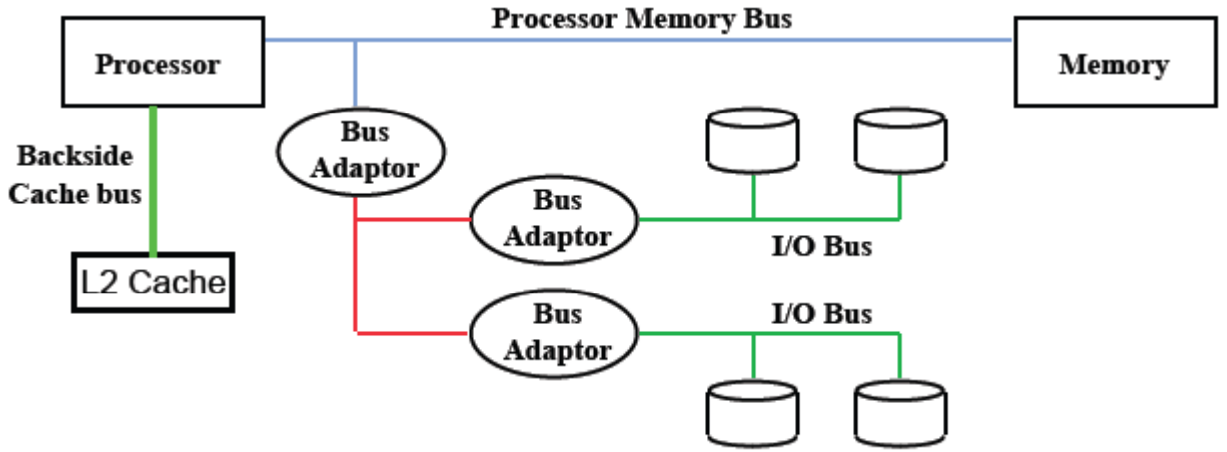
Sistemas electrónicos digitales

Tema 7:
Buses

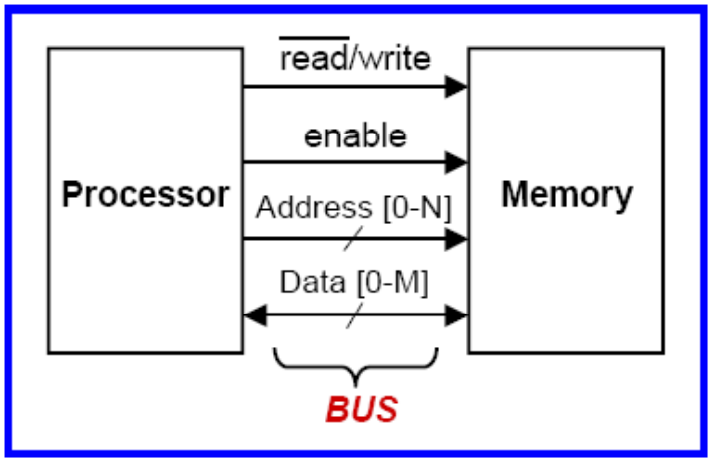


Índice del tema

- Introducción
- Ejemplos de conexionado
- Protocolos de comunicación
- Gestión de memoria en el ARM
- Interfaz hardware
- Ciclos de bus
- Resumen de líneas de interfaz
- El bus AMBA

Introducción: conceptos generales I

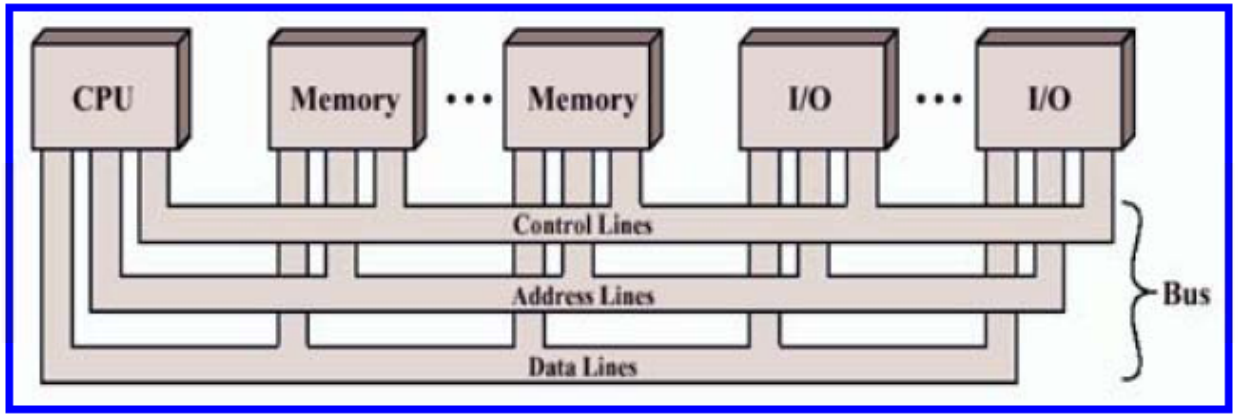
- Consiste en un conjunto de “cables” que conectan dos o más bloques, a menudo denominados “periféricos”
  - Cada uno de los “cables” puede ser unidireccional (*read, write*) o bidireccional (*data*)
  - Cada uno de los cables puede representar un único cable (*read, write*) o un conjunto de cables (*data, address*)



  - Un bus tiene siempre asociado un **protocolo** de comunicación
- Desventajas
  - Los buses suponen un cuello de botella en el sistema
  - La velocidad del bus está limitada por
    - La longitud del bus; el número de dispositivos conectados al bus

Introducción: conceptos generales II

- Un bus puede soportar los siguientes tipos de transferencias
  - Memoria → Procesador: operación de lectura, dato o instrucción
  - Procesador → Memoria: operación de escritura, dato
  - I/O → Procesador: lectura de dato/estado desde periférico
  - Procesador → I/O: escritura de dato/configuración en periférico
  - I/O ←→ Memoria: operación de DMA, necesidad de un controlador de DMA



Introducción: señales del bus

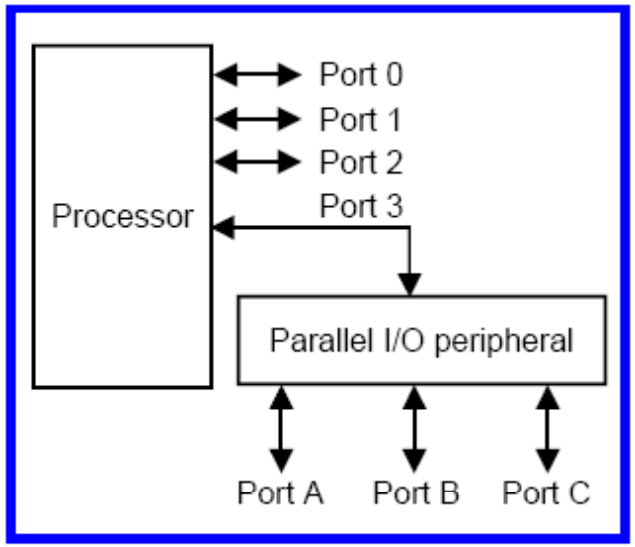
- Un bus contiene, generalmente, las siguientes señales
  - *Address*, *data*
  - Líneas de control
    - *Memory write*: señal de escritura en memoria
    - *Memory read*: señal de lectura en memoria
    - *I/O write*: señal de escritura en I/O
    - *I/O read*: señal de lectura en I/O
    - *Bus Req*: señal de petición de acceso a los buses
    - *Bus Grant*: señal de confirmación de la petición
    - *Int Req*: señal de petición de interrupción
    - *Int Grant*: señal de aceptación de la petición de interrupción
    - *Reset*: señal de inicialización del sistema
    - *Clock*: en buses síncronos, esta señal sincroniza todas las operaciones

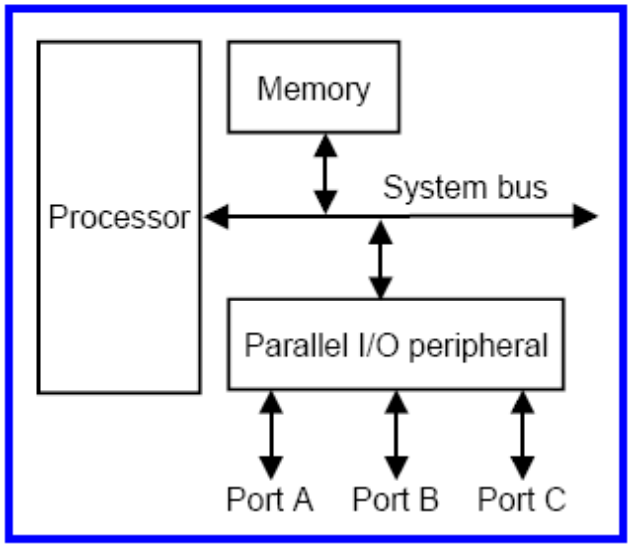
Write

Read

Introducción: comunicación basada en puertos

- Comunicación con periféricos basada en puertos (*I/O mapped*)
  - Uso de señales propias de lectura y escritura (*I/O read and I/O write*), así como instrucciones propias de I/O
  - Un puerto está conectado directamente a un registro dedicado
  - Posibilidad de añadir más puertos mediante dispositivos periféricos
- Comunicación con periféricos por acceso a memoria (*memory-mapped*)
  - Señales únicas de lectura y escritura, muchas veces multiplexadas
  - La comunicación con los puertos se realiza localizando éstos en posiciones de memoria



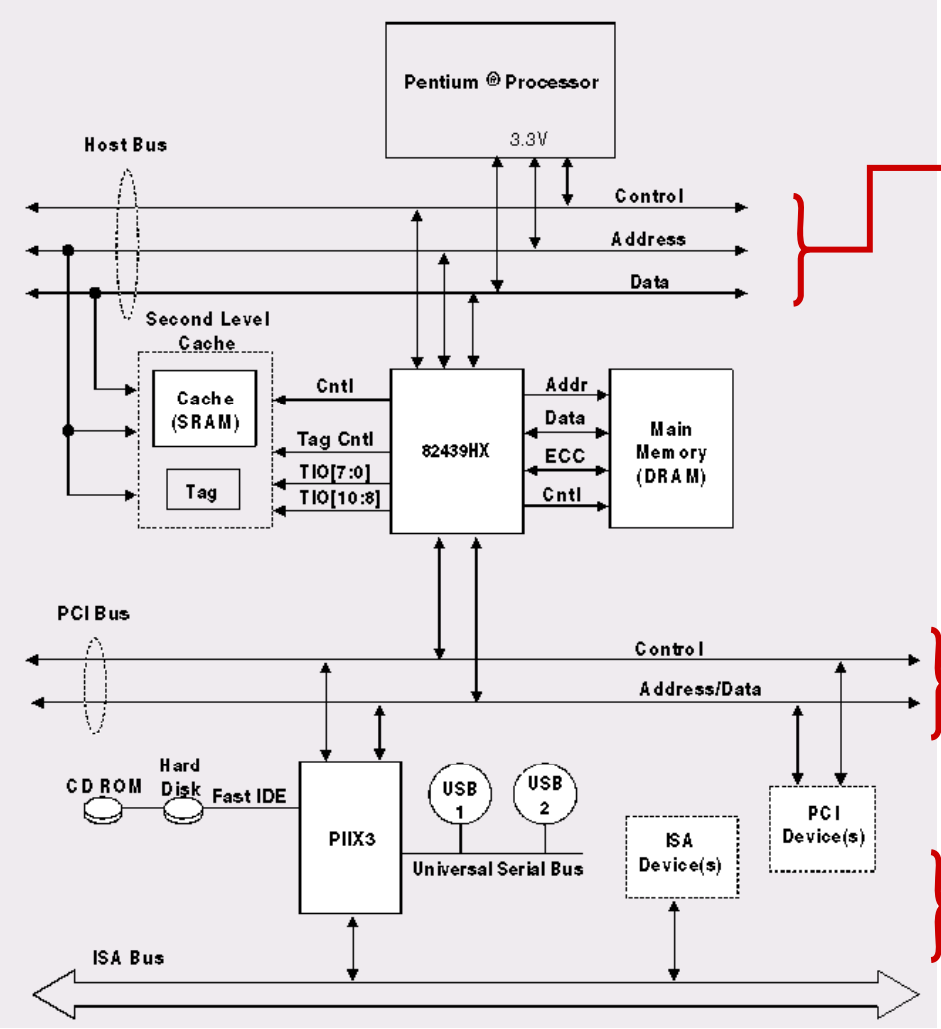


Introducción: tipos de buses

- *Processor-Memory Bus* (específicos):
  - Cortos y de alta/muy alta velocidad
  - Sólo necesitan ajustarse al interfaz de la memoria
    - Maximiza el ancho de banda de la comunicación
  - Conexión directa con el procesador
  - Optimizada para la transferencia de bloques (líneas de palabra)
- *I/O Bus* (estándar):
  - Más lentos y de mayor longitud
  - Necesitan ajustarse a un amplio abanico de periféricos
  - Se conectan al bus que une la memoria con el procesador
- *Backplane Bus* (estándar o propietario):
  - El bus se distribuye en el mismo “chasis” del sistema
  - Permite el conexionado de procesador, memoria e I/O
  - Ventajas en cuanto a coste, un único bus para todos

