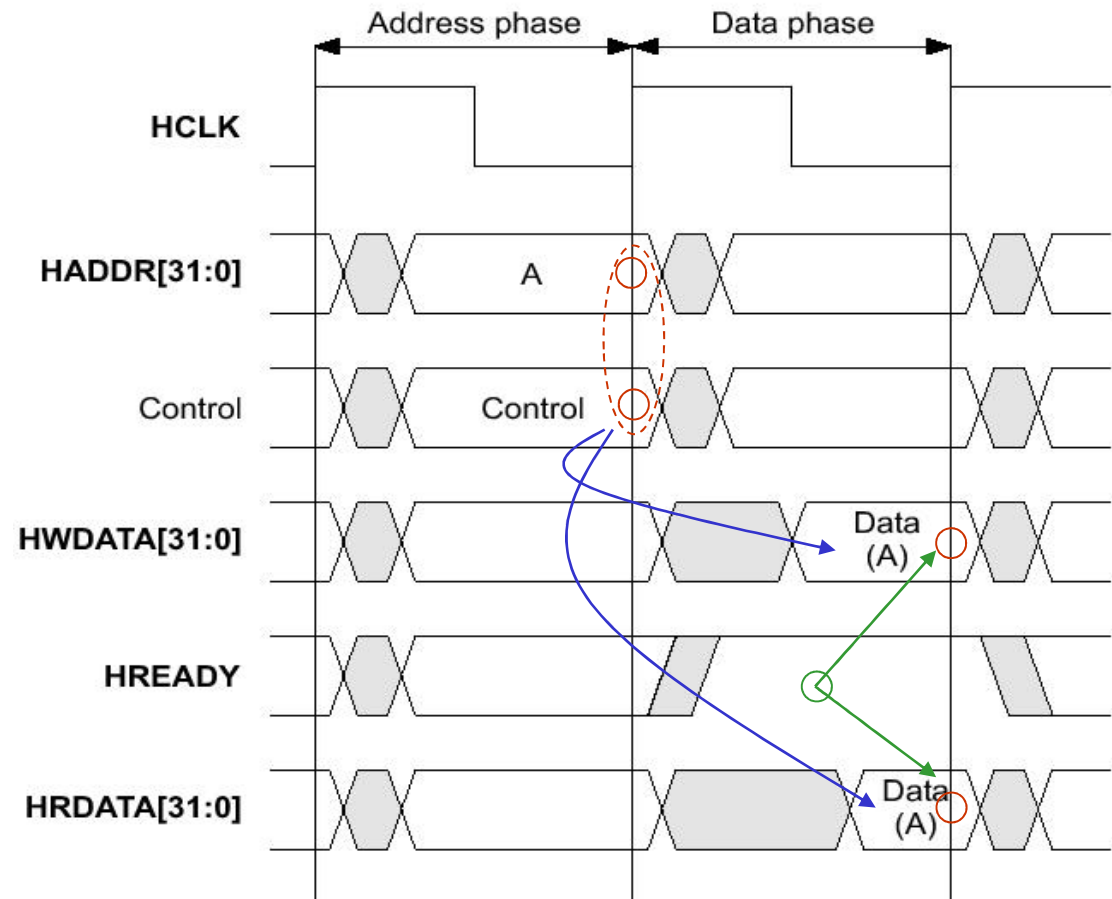
Esquema del arbitraje



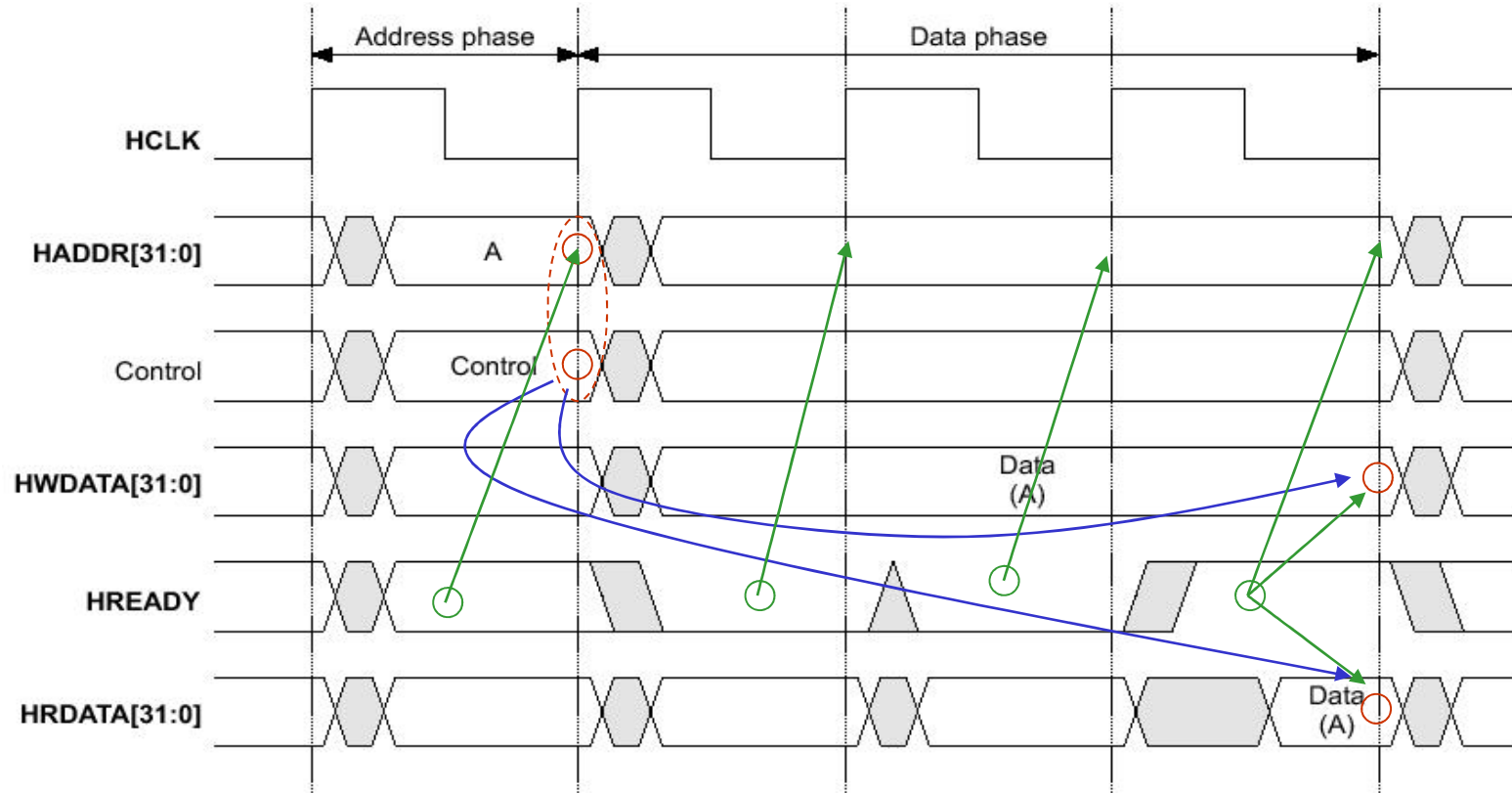
Name		source
HBUSREQx	bus request	master
HLOCKx	locked transfer	master
HGRANTx	bus grant	arbiter
HMASTER[3:0]	master number	arbiter
HMASTLOCK	locked sequence	arbiter
HSPLITx[15:0]	split completion request	slave