Introducción: tipos de buses, ejemplo

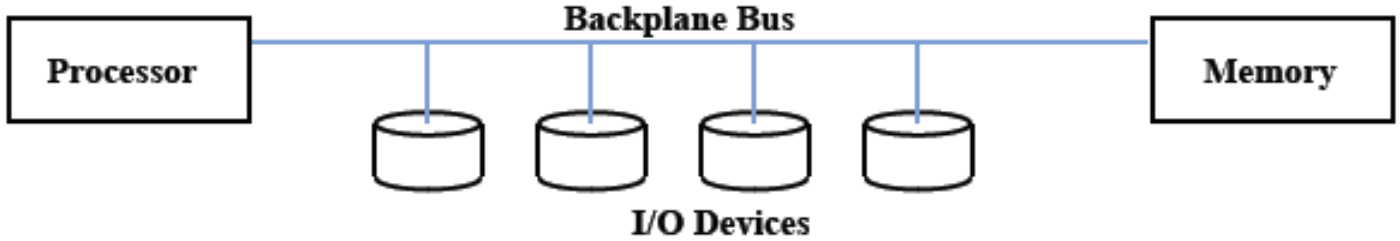
Organización de buses en un sistema Pentium



- *Processor-Memory Bus*
  - Diseño específico propietario de Intel
  - Señales ajustadas al interfaz del procesador
- *Backplane bus*: PCI
  - Conexión de dispositivos PCI
- *I/O Buses*
  - IDE, USB, SCSI

Ejemplos de conexionado: sistemas con un único bus

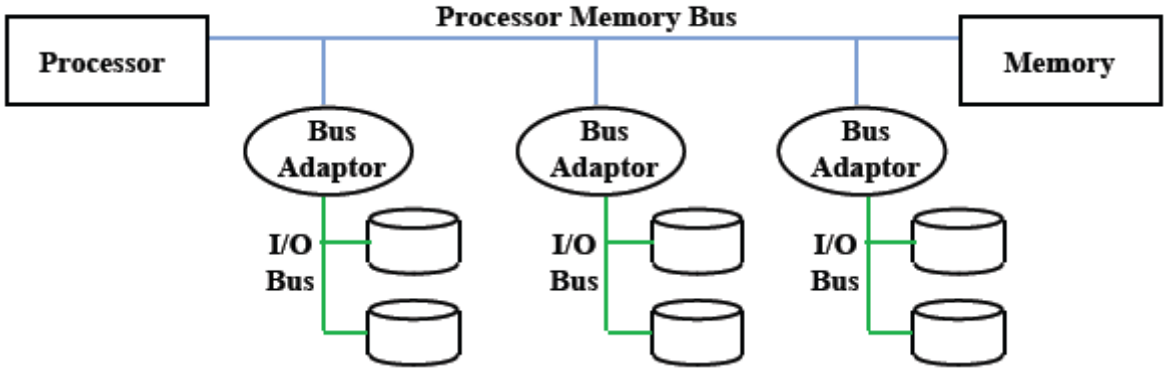
- *Backplane bus*



- Un único bus utilizado para
  - Comunicaciones entre procesador y memoria
  - Comunicaciones entre memoria y periféricos
- Ventajas
  - Simple y de bajo costo
- Desvetajas
  - Lento y, además, el bus representa el principal cuello de botella del sistema
- Ejemplo: IBM PC-AT

Ejemplos de conexionado: sistemas con dos buses

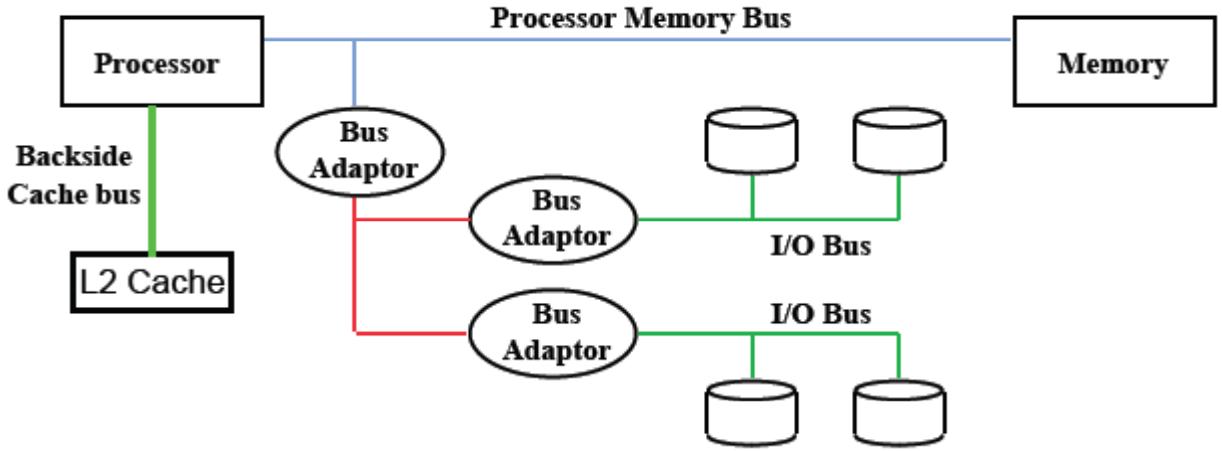
- *Processor-Memory Bus + I/O Bus*



- Conexionado de periféricos mediante adaptadores de buses, lo que permite ajustar las velocidades
  - *Processor-Memory Bus*: principalmente para tráfico entre memoria y procesador
  - *I/O Buses*: permite añadir slots de expansión para I/O
- Ejemplo: Apple Macintosh-II
  - NuBus: conexionado del procesador y memoria
  - SCSI Bus: conexionado del resto de periféricos

Ejemplos de conexionado: sistemas con tres buses

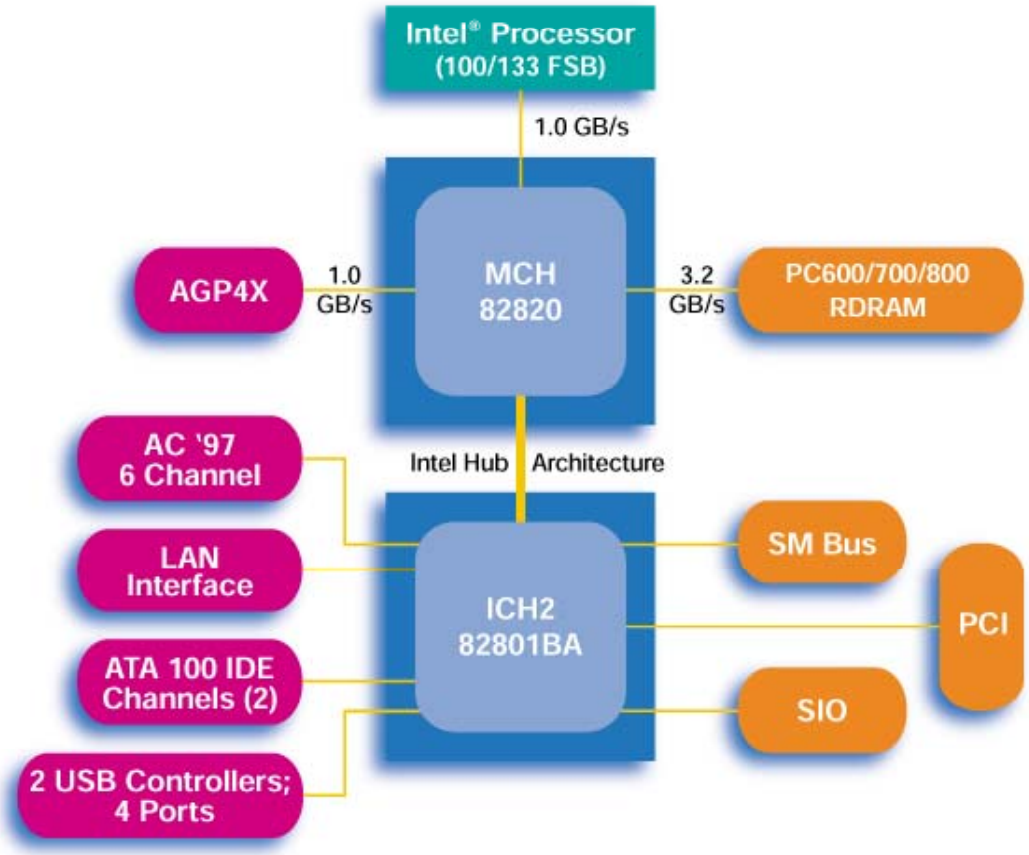
- + *backside cache*



- Uso de un reducido número de *backplane buses* conectados al bus del procesador
  - El bus del sistema queda enfocado a la comunicación con la memoria
  - Los buses de entrada/salida aparecen conectados al *backplane*
- Ventajas: se reduce la carga en el bus del procesador y permite el uso de buses con diferentes velocidades

Ejemplos de conexionado: configuración del Intel Pentium

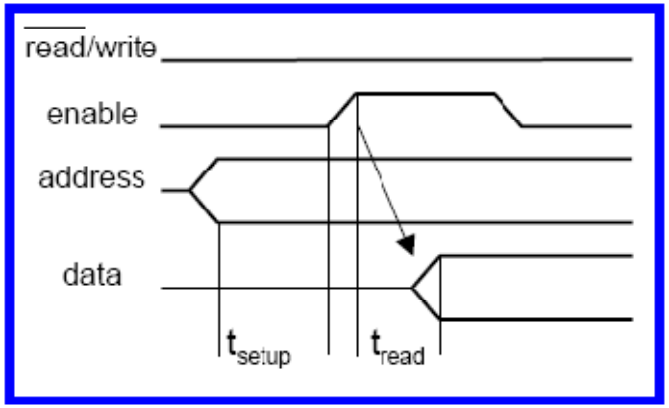
- *Northbridge*:
  - Memoria
  - Sistema Gráfico
- *Southbridge*:
  - *PCI bus*
  - *Disk controllers*
  - *USB controllers*
  - *Audio (AC97)*
  - *Serial I/O*
  - *Interrupt controller*
  - *Timers*



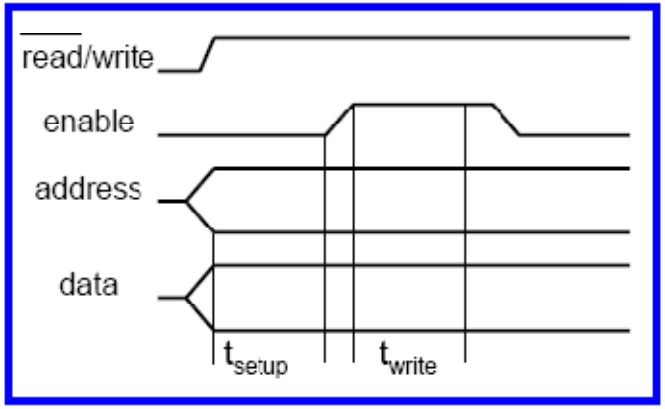
Protocolos de comunicación: *timming diagrams*

- La manera de describir un protocolo de comunicación es mediante cronogramas
- En un cronograma, las señales varían entre los niveles lógicos alto/bajo, mientras que los buses alternan entre datos válidos/no válidos
- Es importante atender a las especificaciones de tiempo establecidas
  - T_{stup}: tiempo de setup
  - T_{hold}: tiempo de hold
  - T_{access}: tiempo de acceso
  - T_{write}: pulso de escritura