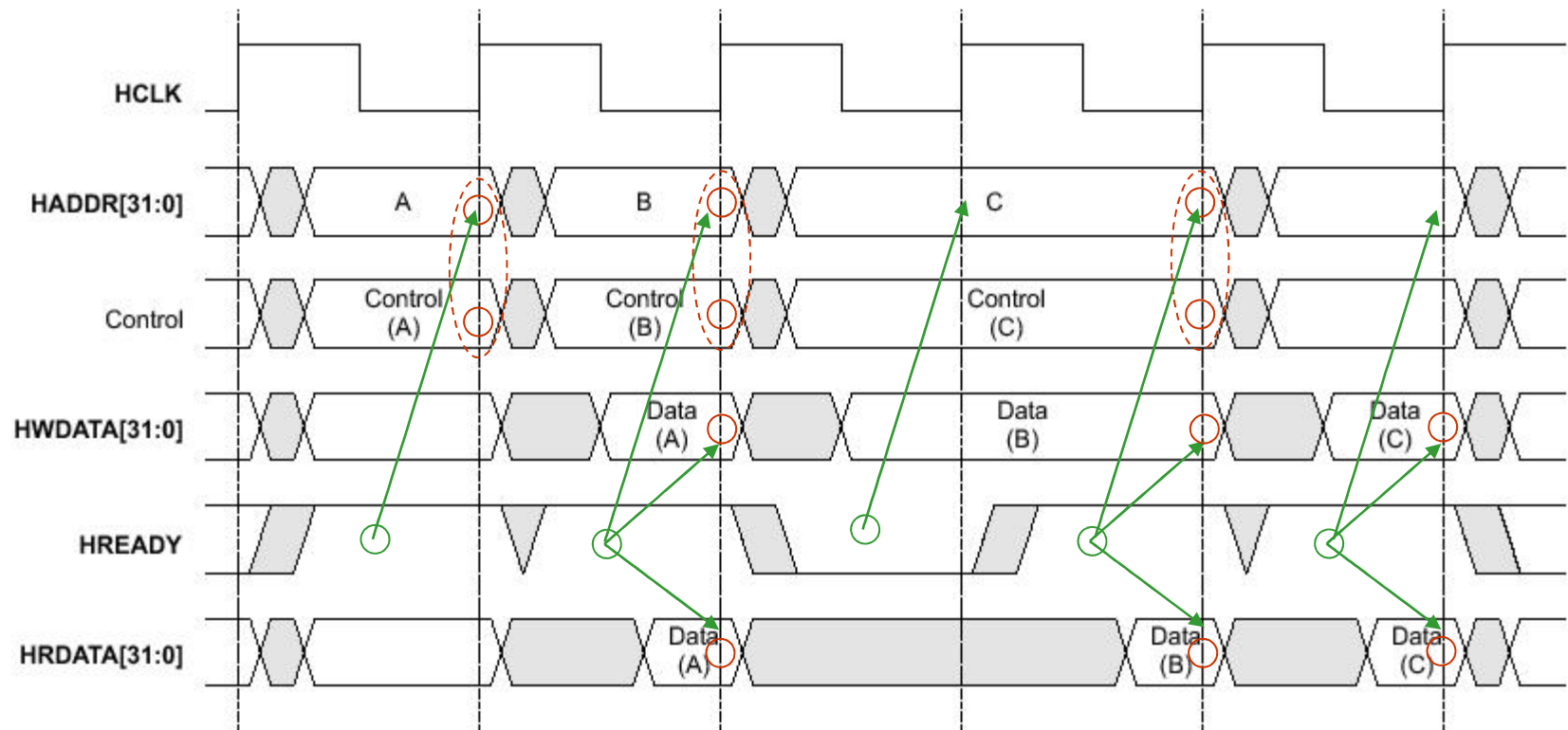
Ejemplo de transferencia simple AHB



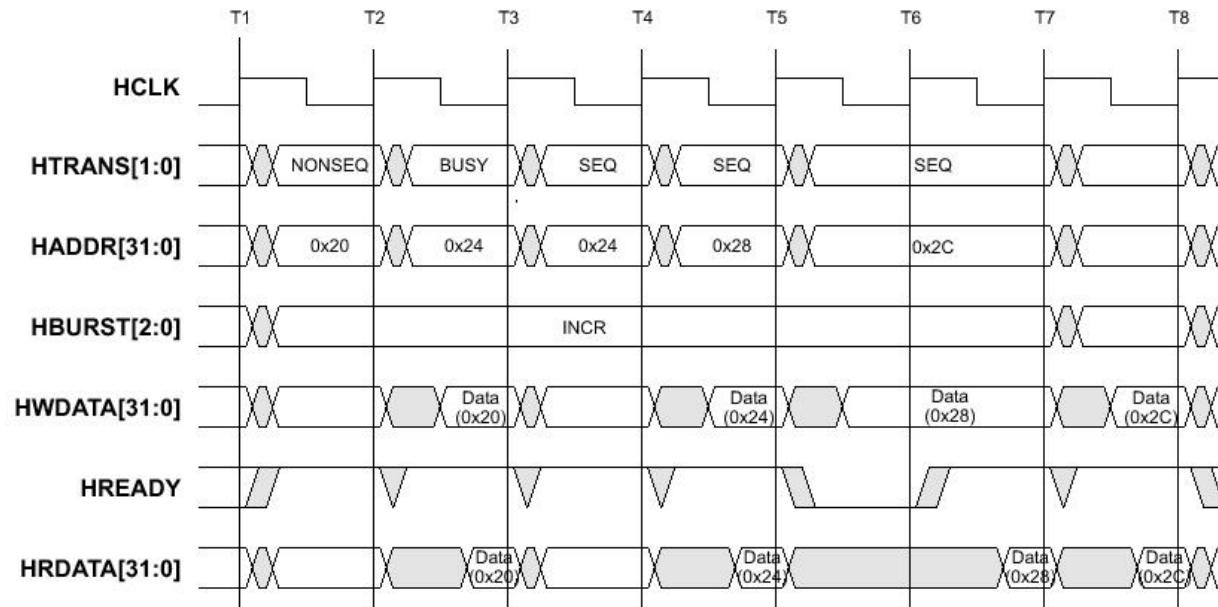
Ejemplo de transferencia simple AHB, con wait



Pipeline de transferencias múltiples AHB



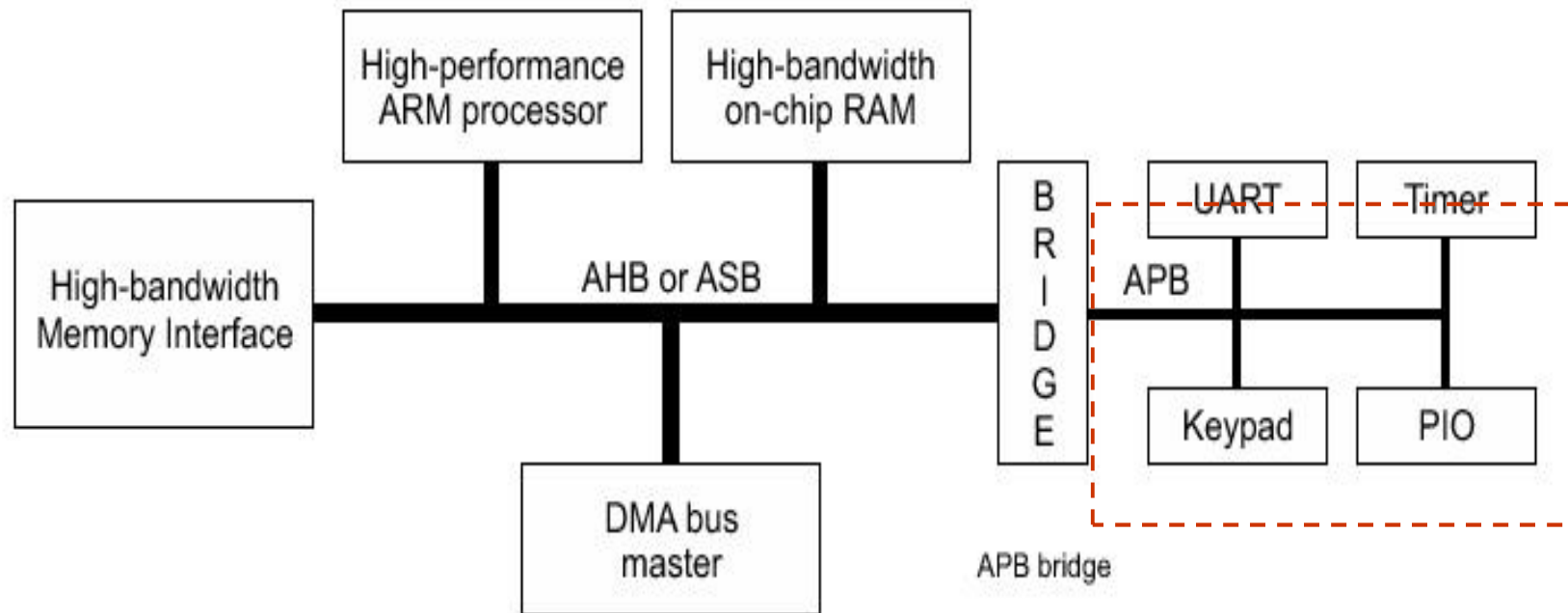
Transferencias AHB, ejemplos



HTRANS[1:0]: identifica al tipo de transferencia

- 00 : IDLE: Sin transferencia, aunque sea con posesión del bus (grant)
- 01 : BUSY: Cuando el master inserta ciclos muertos durante una transferencia
- 10 : NONSEQ: Indica una transferencia simple o la primer transferencia de una ráfaga (burst), donde el valor de las señales de direcciones y control no está vinculado a la transferencia previa
- 11 : SEQ: Indica las sucesivas transferencias en una ráfaga, donde las direcciones están relacionadas con la transferencia previa

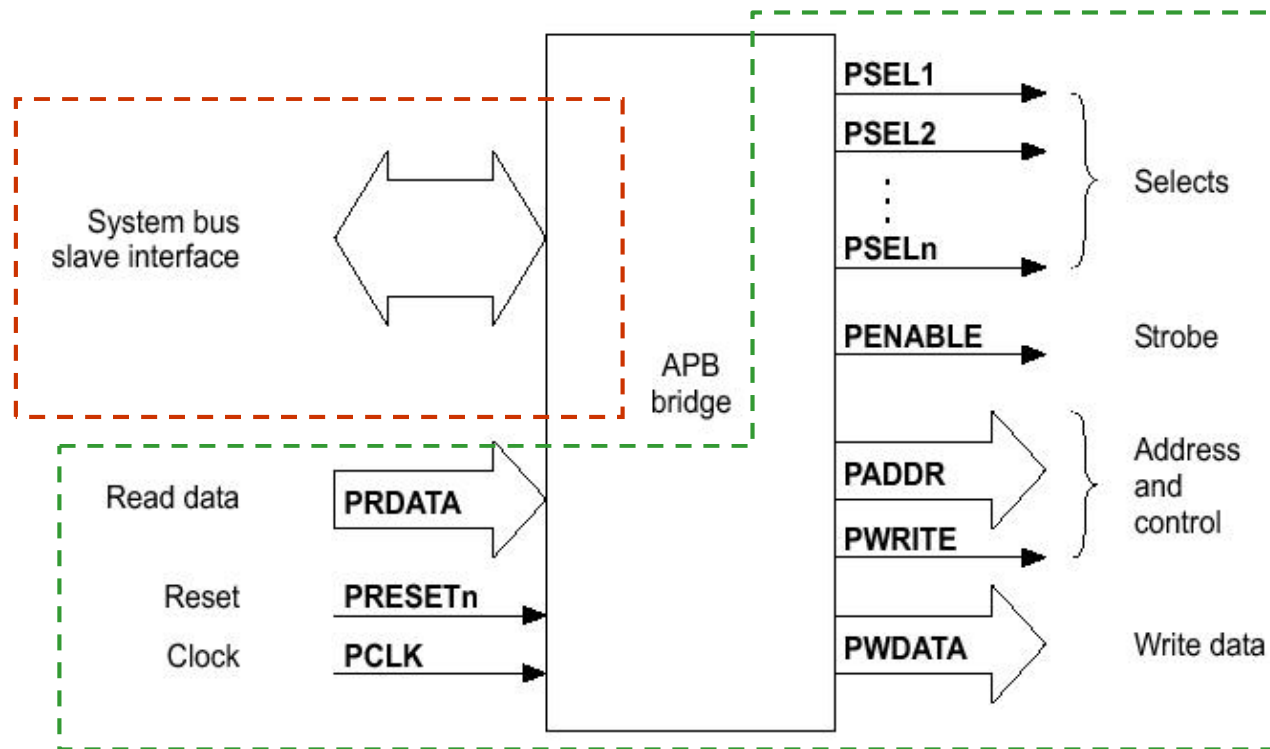
AMBA APB



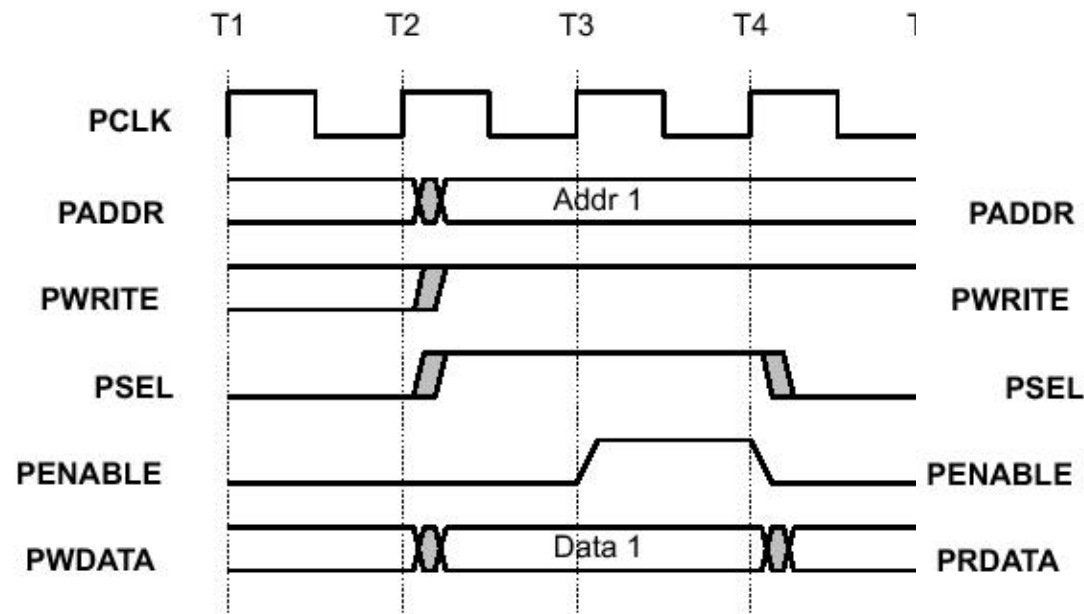
AMBA Advanced Peripheral Bus (APB)

- * Low power
- * Latched address and control
- * Simple interface
- * Suitable for many peripherals

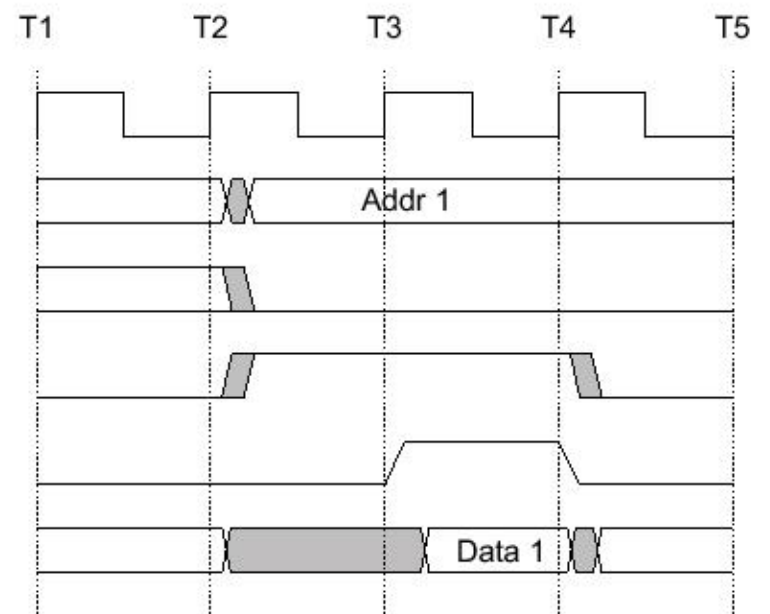
APB bridge: interfase AHB/ASB a APB



Transferencias APB



Write transfer



Read transfer

Bus serial de alta performance: PCI Express

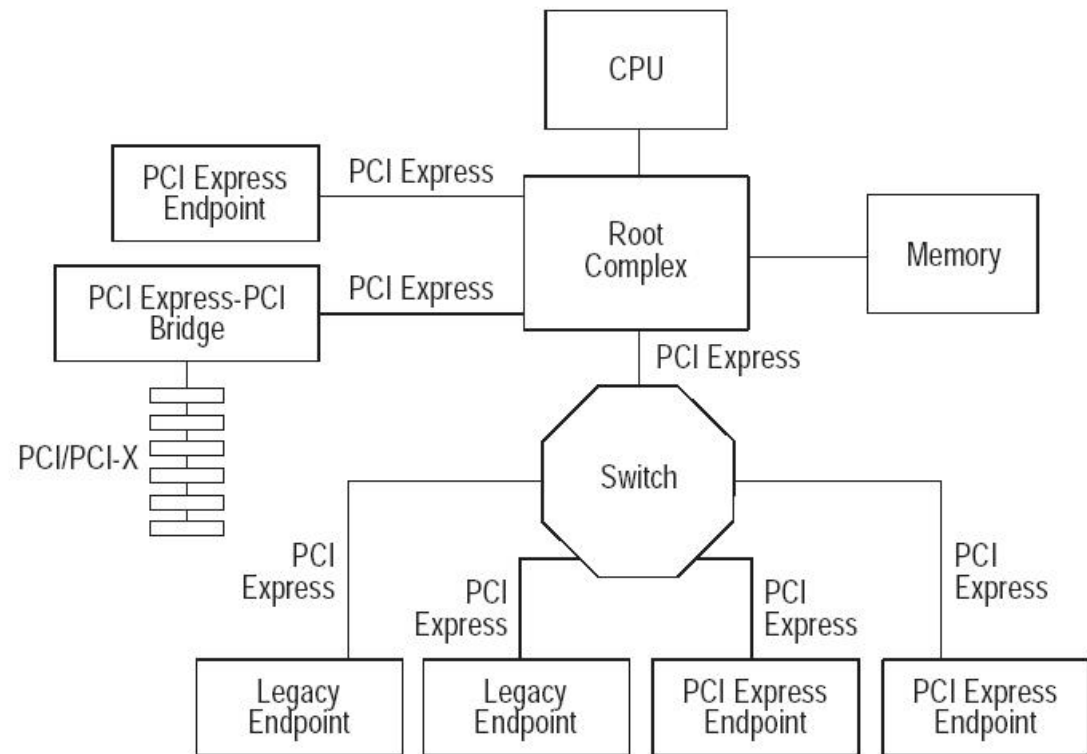
- ❑ La velocidad cada vez mayor (33MHz, 66MHz, 132MHz) de los buses PCI comenzó a imponer serias limitaciones a la arquitectura tipo BUS
- ❑ Los tiempos de propagación estaban fuertemente limitados por la capacidad parásita de cada nuevo conector y agente conectado al bus
- ❑ Y por ello la cantidad de slots disponibles en buses cada vez más veloces era cada vez menor
- ❑ Lo que limitó fuertemente la posibilidad de conectar periféricos no estándar al sistema
- ❑ La solución PCI-Express dedica un bus especial (canal o lane) a cada slot, pero para limitar la cantidad de líneas cada bus es serial (un canal en cada sentido, “dual simplex”, hacia y desde el slot) y emplea líneas diferenciales a 2.5 Gigabits/seg
- ❑ En algunos casos ciertos slots puede tener varios canales (multilane), desde solo 1 hasta 32

Tipo de bus	Frecuencia de reloj	Ancho de banda pico	Slots disponibles
PCI-32bits	33 MHz	133 Mbytes/s	4 a 5
PCI-32 bits	66 MHz	266 Mbytes/s	1 o 2
PCI-X-32 bits	66 MHz	266 Mbytes/s	4
PCI-X-32 bits	133 MHz	533 Mbytes/s	1 o 2
PCI-X-32 bits	266 MHz	1066 Mbytes/s	1
PCI-X-32 bits	533 MHz	2131 Mbytes/s	1

*1 único **lane** a 2.5Gbps equivale a 312 Mbytes/seg en cada sentido, y 32 **lanes** a 2.5Gbps transportan 80 gigabit/s, es decir 10 Gigabytes/seg en cada sentido!!! Y esto para cada slot en forma independiente!!!*