Read protocol

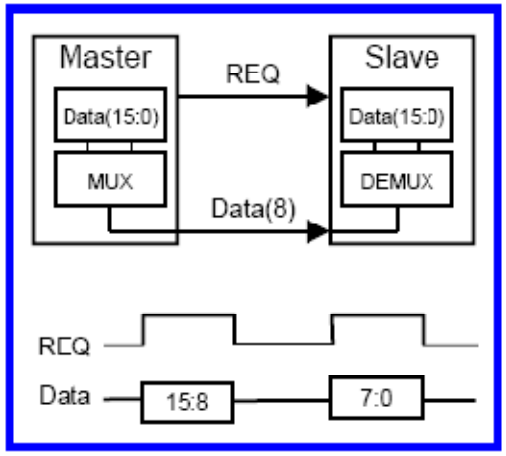


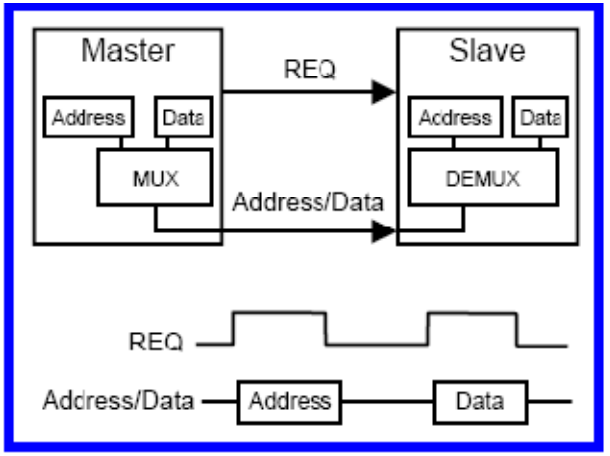
Write protocol



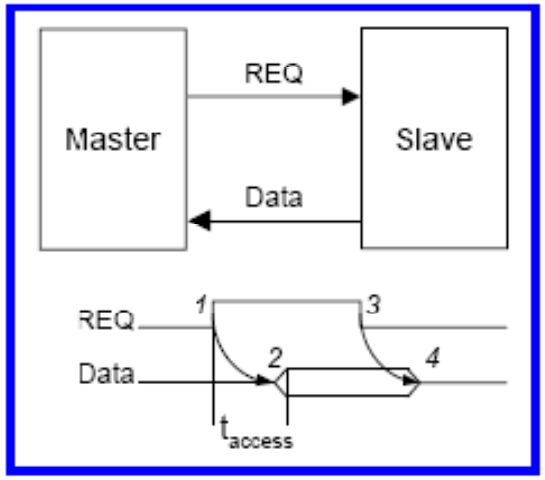
Protocolos de comunicación: *conceptos básicos*

- La comunicación engloba a dos actores: *master* & *slave*
- El *master* inicia la transferencia de datos y el *slave* responde a la solicitud. En caso de transferencia con memoria el procesador siempre actúa de *master*
- En cada transferencia existe un elemento que envía el dato (sender) y otro que lo recibe (receiver). La dirección del dato está definida en el propio protocolo (ejemplo write, read)
- Muchos protocolos gestionan datos y direcciones



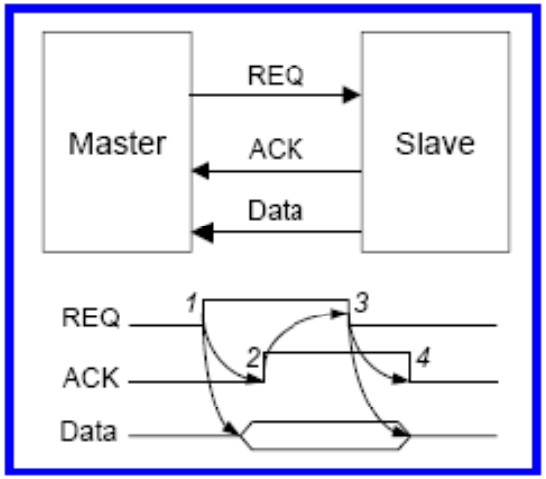


Protocolos de comunicación: métodos de control I



Strobe protocol

- *1.- Master set REQ to receive data*
- *2.- Slave puts data on bus after access time*
- *3.- Master receives data and resets REQ*
- *4.- Slave ready for next REQ*

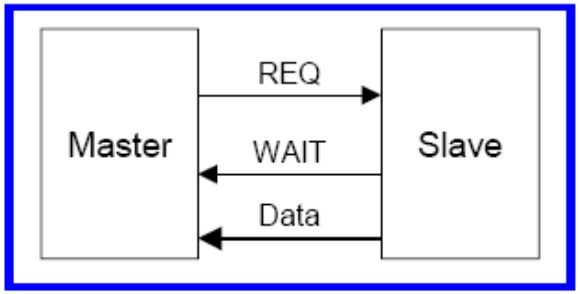


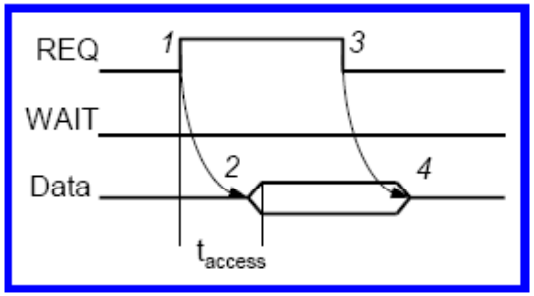
Handshake protocol

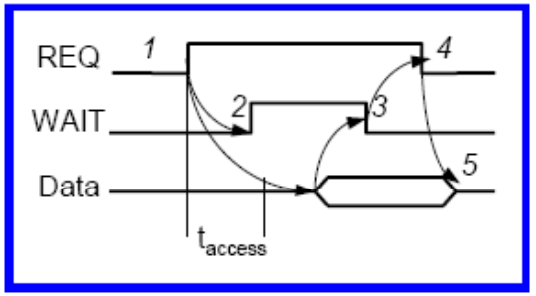
- *1.- Master set REQ to receive data*
- *2.- Slave puts data on bus and sets ACK*
- *3.- Master receives data and resets REQ*
- *4.- Slave ready for next REQ*

Protocolos de comunicación: métodos de control II

- El protocolo de *handshake* posibilita ajustarse a la velocidad de respuesta de los periféricos
- La principal desventaja aparece cuando el tiempo de respuesta es desconocido, lo que ralentiza la comunicación y añade líneas extras al protocolo
- Para conseguir la solución óptima se adopta un protocolo de compromiso (ejemplo protocolo ISA)



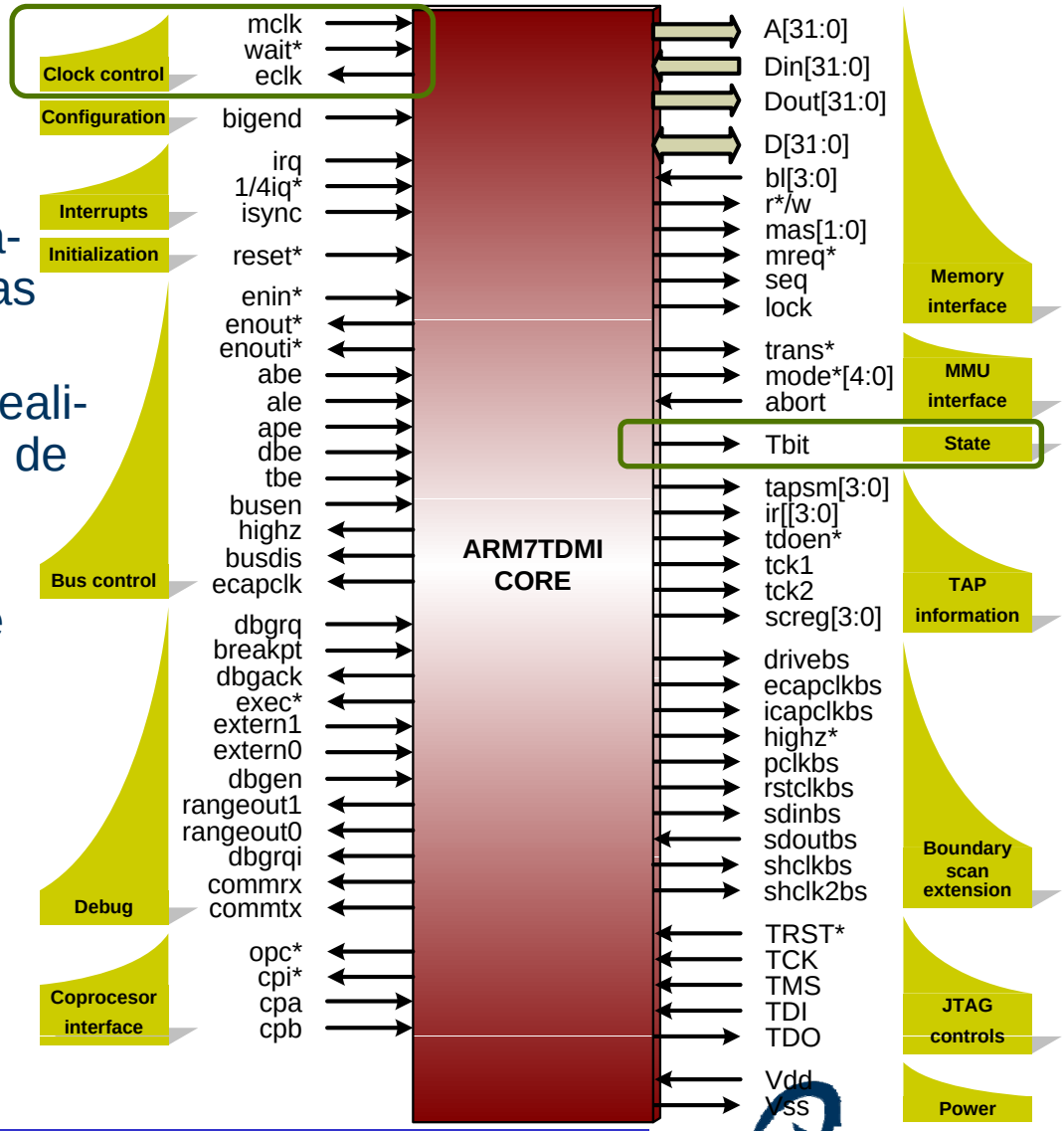




  - Se establece la señal WAIT en el protocolo de comunicación
  - Se adopta un protocolo de comunicación síncrono con tiempo de acceso establecido
  - En caso de no tener el dato preparado, se activa la señal WAIT

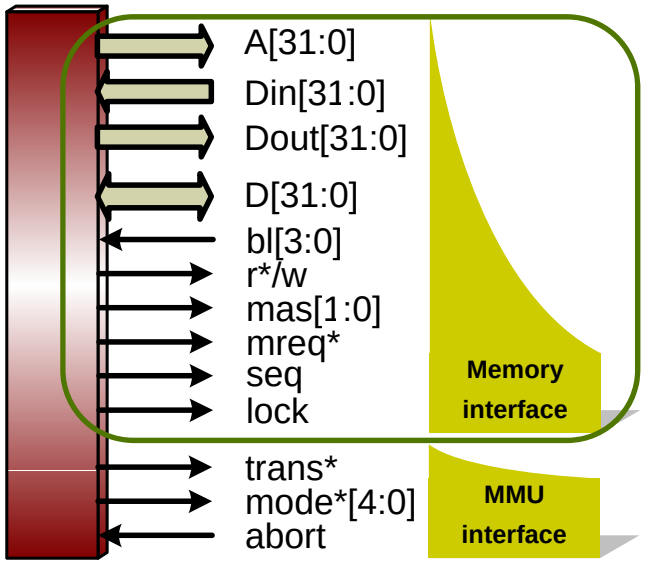
Gestión de memoria en el ARM: líneas de interfaz → *Control clock, State*

- Ejemplo ARM7TDMI
- Líneas de interfaz *Control clock*
  - *mclk* → todas las operaciones están controladas por el reloj de memoria
  - *wait** → utilizada para realizar bloqueos en el reloj de memoria *mclk*
  - *eclk* → señal de salida que refleja el estado de *mclk* incluso si está bloqueado por *wait*
- Líneas de interfaz *State*
  - *Tbit* → bit indicativo del tipo de modo
    - *Thumb* = 1
    - ARM = 0