Topología de un sistema PCI-Express

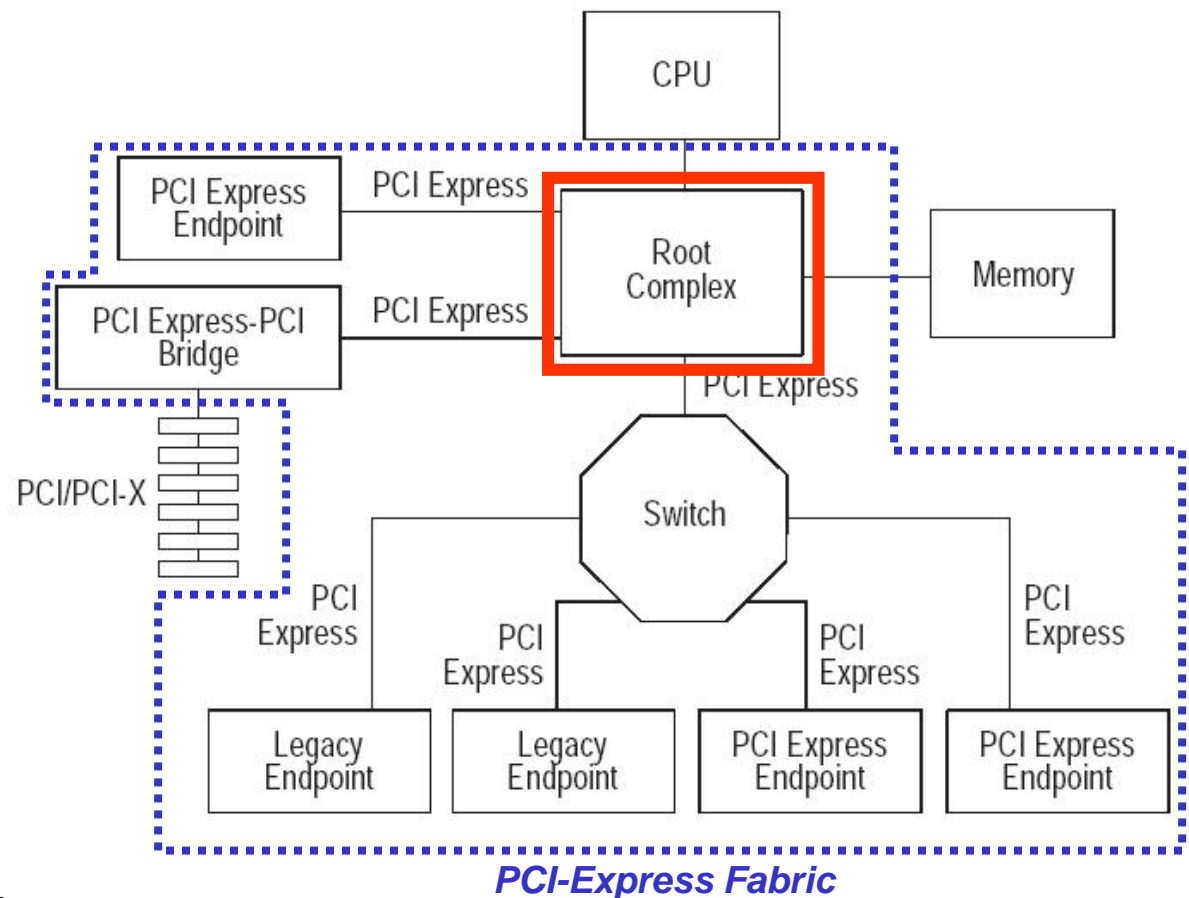
- ❑ La arquitectura de un sistema PCI-Express tiene forma de árbol, donde de un nodo central (**Root Complex**) surgen ramas que pueden ir a dispositivos extremos (**Endpoints**, o **Bridges** a otros tipos de buses como PCI o PCI-X), o a nodos que subdistribuyen esas ramas a otras ramas (**Switches**)
- ❑ EL nodo raíz tiene a su vez conexión con la CPU y la memoria masiva
- ❑ La comunicación se realiza punto a punto y en forma serial a través de paquetes con formatos definidos
- ❑ Los nodos terminales tipo *Legacy* toleran transacciones propias de sistemas PCI previos, en cambios los *PCI Express Endpoints* toleran transacciones especiales



El uso de una estructura tipo árbol de enlaces simples define canales UpStream (hacia la raíz) y DownStream (hacia los Endpoints). A su vez las puertas de un Switch también son UpStream o DownStream.

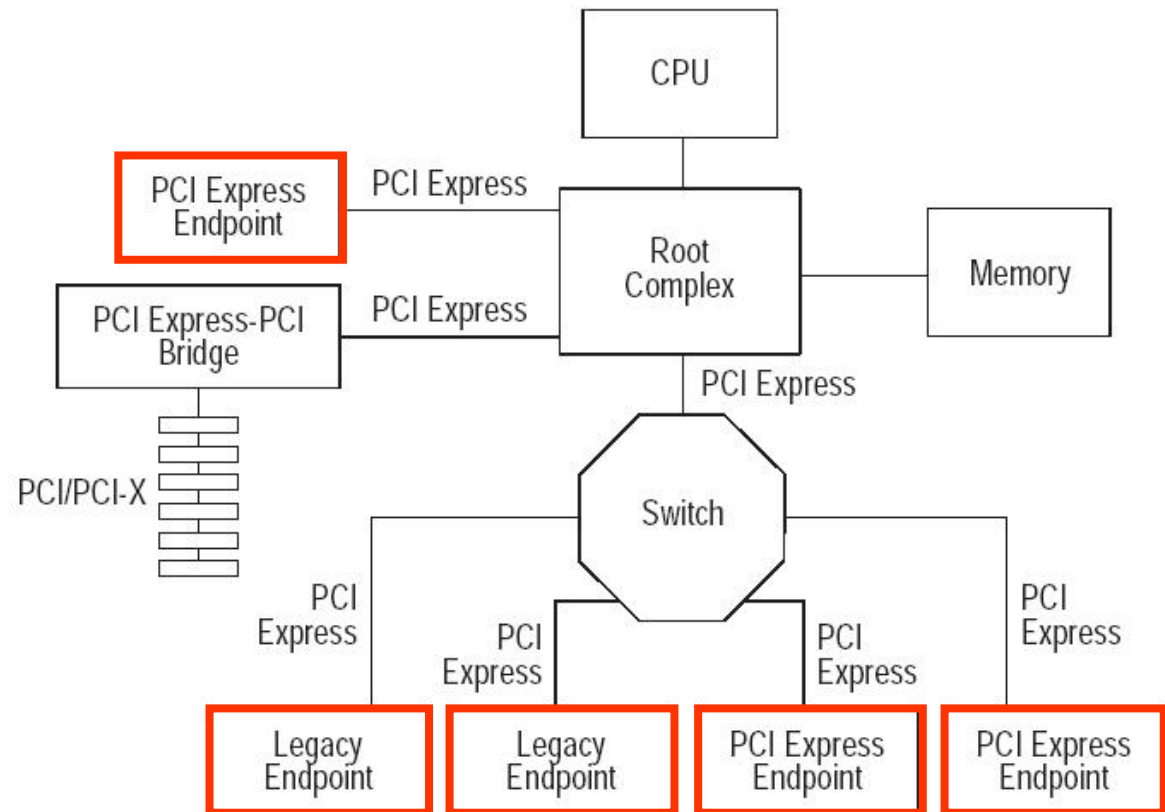
El nodo raíz (Root Complex)

- ❑ Conecta a la CPU y al sistema de memoria masiva al sistema PCI-Express (se usa el término PCI-Express Fabric) a través de puertas (ports).
- ❑ Cada puerta se conecta a un endpoint o a un switch, generando un sub-jerarquía
- ❑ Genera transacciones a pedido de la CPU, para configurar el sistema y acceder a memoria o a I/O.
- ❑ Un nodo raíz *puede* rutear paquetes de un Endpoint a otro, pero esto no es exigido por la norma PCI-Express
- ❑ Las funcionalidades de un nodo raíz incluyen manejo de conexiones dinámicas (*hot plug*), de consumo de potencia, de interrupciones, y detección de errores



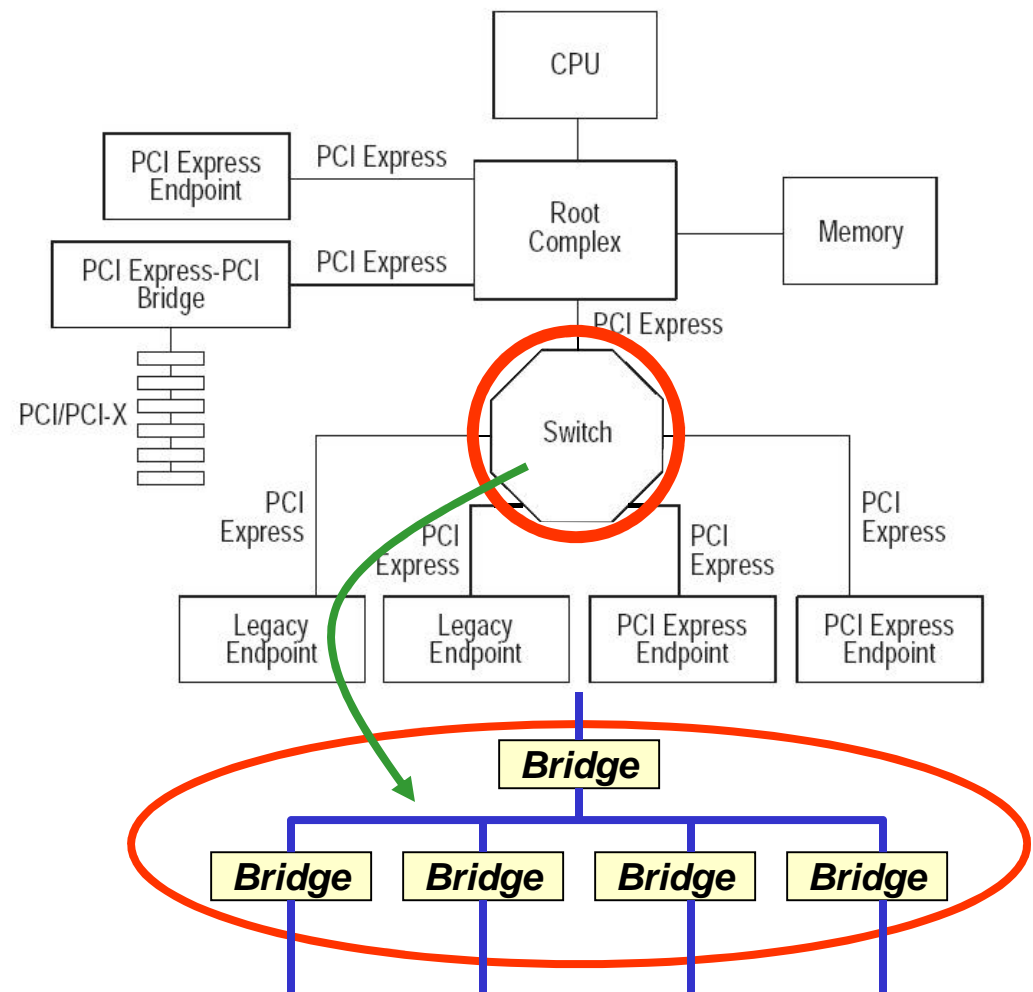
PCI-Express Endpoint

- ❑ Corresponde a circuitos diseñados específicamente para cada aplicación, que funcionan como extremos en la cadena de downstream
 - ❑ Por ejemplo, dispositivos Ethernet, USB o graficos
- ❑ **PCI Express Endpoint**
 - ❑ Diseñado para el aprovechamiento óptimo de las funciones PCI Express
- ❑ **Legacy Endpoint**
 - ❑ Soporta transacciones y métodos de interrupción similares a los disponibles en las normas PCI previas
- ❑ Tal como en PCI, cada Endpoint puede soportar desde 1 hasta 8 distintas funciones



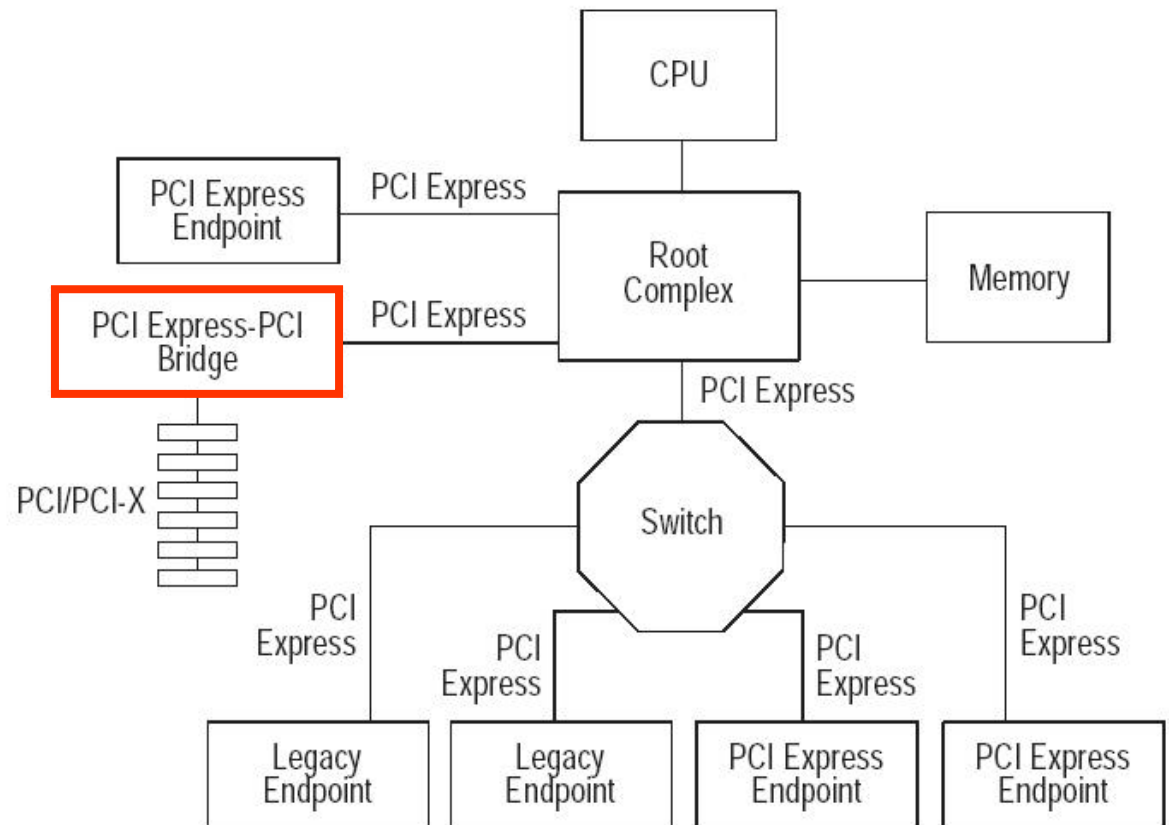
PCI-Express Switch

- Puede ser visto como múltiples bridges virtuales PCI-Express a PCI conectado con bridges PCI a PCI-Express, con sus correspondientes puertas
- Conecta un flujo de datos ascendente (upstream link) a varios flujos de datos descendentes (downstream links)
- Debe manejar mecanismos de arbitraje y de control de flujo, establecidos en función de los circuitos virtuales y los parámetros de calidad de servicio

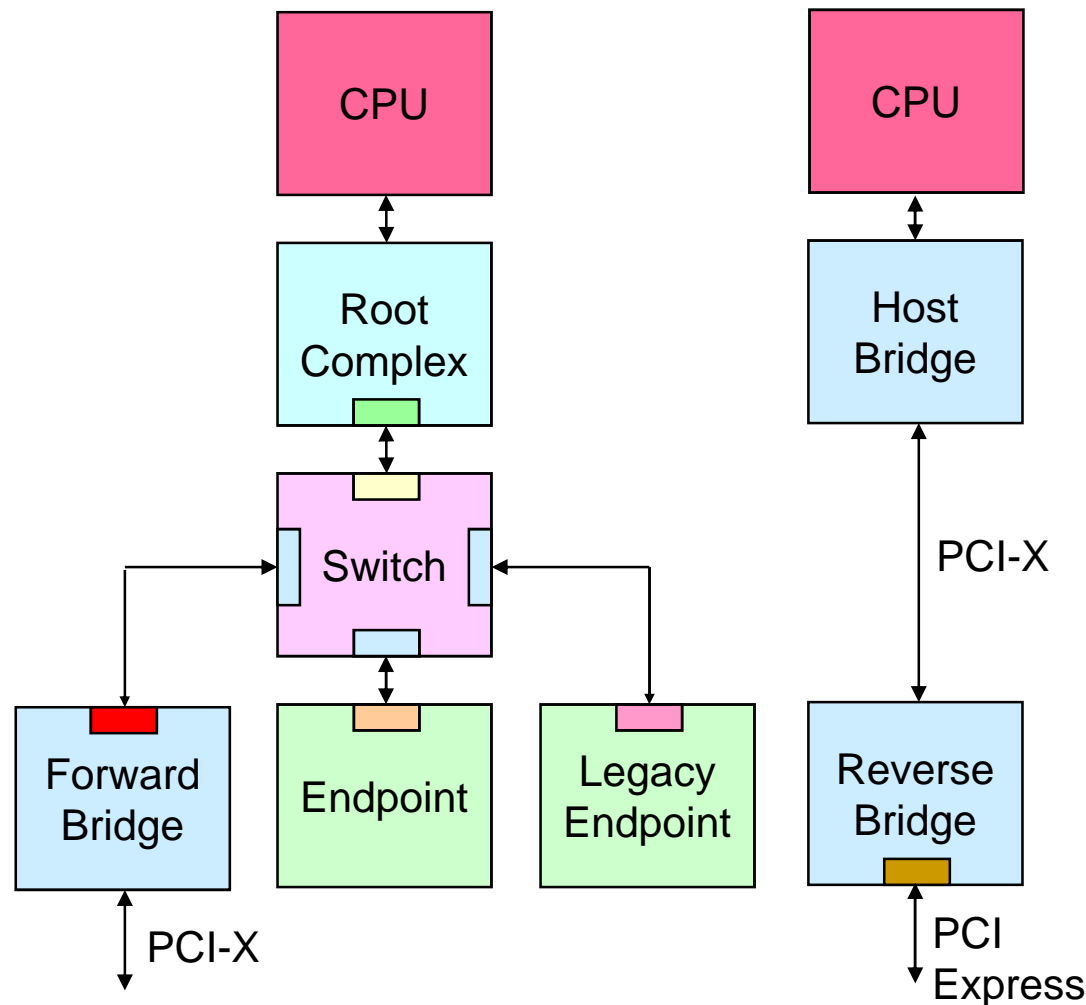


PCI Express: PCI Bridge y Hub Controller

- Permite conectar un bus paralelo PCI o PCI-X a un bus PCI-Express, operando como bridge
- Un HUB Controller es el caso en que además de convertir de PCI-Express a PCI, también se convierte a otros buses, como USB 2.0, IEEE1394, IDE, ATA, LPC, AC'97, u otros



Los 7 posibles tipos de puerta PCI-Express



Tipos de puerta PCI Express

- Root Complex Port
- Upstream Switch Port
- Downstream Switch Port
- Endpoint Port
- Legacy Endpoint Port
- Forward Bridge Port
- Reverse Bridge Port

Cada tipo de puerta, si bien tiene características eléctricas similares, tiene distinto comportamiento funcional y administra recursos muy diferentes