Gestión de memoria en el ARM: líneas de interfaz → *Memory interface*

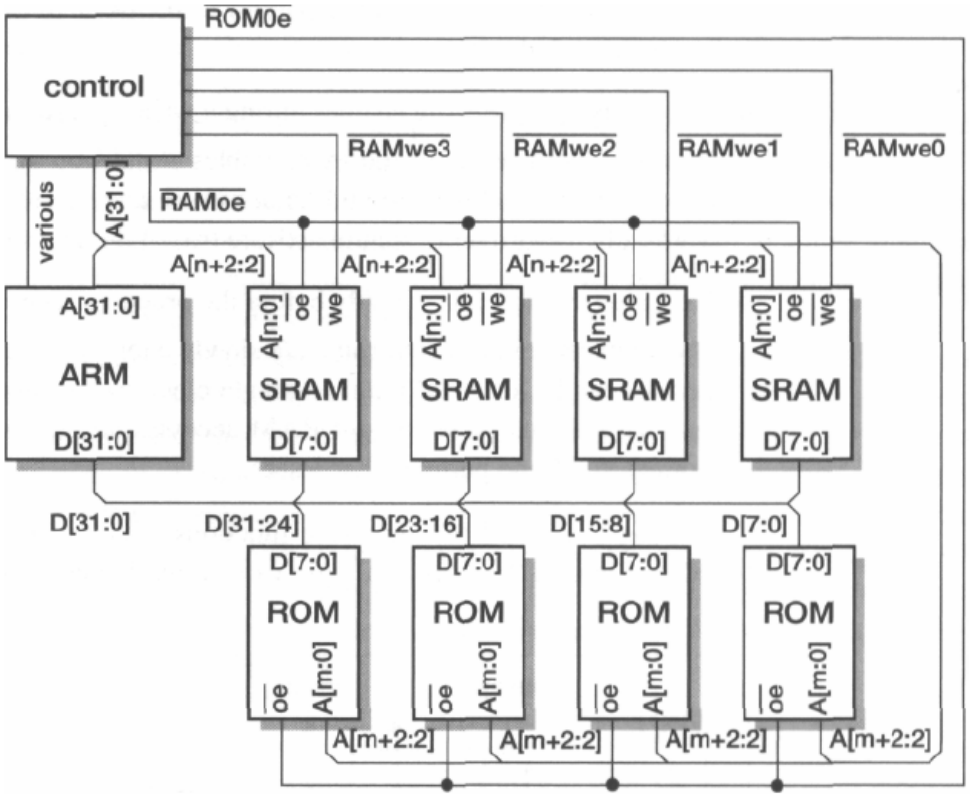
- Líneas de interfaz *Memory interface*
  - *A[31:0]* → bus de direcciones de 32 bits
  - *Din[31:0]* → bus de entrada de datos de 32 bits
  - *Dout[31:0]* → bus de salida de datos de 32 bits
  - *D[31:0]* → bus bidireccional de datos de 32 bits
  - *bl[3:0]* → habilita los latches en operaciones de entrada en caso de sistemas de memoria inferiores a 32 bits
  - *r*/w* → señal de lectura, a nivel bajo, y escritura
  - *mas[1:0]* → indica el tamaño de la transferencia
  - *mreq** → señal de solicitud de acceso a memoria



  - *seq* → señal que indica que la dirección de acceso a memoria se obtiene a partir de la anterior de forma secuencial
  - *lock* → indica que se está realizando una operación indivisible (ej. *swap*). Útil para el uso de semáforos.

Gestión de memoria en el ARM: interfaz de sistema básico de memoria

- Conexionado de memorias ROM y SRAM de 8 bits
  - Necesidad de actuar sobre la señal *ape* o la señal *ale*
  - Acceso a la palabra → solo se utilizan las señales *A[2]* en adelante
  - Las señales *A[1:0]* son utilizadas por la lógica de control para la selección de tamaños *byte* y halfword

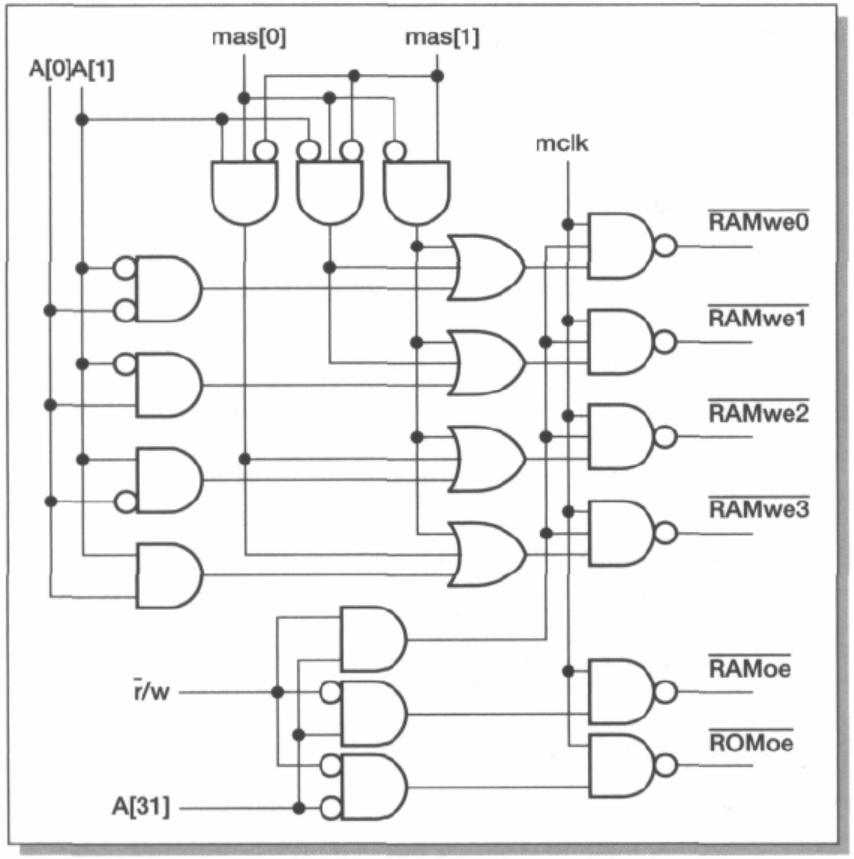


- No existe problema en usar memorias ROMs de 16 bits. En caso de las memorias RAM, éstas deben tener *byte enable*

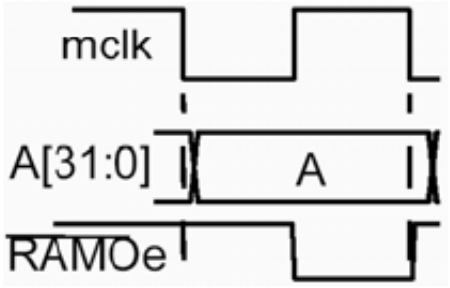
Gestión de memoria en el ARM: control de sistema básico de memoria

- Sistema de decodificación simple
- Controla la escritura a nivel de byte durante las operaciones de escritura

Tamaño	mas[1]	mas[0]
Byte	0	0
Halfword	0	1
Word	1	0
No usado	1	1



- Uso de *mclk → dbe*

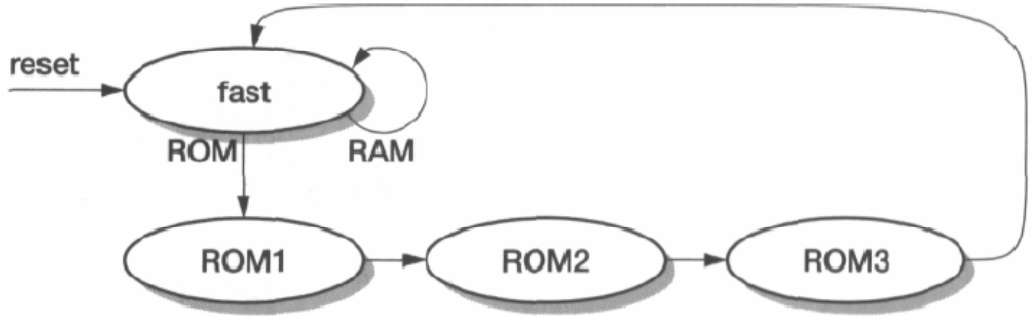


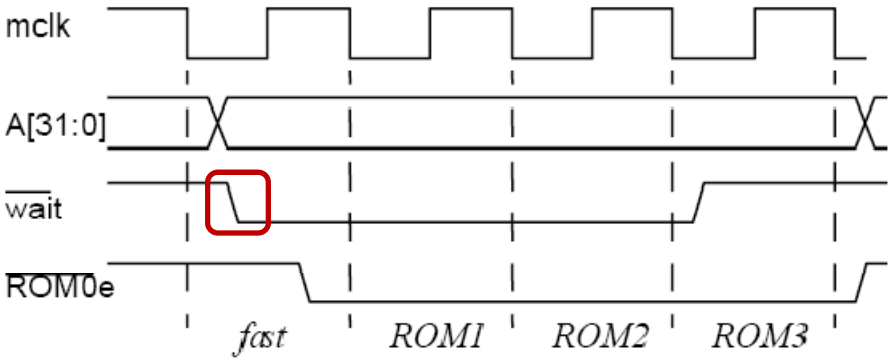
Gestión de memoria en el ARM: análisis temporal

- El sistema anterior asume que las salidas del ARM están estables al final del ciclo
  - Últimos ARMs → se consigue poniendo *ape* a nivel bajo
  - Primeros ARMs → se consigue igualando *ale* a *mclk*
- El sistema planteado no requiere el uso de señales de *mreq** o *seq*
- Análisis de requisitos temporales de las memorias
  - Memoria SRAM muy rápida
  - Memoria ROM más lenta
  - ¿Solución?
    - Disminuir la frecuencia de la señal de reloj → reducción de prestaciones
    - Uso de estados de espera (*wait states*) → solución más compleja, *mclk* se ajusta al tiempo de acceso de la memoria RAM

Gestión de memoria en el ARM: uso de ciclos de espera en el acceso a memoria ROM I

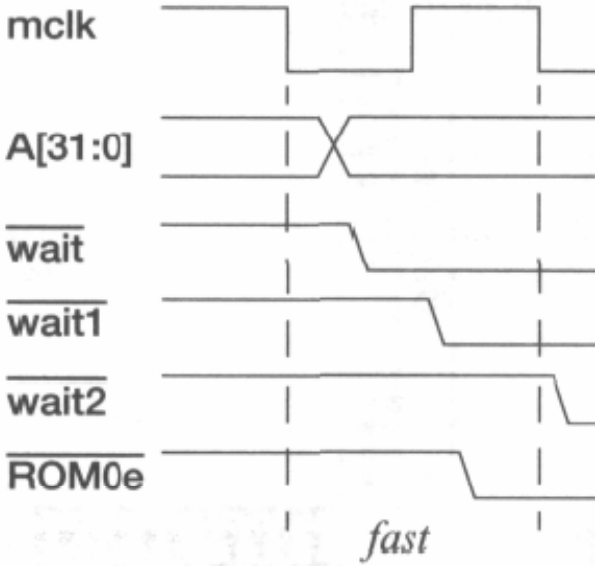
- El número de ciclos de espera para la memoria ROM viene determinado por la frecuencia de *mclk* y el *datasheet* de la memoria
  - Asumimos en este ejemplo un tiempo de acceso de 4 ciclos de reloj
  - La lógica de control debe incorporar la siguiente máquina de estado
  - Diagrama de tiempo

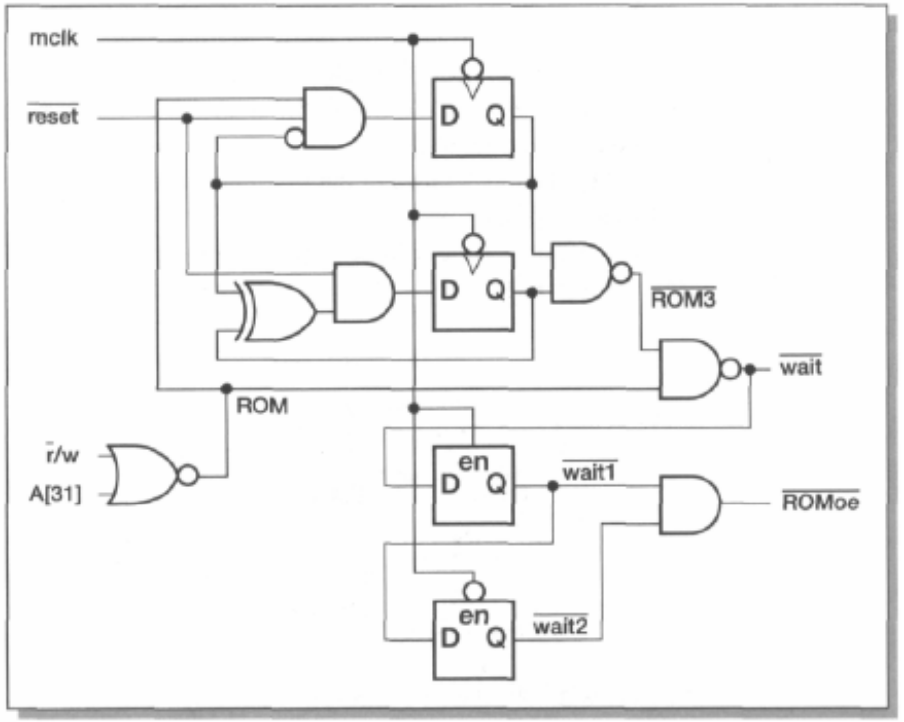




Gestión de memoria en el ARM: uso de ciclos de espera en el acceso a memoria ROM II