Distintos fabricantes de IP sólo ofrecen algunos de estos tipos

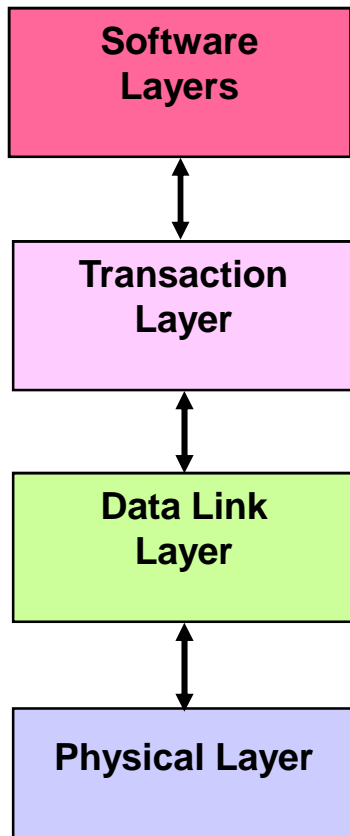
Transacciones PCI Express

- ❑ Las transacciones en PCI-Express pueden ser:
 - ❑ Del Root Complex a un Endpoint
 - ❑ De un Endpoint hacia el Root Complex
 - ❑ Entre Endpoints
- ❑ Cada transacción se realiza mediante la transmisión y recepción de paquetes, llamados Transaction Layer packets (o TLPs)
- ❑ Las transacciones pueden ser:
 - ❑ Non-Posted: cuando el *Requester* transmite un *TLP request packet* al *Completer* y el *Completer* devuelve un *TLP completion packet* al requester
 - ❑ Posted transactions: cuando el *Requester* transmite un *TLP request packet* al *Completer*, pero éste no retorna un *TLP completion packet*

Transaction Type	Non-Posted vs. Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read	Non-Posted
Configuration Write	Non-Posted
Message	Posted

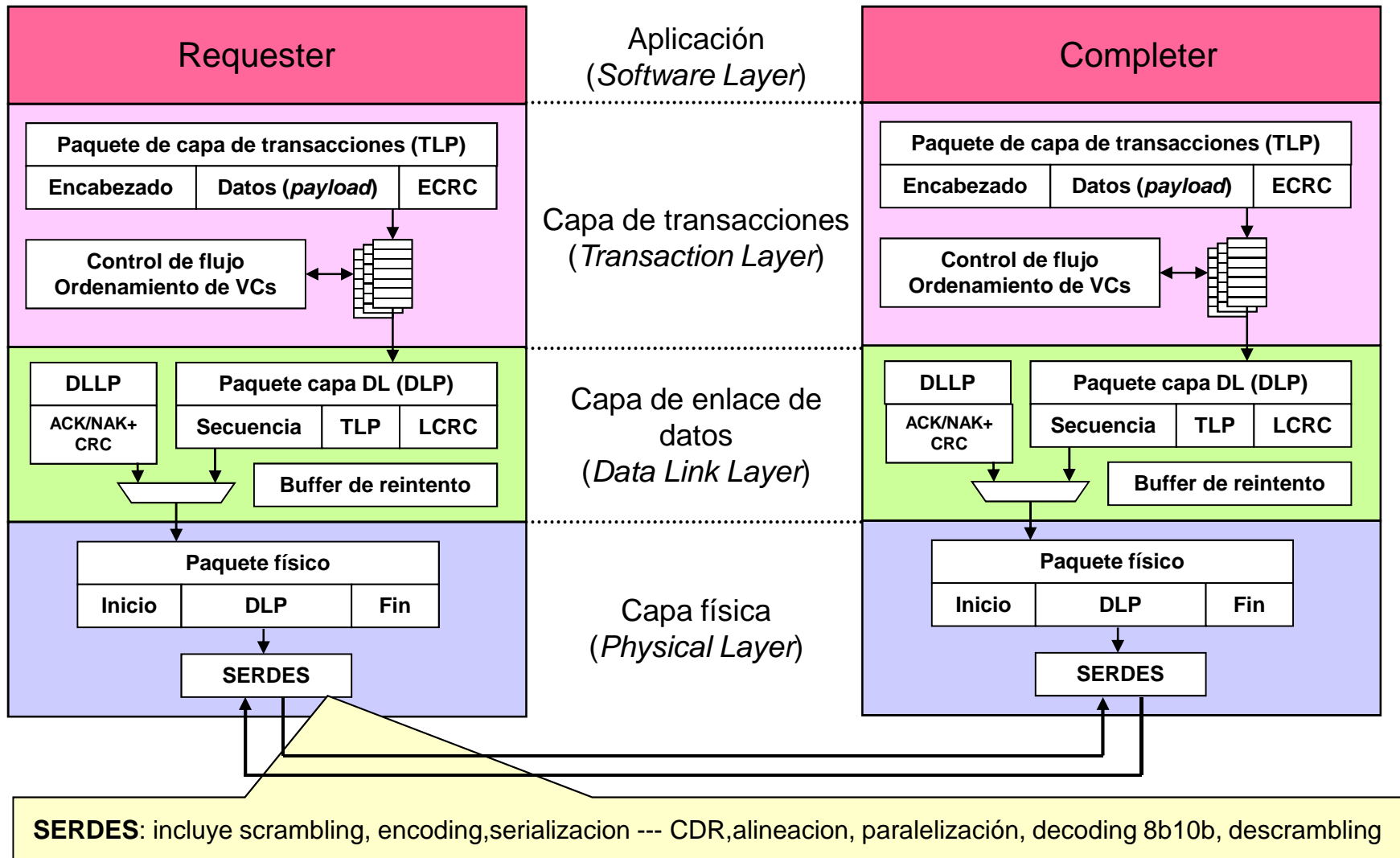
Capas del protocolo PCI Express

- El modelo del protocolo PCI Express puede ser descripto mediante el modelo de capas



- Las **capas de software (Software Layer(s))** realizan la interfase del elemento de conexión con el software con idénticas funcionalidades que las de una interfase PCI estándar
- La **capa de transacciones (Transaction Layer)**, resuelta en hardware, crea, transmite y recibe paquetes (Transaction Layer Packets o TLPs)
- La **capa de enlace (Data Link Layer)** se encarga del correcto envío y recepción de los paquetes, y de otras acciones que hacen al manejo general del enlace
- La **capa física (Physical Layer)** agrega a estos paquetes cierta información y los transmite por el medio físico.

Capas del protocolo PCI Express



Los paquetes (TLPs) en la Transaction Layer

- ❑ Responsable de la generación de los TLPs salientes, y de la recepción de los TLP entrantes
- ❑ Los paquetes se componen de:

Header	Data Payload	ECRC
3-4 Doublewords	0-1024 Doublewords	1 Doubleword

- ❑ Un encabezado (**Header**), formado por 3 o 4 doublewords que contienen información “logística”
 - ❑ Los datos a ser transportados (**Data Payload**), que pueden ser hasta 4 kbytes
 - ❑ Información de validación opcional (**ECRC**)
- ❑ Dado que el *Payload* es de tamaño variable, el largo total del TLP también lo es

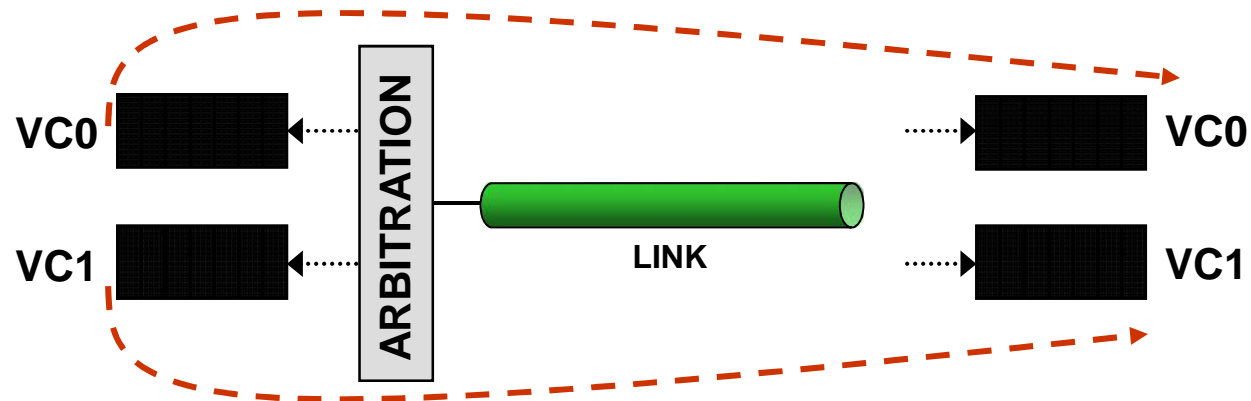
El header en un Transaction Layer Packet

El **Header** es de 12 o 16 bytes, y define los siguientes campos de la transacción:

- ❑ **Tipo de transacción:** 7 bits
 - ❑ el tamaño del header (3 o 4 DW)
 - ❑ si se transportan o no datos
 - ❑ si es una acceso a memoria, a I/O, a espacio de Configuración de tipo 0 o 1
 - ❑ si es de lectura o escritura
 - ❑ si hay reserva de acceso (Locking) en los accesos de lectura a memoria
 - ❑ si es un mensaje
 - ❑ si es una transacción de cierre (Completion)
- ❑ **Destinatario:** este campo depende del tipo de transacción, cuál es su origen, su destino, quién la originó, quién la satisface, y en accesos a memoria la dirección de memoria de destino.
- ❑ **Tamaño de los datos transmitidos:** 10 bits con el tamaño de los datos útiles transmitidos desde 1 DW a 1024 DW (4096 bytes)
- ❑ **Otros:** Ordenamiento de datos, manejo de memoria cache, validez de los datos, presencia o no de ECRC, clase de tráfico, bytes válidos en la primer y en la última DW

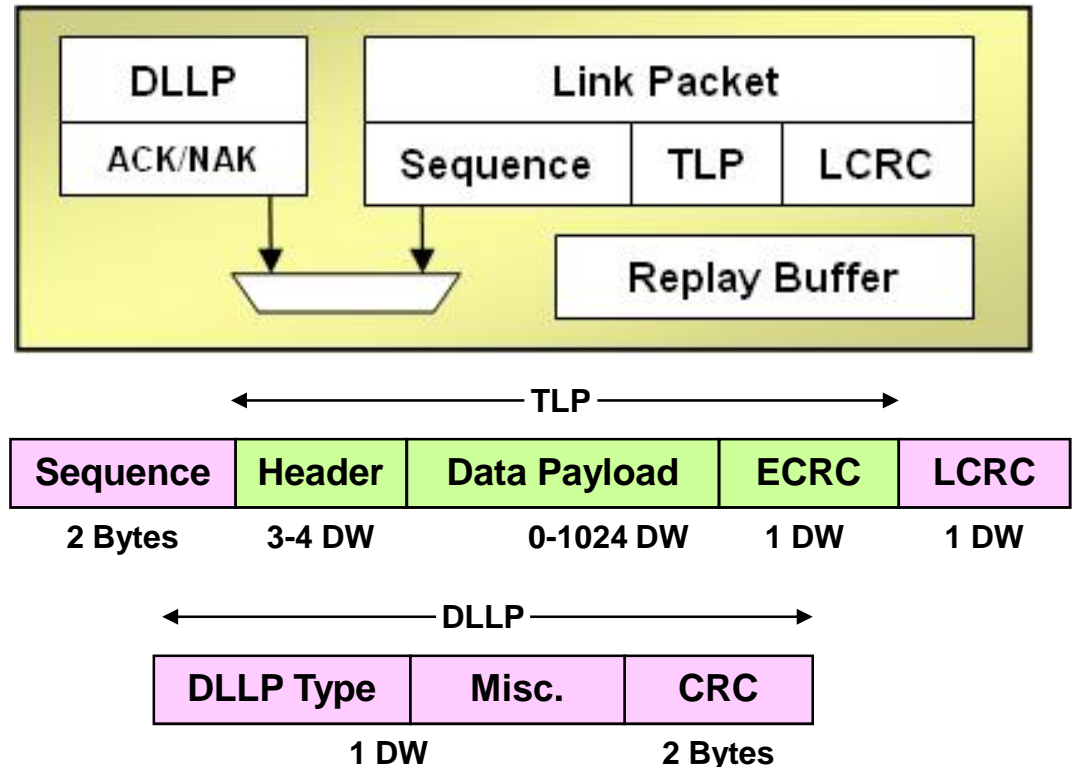
Canales virtuales (VC) y control de flujo en la Transaction Layer

- ❑ PCI-Express agrega el concepto de hasta 8 clases de tráfico (**TC[7:0]: Traffic Class**) asignadas a los TLPs que definen la prioridad de acceso entre paquetes que contienen por el uso de un mismo enlace físico
- ❑ Y hasta 8 buffers llamados **Virtual Channel Buffers (VC[7:0])** a los que son asignados distintas (una o mas) clases de tráfico. Por ejemplo, si se dispone de dos **VC**, las clases de tráfico **TC[2:0]** pueden mapearse a **VC0** y **TC[7:3]** a **VC1**.
- ❑ Además, entre originario y destinatario de un VC existe intercambio de información de control que permite realizar el control de flujo (**Flow Control**), es decir garantizar que un paquete no sea transmitido si no hay espacio en el destino para recibirlo



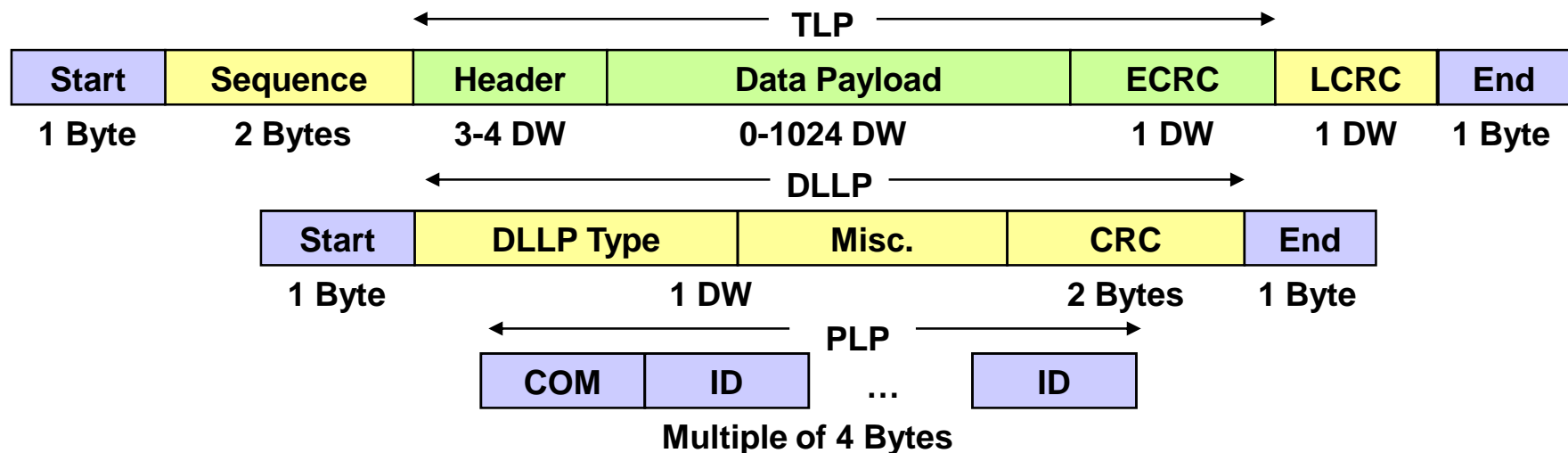
La capa de enlace (Data Link Layer)

- En esta capa a los TLPs se les agrega información de control de errores y para garantizar el orden de arribo de secuencias de paquetes, generando un **LP (Link Packet)**
- Cada **LP** enviado es salvado en forma temporaria en un buffer de replay en el caso de que no se reciba un **ACK** o se reciba un **NACK**.
- Además de los LPs, esta capa puede generar paquetes de administración de la capa de enlace, llamados **Data Link Layer Packets (DLLPs)**
 - Power Management
 - Flow Control
 - ACK/NAK

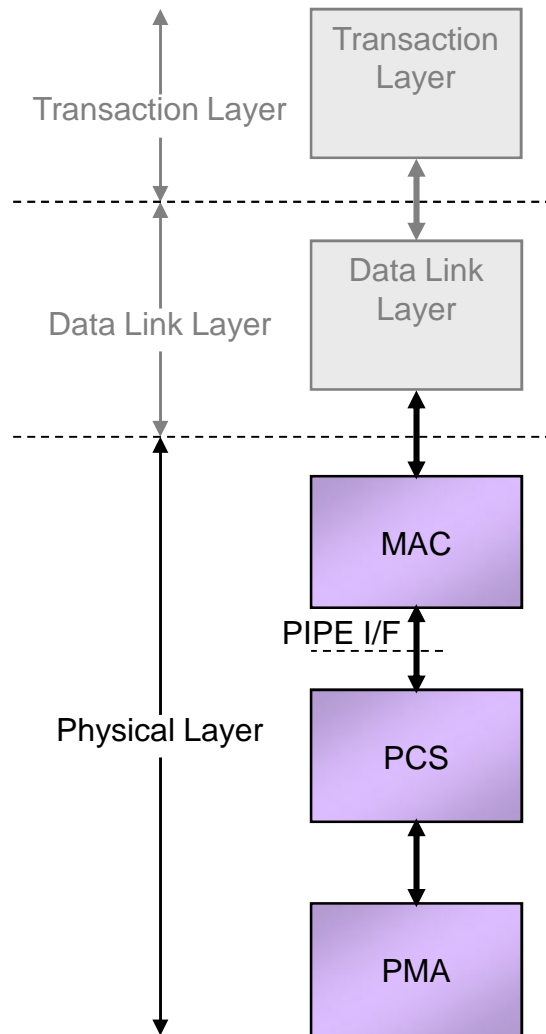


PCI-Express: la capa física (Physical Layer)

- ❑ Responsable de la transmisión física (eléctrica, serial, etc) sobre el enlace (uno o más *lanes*)
- ❑ Tiene circuitos lógicos encargados de tareas de byte stripping, scrambling, codificación y decodificación 8b10b, serialización, CDR, alineación de datos, etc..
- ❑ Y transceivers analógicos encargados de generar los niveles eléctricos necesarios, con el pre-énfasis en el transmisor y ecualización en el receptor
- ❑ Además de agregar información mínima a los paquetes de la capa de enlace, esta capa puede generar paquetes muy simples (**Physical Layer Packets** o **PLPs**)