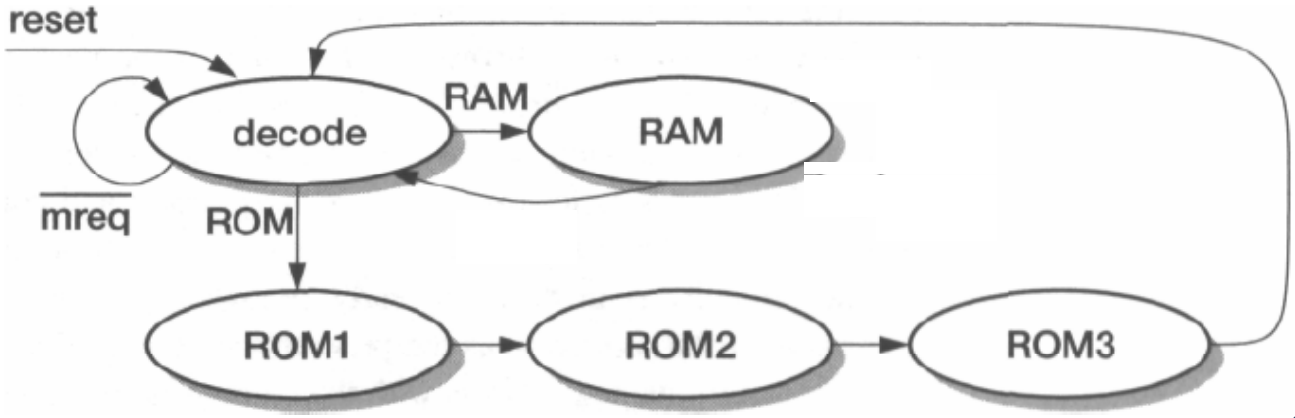
- Problemas de diseño
  - La señal *wait* debe generarse antes del flanco de subida de la señal *mclk* y no existe flanco de reloj que pueda usarse
  - Dificultad para generar la señal *ROMoe* libre de *glitches*
- Posible solución
  - Uso de un contador síncrono con dos flancos





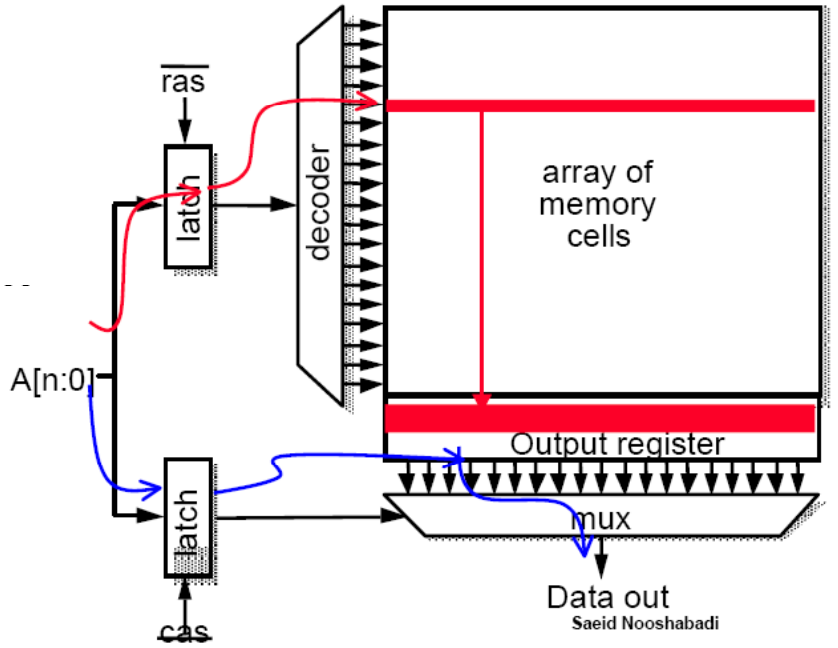
Gestión de memoria en el ARM: mejora de prestaciones

- Aumento de la frecuencia de funcionamiento
  - Permite operar internamente *(mreq* = 1)* a la máxima frecuencia
  - Es necesario la introducción de un ciclo de espera para acceso a la memoria RAM
- Nuevo esquema de la máquina de estados
  - La transición RAM indica acceso a memoria RAM con dirección desconocida → es necesaria una nueva decodificación de la dirección



Gestión de memoria en el ARM: gestión de DRAM I

- Uso de RAM dinámica → más barata, mayor capacidad, más lenta que la SRAM
- Funcionamiento
  - Bus de direcciones multi-plexado en *row and co-lumn address*
  - Codificación únicamente en selección de la fila
  - La siguiente columna es accesible sin necesidad de decodificación, dos a tres veces más rápida

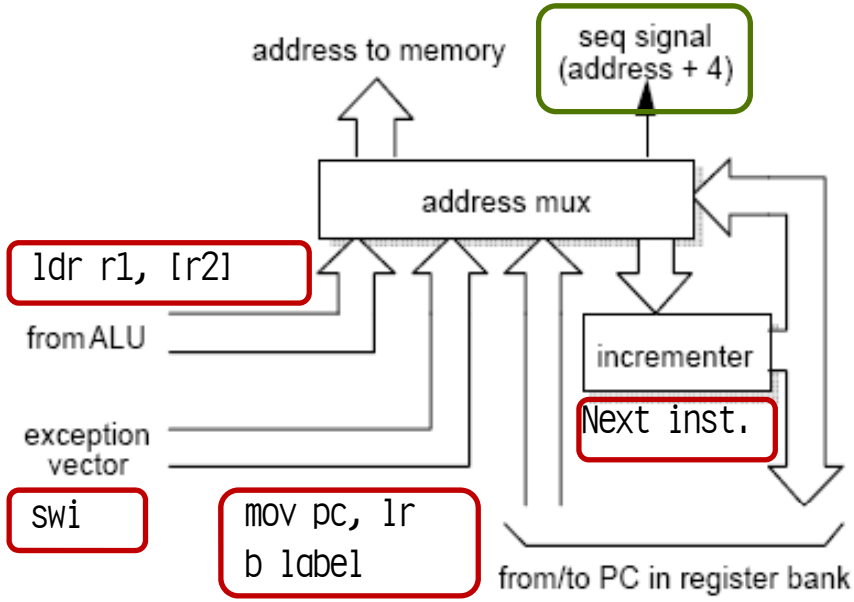


Gestión de memoria en el ARM: gestión de DRAM II

- El acceso a un dato en la misma fila es de dos a tres veces más rápido
  - El acceso mediante *cas** no activa la lógica de decodificación
  - Si el acceso es dentro de la misma columna, basta con activar la dirección correspondiente a *cas**
- Importancia
  - Se ha comprobado que el 75% de las direcciones generadas por un procesador son secuencias (principio de localidad espacial)
  - Si existiera una manera de conocer que el siguiente acceso a memoria es secuencia, podríamos acelerar la ejecución de un programa
  - Dificultad
    - Detectar **al comienzo** del ciclo que el acceso es dentro de la misa fila

Gestión de memoria en el ARM: solución al acceso secuencial

- La solución dada por ARM a este conflicto está en el diseño de su *address incrementer*
  - Esta lógica selecciona la siguiente dirección de un total de cuatro posibles fuentes
  - La generación de una dirección secuencial se indica con la señal *seq*

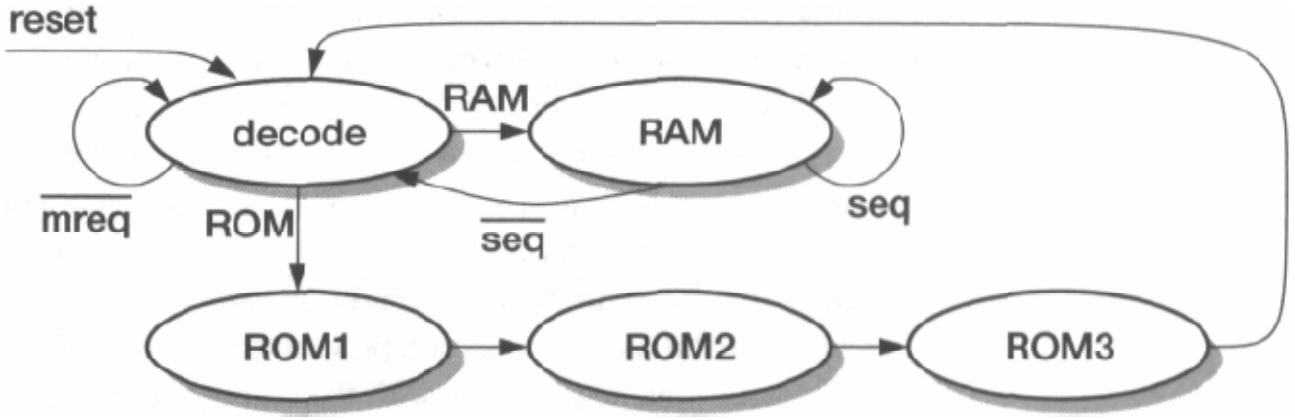


Gestión de memoria en el ARM: optimización del diagrama de estados

- Consideraciones:
  - Uso de la señal *mreq** y la señal *seq* para optimizar el acceso a memoria
  - Modificación de la máquina de estados teniendo en cuenta las siguientes acciones →

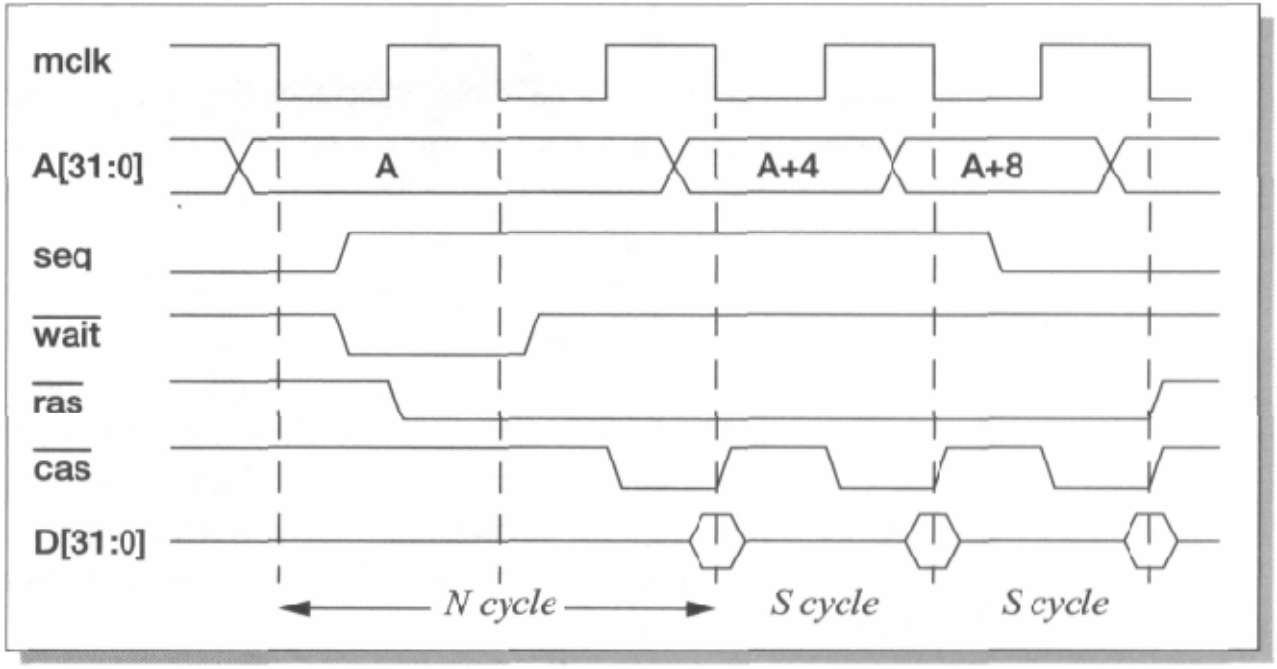
Señal	Acción
mreq* = 0	**Acceso a memoria**
mreq* = 1	**Operación interna**
***seq* = 0**	**Acceso no secuencial**
***seq* = 1**	**Acceso secuencial**