Subdivisión de la capa física

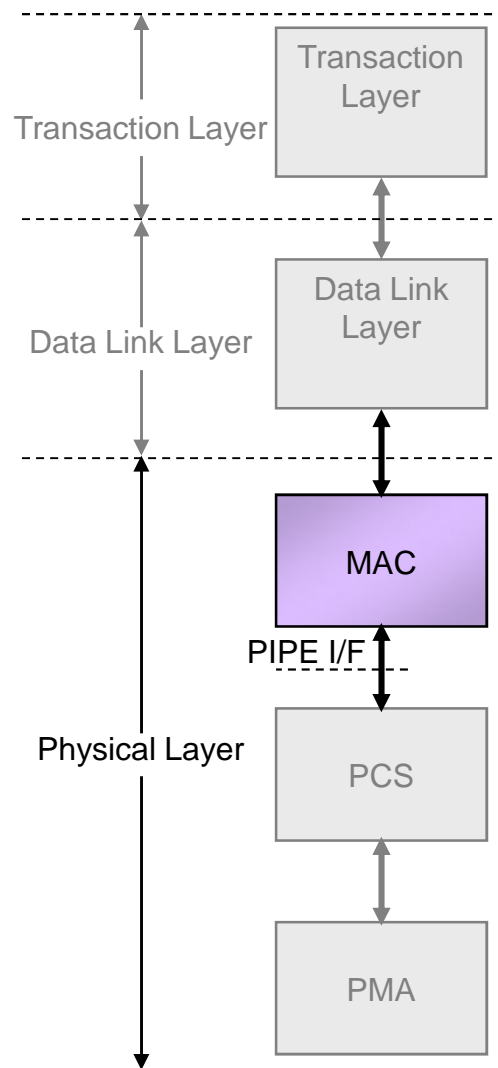


La especificación subdivide la capa física en tres niveles:

- ❑ **MAC (Media Access Layer):** responsable de las máquinas de estado que administran el enlace y de reconfigurar los paquetes en el caso de enlaces que emplean varios canales físicos simultáneos (de-skew en enlaces multi-lane)
- ❑ **PCS (Physical Coding Sublayer):** realiza la codificación y decodificación 8b/10b, el agregado de caracteres de control (K-Characters), entre ellos los “comma characters”. Suele incluir un buffer que puede agregar o quitar caracteres especiales de relleno durante la transmisión y quitarlos en la recepción para acomodarse a los relojes de transmisión y recepción.
- ❑ **PMA (Physical Media Attachment Layer):** responsable de los procesos de serialización y deserialización, y de generación o detección según los niveles eléctricos requeridos por la norma.

Con PCI-Express operando a 2.5Gbps, es usual resolver las capas PCS y PMA con dispositivos ASIC, por lo que entre las subcapas MAC y PCS existe una interfase estándar definida por INTEL para circuitos que usan transceptores externos: **PIPE (“Physical Interface for PCI Express”)**

Subdivisión de la capa física MAC (Media Access Layer)

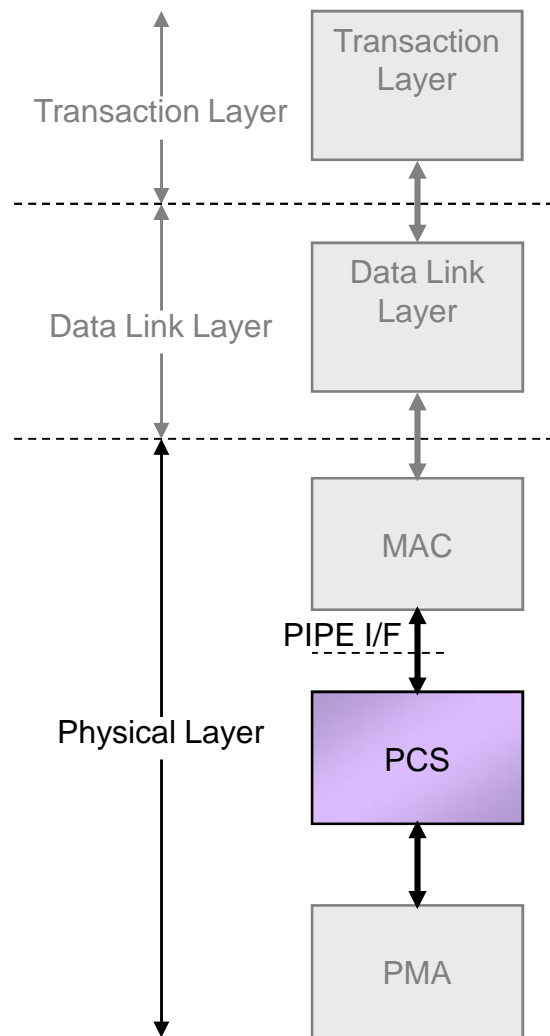


Esta capa maneja enlaces que ocupan uno o más enlaces físicos (*multi-lane*) realizando tareas de multiplexado, rearmado, y de control.

❑ **En la transmisión**, los paquetes del DLL ingresan a un buffer donde se les agrega caracteres de control para indicar su inicio (**STP (K27.7)** si es un TLP y **SDP (K28.2)** si es un DLLP) y su fin (**END (K29.7)** o **ENB (K30.7)**), y de relleno (**PAD (K23.7)**) para completar las DWords parcialmente válidas. Además esta etapa envía periódicamente **PLPs**: paquetes IDLE (4 caracteres **IDL(K28.3)**), y los llamados Ordered-Sets, usados para relleno entre paquete útiles (carácter de control **COM (K28.5)** seguido de 3 caracteres **SKP (K28.0)**), secuencias de training o manejo de potencia. Si el enlace es multi-lane, los paquetes TLP y DLLP son distribuidos de a byte a etapas PCS responsables de cada lane; los PLPs se transmiten simultáneamente por todos los lanes.

❑ **En la recepción**, se remueven los caracteres de relleno y se realizan las acciones de sincronismo en base a los caracteres de control. Para el caso de enlaces multi-lane existe un buffer y circuitos de control para ir reagrupando datos de un único paquete de datos pero que van ingresado en paralelo por distintos transceivers (*channel bonding*).

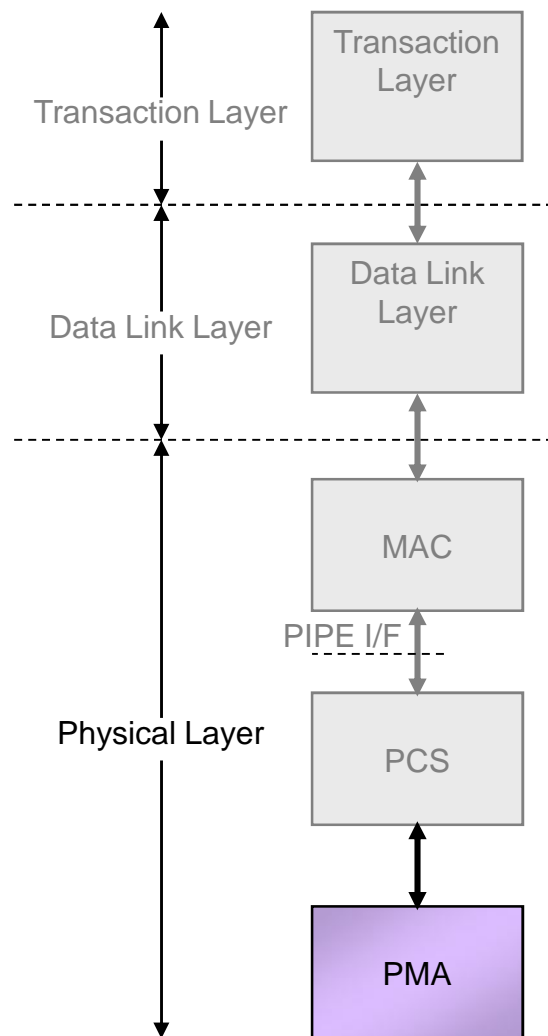
Subdivisión de la capa física: PCS (Physical Coding Sublayer)



Contiene el codificador y decodificador 8B/10B, etapas de CRC y un buffer elástico capaz de soportar correcciones de reloj.

- ❑ **En la transmisión**, los datos pasan por un scrambler (se hace su XOR con un generador pseudo aleatorio de ecuación $(X^{16}+X^5+X^4+X^3+1)$ a fin de eliminar patrones repetidos, luego por el encoder 8B/10B, y finalmente a un buffer FIFO de transmisión. El scrambler avanza de a 8 pasos por vez, se inicializa en FFFF con el K-character COM, no se actualiza en los caracteres SKIP, y sólo afecta a ciertos caracteres.
- ❑ **En la recepción**, los datos son convertidos de 10 bits a 8 bits, detectándose los caracteres de “control K”, problemas de disparidad y de códigos inexistentes, así como los caracteres de sincronismo (caracteres “comma” K28.1, K28.5 y K28.7). Luego del conversor 10B/8B existe un buffer de recepción “elástico” para acomodarse a diferencias entre la frecuencia de recepción de datos y su consumo por MAC, lo que se logra agregando o removiendo caracteres específicos, presentes en los períodos que separan entre sí a paquetes con datos significativos.

Subdivisión de la capa física: PMA: Physical Media Attachment



Contiene el serializador y el deserializador, incluyendo las etapas de etapas de CDR (Clock&Data Recovery).

- ❑ **En la transmisión**, un PLL genera los 2.5Gbps con los que los datos provenientes del FIFO de transmisión son serializados y salen en forma diferencial LVDS por dos salidas. Es posible controlar la polaridad, la impedancia de salida, el valor de pre-énfasis a aplicar (P.Ej: 10%, 20%, 25%, y 33%), y la amplitud diferencial (usualmente entre 400mV y 800mV).
- ❑ **En la recepción**, esta etapa emplea un lazo de enganche de fase (PLL) interno que extrae frecuencia y fase a partir de los datos ingresados. Luego de la separación de datos y reloj existe una etapa de alineación (Comma Detect Realign) que permite la sincronización de tramas a nivel de palabras detectando la ocurrencia de uno o dos códigos programables de 10 bits (típicamente, los caracteres de “control K” Comma+ y Comma-). Dentro del dispositivo es posible optar por la Z de terminación de recepción y eventualmente qué ecualización aplicar.