  - Nuevo diagrama de la máquina de estados



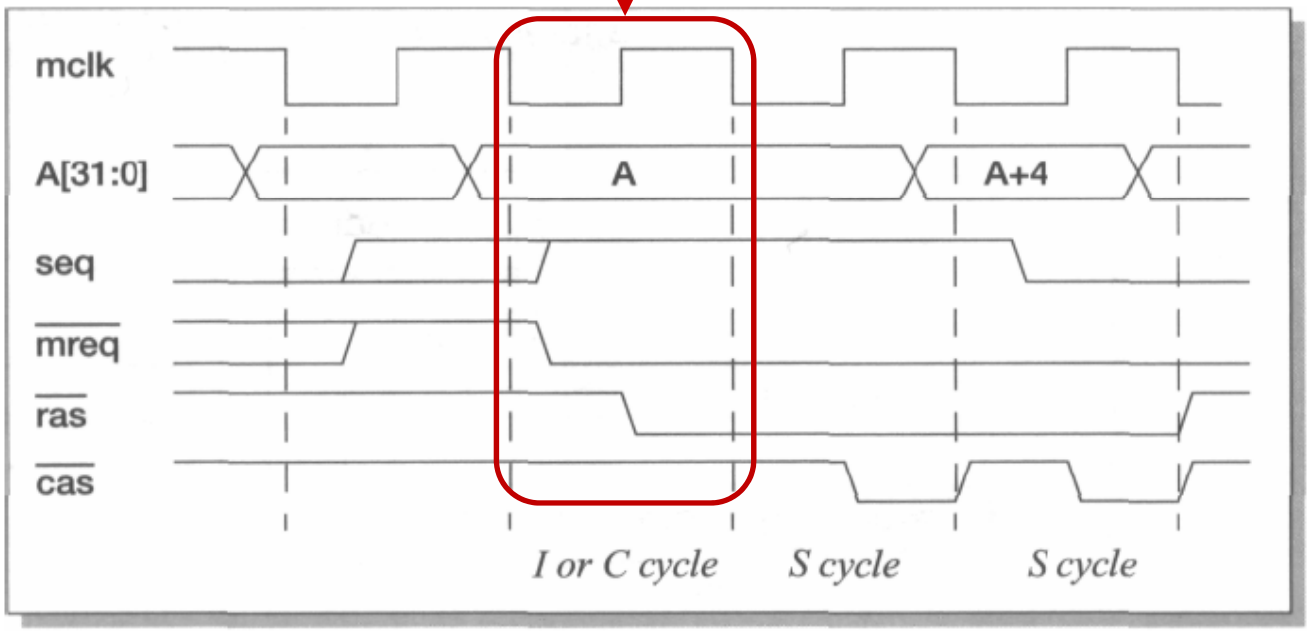
Gestión de memoria en el ARM: cronograma de acceso a RAM dinámica I

- A partir del siguiente cronograma, se observa que
  - El acceso contiene un ciclo no secuencial seguido de dos ciclos secuenciales
  - Durante el ciclo no secuencial es necesario mantener *ape* o *ale* a nivel alto para conseguir mantener la dirección



Gestión de memoria en el ARM: cronograma de acceso a RAM dinámica II

- Uso de la señal *seq* para optimizar el acceso a DRAM
  - En el caso de ciclos internos o de coprocesador, la señal *seq* indica que el ciclo de acceso usará la **misma** dirección que el ciclo anterior
  - La optimización se consigue empezando el ciclo de acceso a memoria durante el ciclo interno o de coprocesador



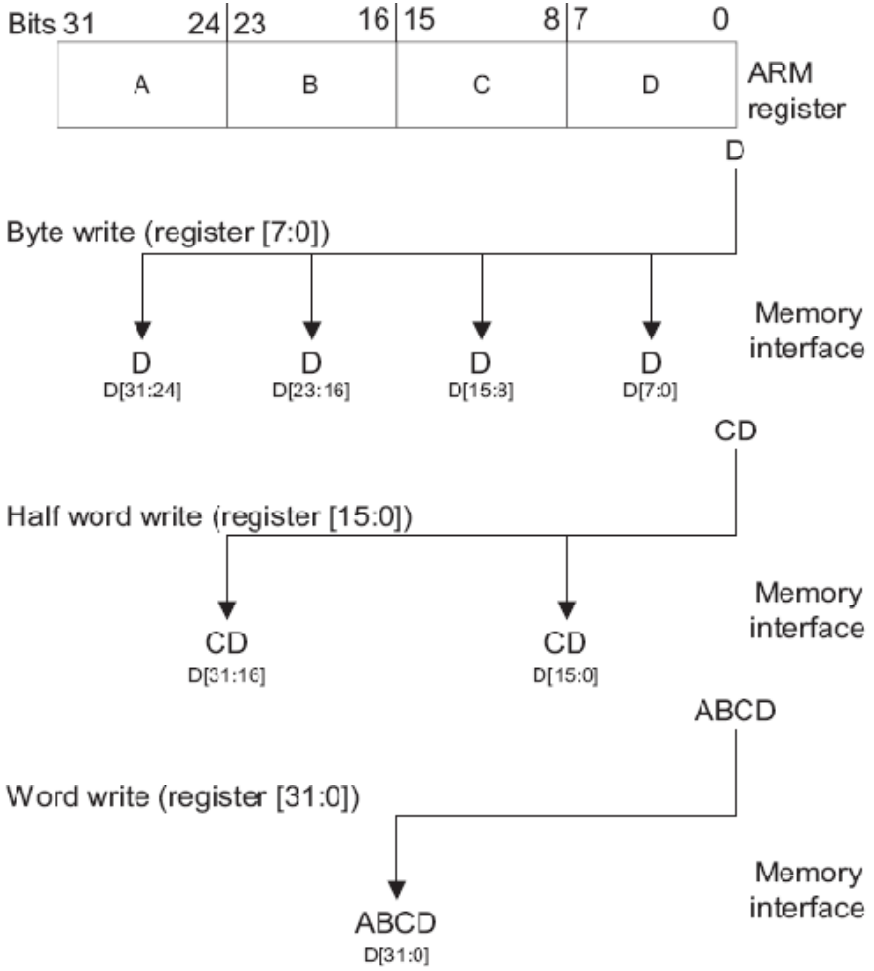
Gestión de memoria en el ARM: accesos a memoria I

- Accesos, en lectura, en tamaño *byte* y *halfword*
  - Se lee el bus completo y se extrae la parte significativa según los valores de las líneas *mas[1:0]*
  - El dato leído siempre se localiza en la parte baja del registro, independientemente de la línea activa
    - Ejemplo: Si se lee un byte con A[1:0] = 01 y *little endian*, ocurre ARM reg[31:8] ← 0x0 ; ARM reg[7:0] ← D[15:8]

Access type	MAS[1:0]	A[1:0]	Little-endian BIGEND = 0	Big-endian BIGEND = 1
Word	10	XX	**D[31:0]**	**D[31:0]**
Halfword	01	0X	**D[15:0]**	**D[31:16]**
	01	1X	**D[31:16]**	**D[15:0]**
Byte	00	00	**D[7:0]**	**D[31:24]**
	00	01	**D[15:8]**	**D[23:16]**
	00	10	**D[23:16]**	**D[15:8]**
	00	11	**D[31:24]**	**D[7:0]**

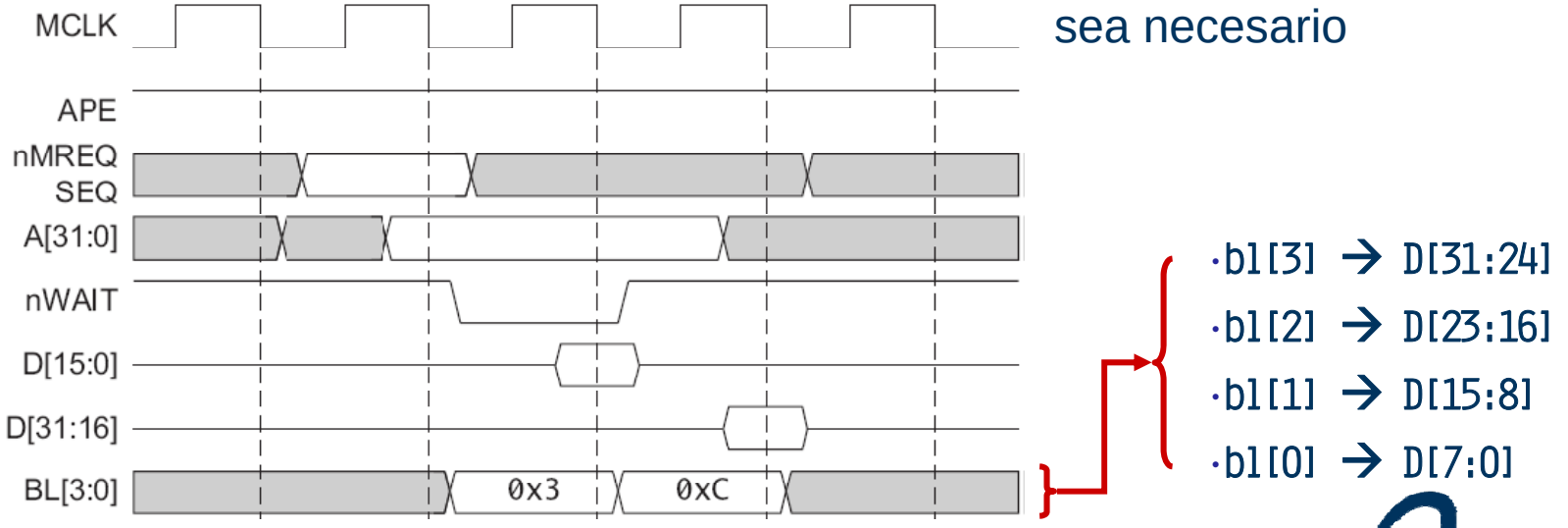
Gestión de memoria en el ARM: accesos a memoria II

- Accesos, en escritura, en tamaño *byte* y *halfword*
  - El dato se replica en el bus de dato, 4x *byte*, 2x *halfword*, 1x *word* y 1x *ABCD*
  - Es indispensable que la memoria disponga de señales propias para la escritura de *bytes* o *halwords*, más comúnmente denominadas *byte enable*



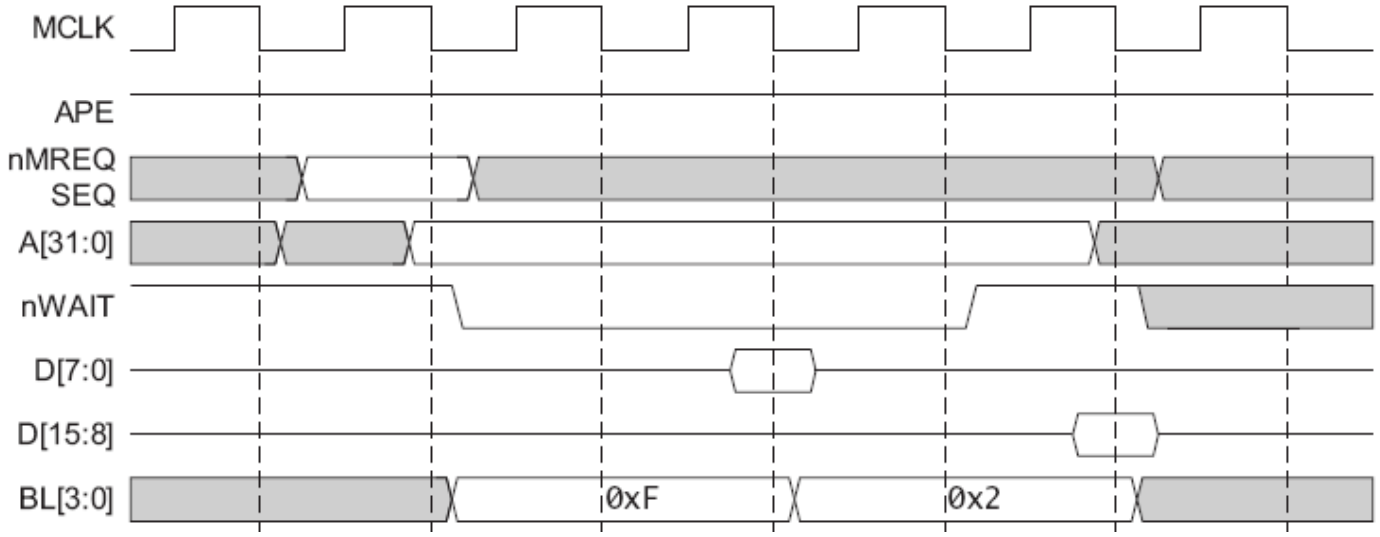
Gestión de memoria en el ARM: control de latches de datos I

- El ARM puede conectarse a sistemas de tamaño *byte* o *halfword*
  - Tanto los datos como las instrucciones pueden transferirse *byte* a *byte*
  - Las transferencias se controlan mediante las señales *bl[3:0]*
- Ejemplo de transferencia con ancho *halfword*
  - La transferencia necesita un total de dos ciclos
  - Es necesario el uso de la señal *wait** para extender el ciclo interno de reloj
  - El sistema puede utilizar tantos ciclos de espera como sea necesario



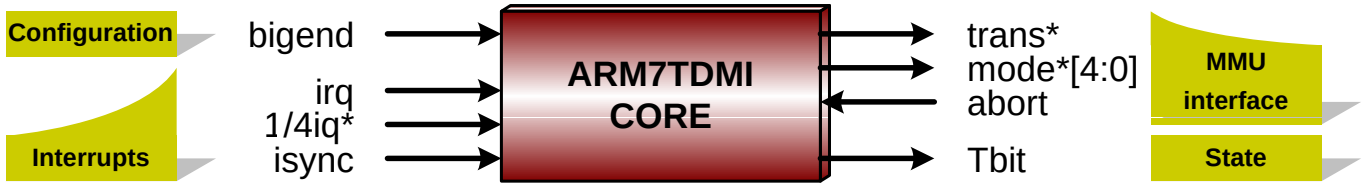
Gestión de memoria en el ARM: control de latches de datos II

- Ejemplo de acceso en tamaño *halfword* a sistemas de ancho *byte*
  - Se ha supuesto el uso de una memoria lenta con accesos a memoria de duración dos ciclos de reloj → necesidad de activar la señal *wait**
  - Si bien durante el primer ciclo *bl[3:0]* vale 0xF, únicamente es válido el dato cargado desde *D[7:0]*



Interfaz hardware: líneas de interfaz → *Configuration, State, Interrupts, MMU interface*

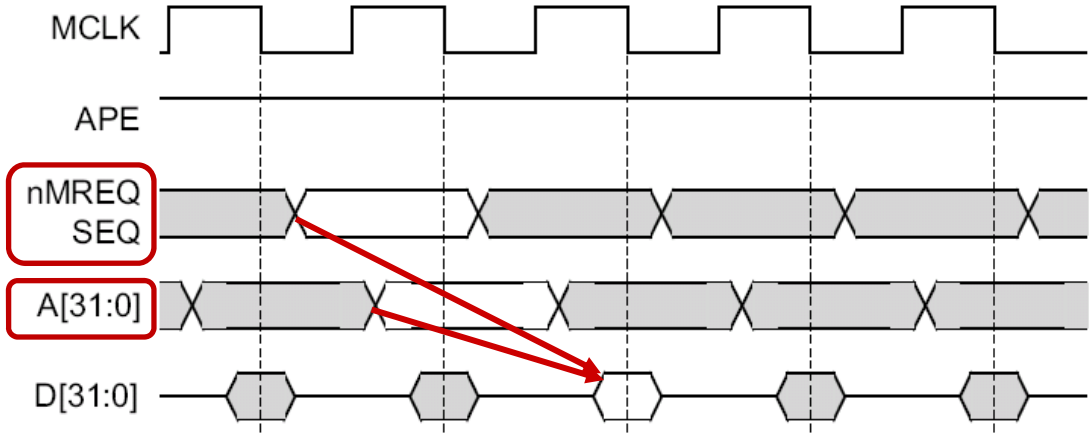
- Líneas de interfaz *Configuration*
  - *bigend* → entrada de configu-ración para la distribución de los bytes en memoria
- Líneas de interfaz *State*
  - *Tbit* → indica el tipo de instrucción que se está ejecutando (*Tbit=1*, modo *Thumb*)
- Líneas de interfaz *Interrupts*
  - *fiq** → solicitud de interrupción rápida
  - *irq** → solicitud de interrupción
  - *isync* → las solicitudes de interrupción se sincronizan internamente en la CPU. Si una interrupción externa está sincronizada con *mclk* se debe indicar poniendo esta entrada a nivel alto
- Líneas de interfaz *MMU interface*
  - *trans** → indica modo del procesador, usuario (“0”) o privilegiado (“1”)
  - *abort* → entrada que indica que no se puede completar el acceso a memoria
  - *trans*[4:0]* → CPSR[4:0]*



Ciclos de bus: ciclo simple de memoria

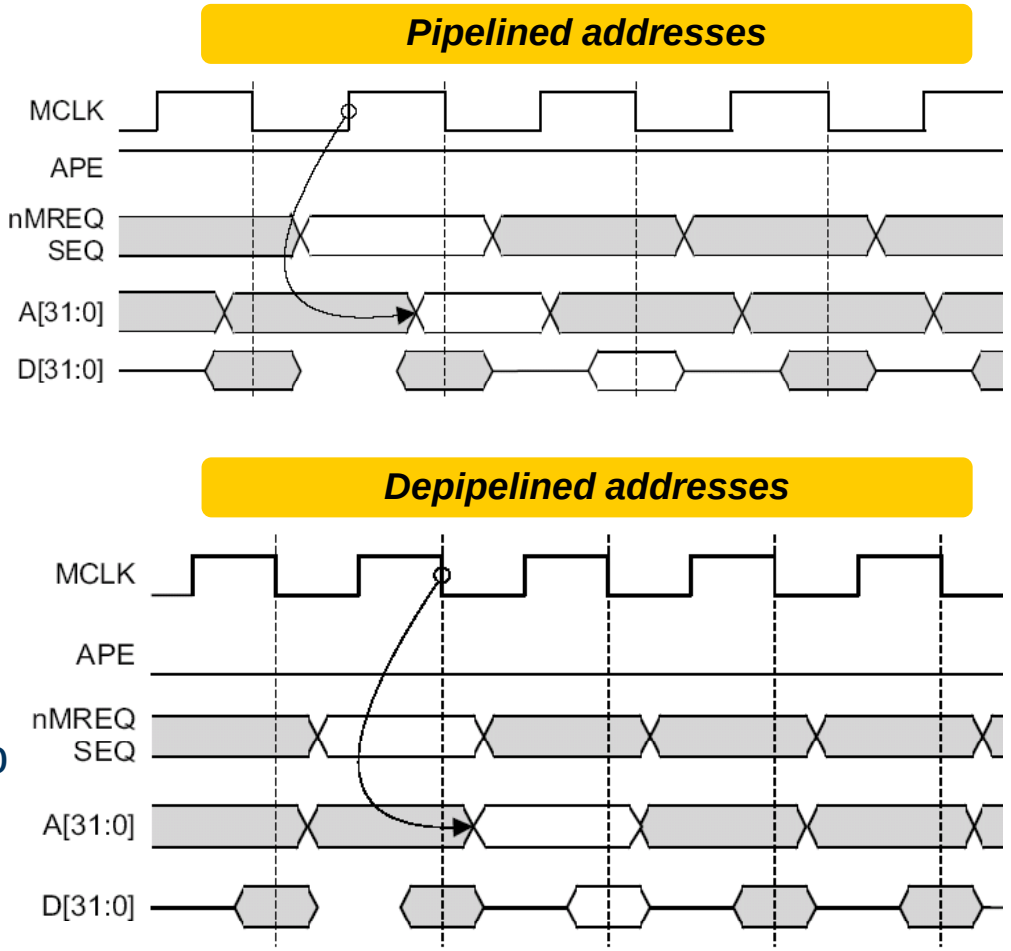
Ciclo simple de memoria

- El ciclo de reloj en el ARM comienza en el flanco de bajada
- Las señales de control aparecen en el ciclo de bus anterior al que se refieren
- Las líneas de dirección aparecen medio ciclo de reloj antes del ciclo de bus a que se refieren
- El ARM suele usar la señal *ape* a nivel alto
  - *Address pipeline enable* → establece si el acceso a memoria se realiza con *pipeline* o no



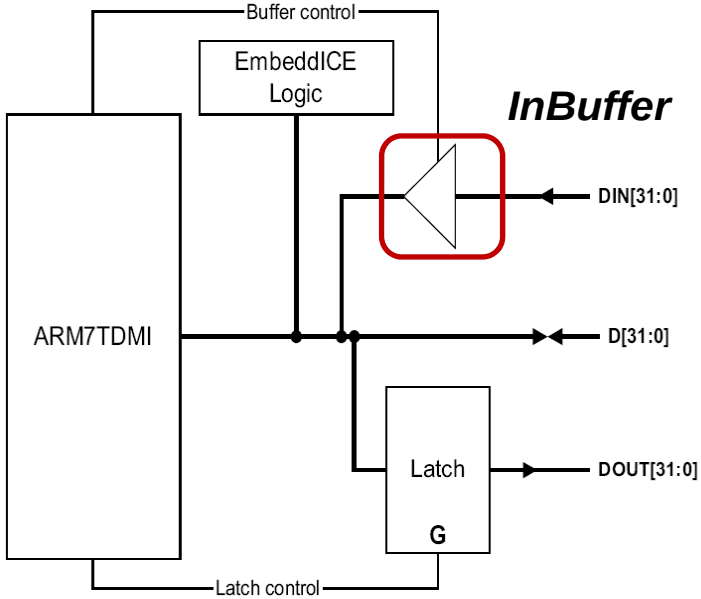
Ciclos de bus: *pipelined and depipelined addresses*

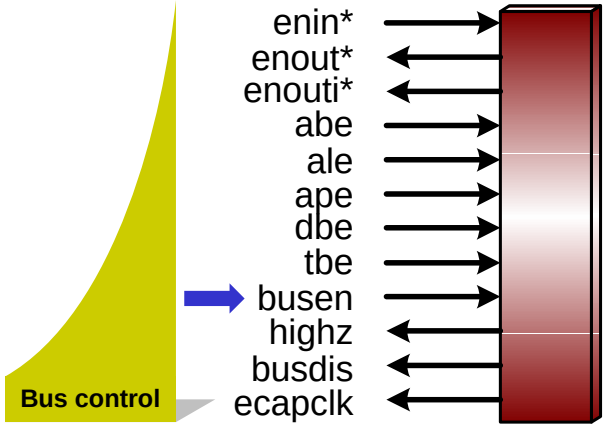
- La transferencia siempre se realiza en flanco de bajada de la señal de reloj *mclk*
- La señal *ape* señaliza el comportamiento de la dirección
  - *Pipelined*: la dirección de acceso aparece un ciclo antes del ciclo de bus al que hace referencia (DRAM)
  - *Depipelined*: la dirección aparece en el mismo ciclo (SRAM y ROM)



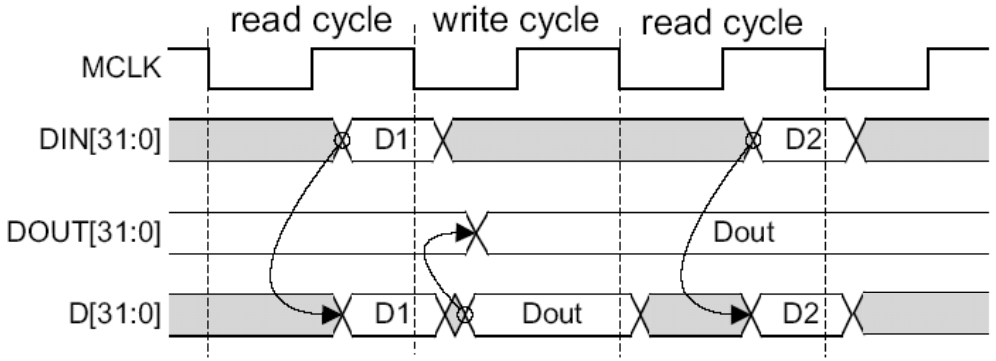
Ciclos de bus: *bidirectional and unidirectional bus* I

- El ARM puede enviar y recibir datos mediante el bus *D* o *Din* y *Dout*
  - La configuración se realiza mediante la entrada *busen*
    - "0"→ habilita el bus *D*
    - "1" → habilita los buses *Din* y *Dout*. En este caso el bus *D* debe permanecer desconectado
  - Si *busen* está a "0", *InBuffer* queda deshabilitado y *DOUT[31:0]* toma el valor 0x0000_0000



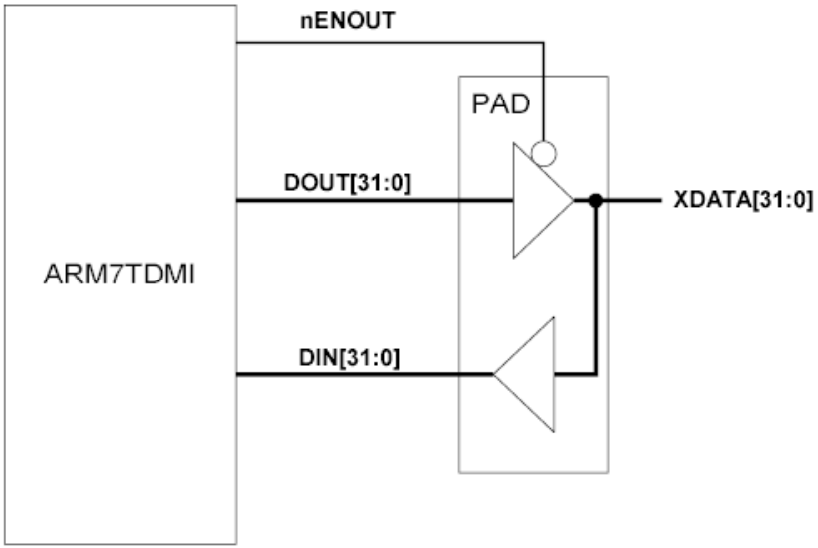


External bus arrangement

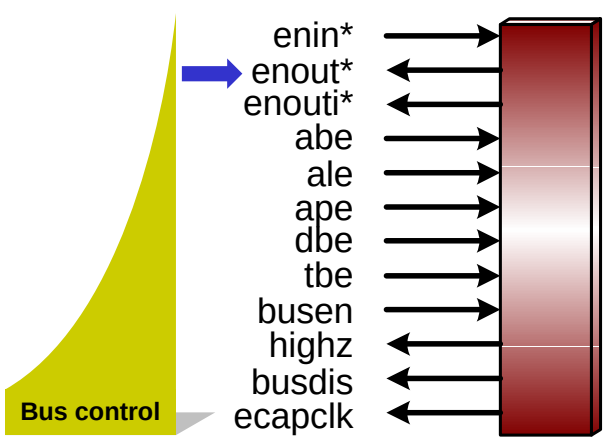


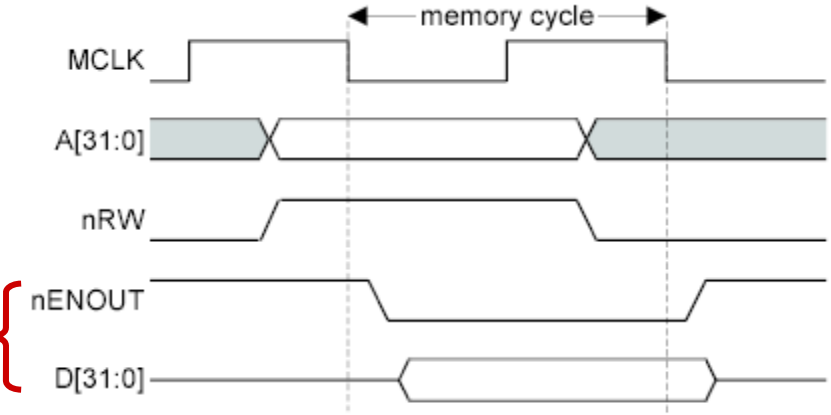
Ciclos de bus: *bidirectional and unidirectional bus* II

- Es posible conectar un bus externo bidireccional a los buses *DIN* y *DOUT*
- Esta posibilidad se consigue mediante la señal *enout**
- Cuando *enout** se activa, indica que el ARM toma el control del bus bidireccional
- La señal *enout** debe mantener-se habilitando el dato



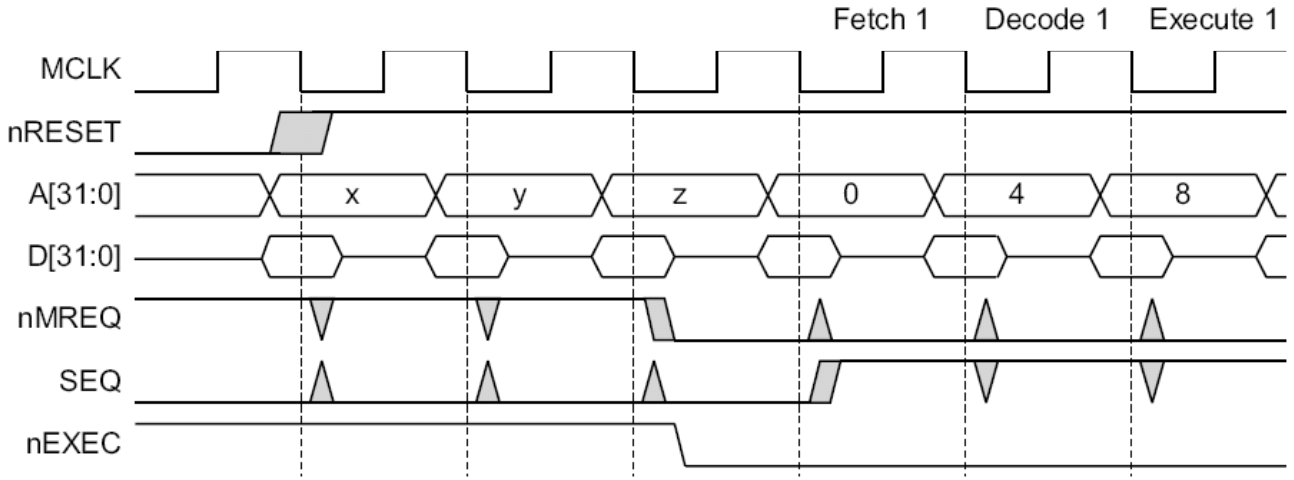
External bidirectional data bus





Ciclos de bus: secuencia de reset tras un *power up*

- La señal de reset debe activarse, al menos, durante dos ciclos de reloj
- Tras desactivarse, el sistema aún opera en modo interno antes de empezar el ciclo de *fetch* de la primera instrucción
  - La primera instrucción corresponde al contenido del vector de reset en la dirección 0x0000_0000



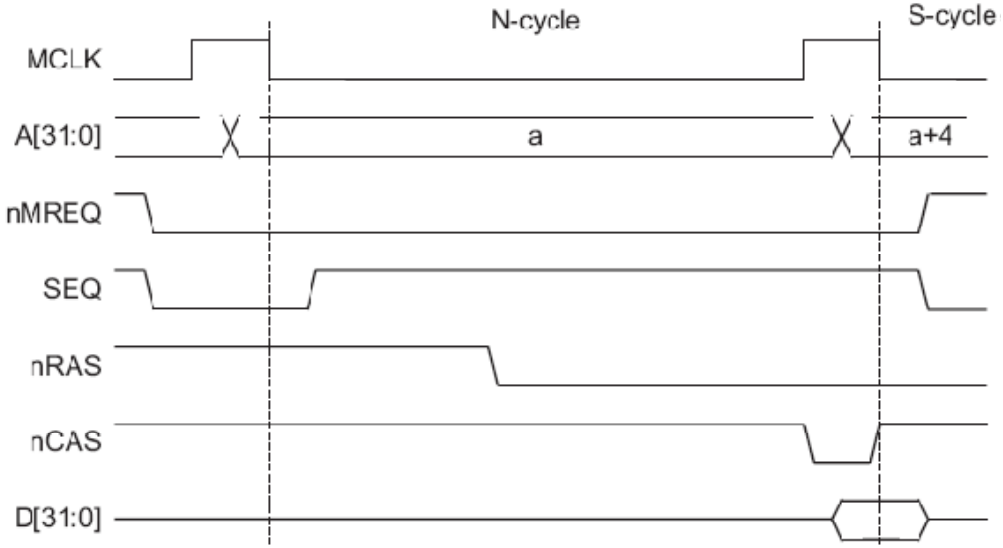
Ciclos de bus: tipo de ciclos

- El ARM dispone de cuatro tipos de ciclos
  - **No secuencial**: se realiza una transferencia a o desde una dirección no relacionada con la dirección del ciclo precedente
  - **Secuencial**: se realiza una transferencia a o desde una dirección que es una palabra o media palabra mayor que la anterior
  - **Interno**: el ARM está realizando operaciones internas y no se requiere acceso a memoria
  - **Transferencia a registro de coprocesador**: el ARM utiliza el bus de datos para comunicarse con el coprocesador, si bien no se requiere acceso a memoria

*mreq**	*seq*	Ciclo	Descripción
0	0	*N-cycle*	Ciclo no secuencial
0	1	*S-cycle*	Ciclo secuencial
1	0	*I-cycle*	Ciclo interno
1	1	*C-cycle*	Ciclo de transf. a reg. de cop.

Ciclos de bus: ciclo no secuencial

- *N-cycle*
  - La transferencia con memoria no está relacionada con la del ciclo precedente
  - La información del ciclo aparece siempre en el ciclo precedente (*mreq** y *seq*)
  - Es posible realizar el ajuste de reloj con la señal *wait** para adaptar el ciclo de bus al tiempo de acceso de la memoria
  - Ejemplo:
    - Conexionado con DRAM

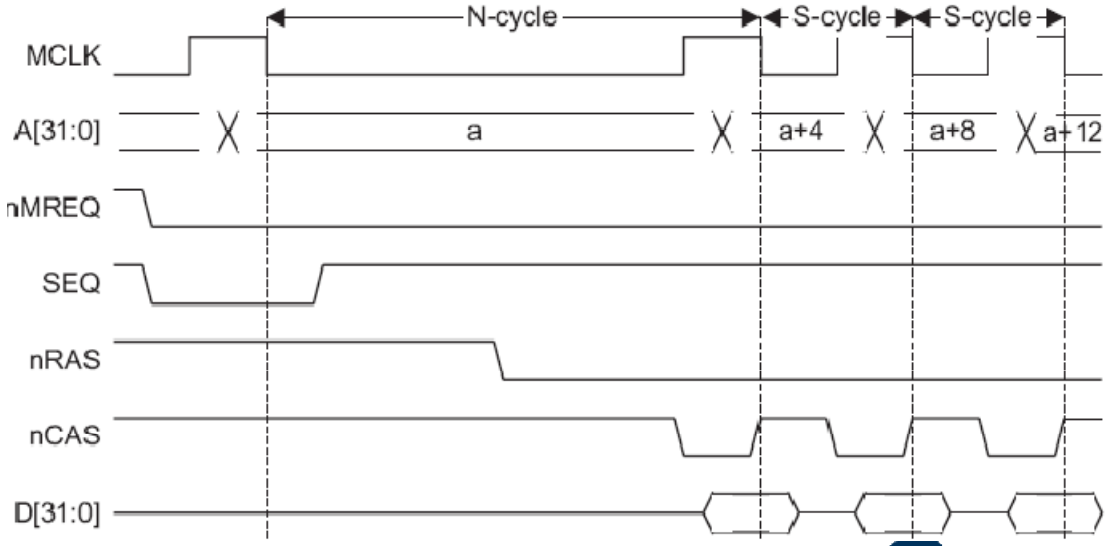


Ciclos de bus: ciclo secuencial

S-cycle

- Posibilita la transferencia en modo ráfaga, ejemplo DRAM, lo que permite acelerar el acceso a memoria
- Ráfagas de tamaño *word* (+4) o *half-word* (+2) y comienzan siempre con un *N-cycle* y continúa con *S-cycles*

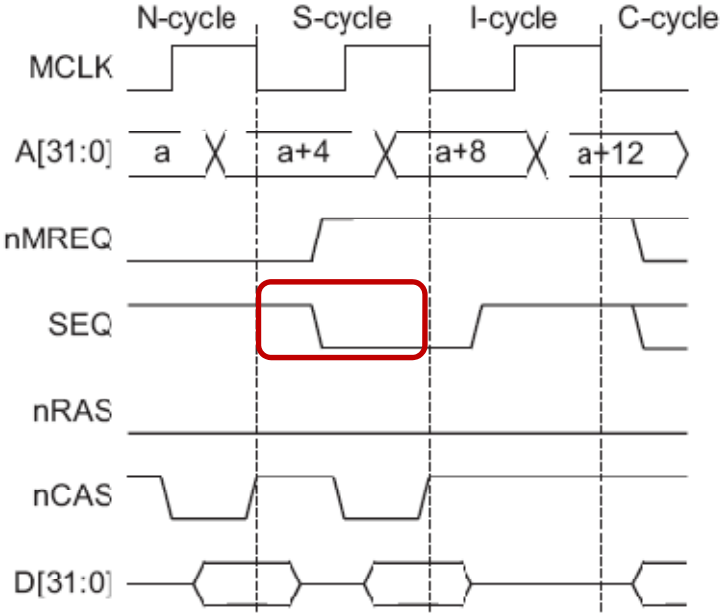
Type	Add. Inc.	*Cause*
Word read	***N-cycle***	***Code fetch, LDM instr.***
Word write	***S-cycle***	***STM instr.***
Halfword read	***I-cycle***	***Thumb code fetch***



Ciclos de bus: ciclo interno

I-cycle

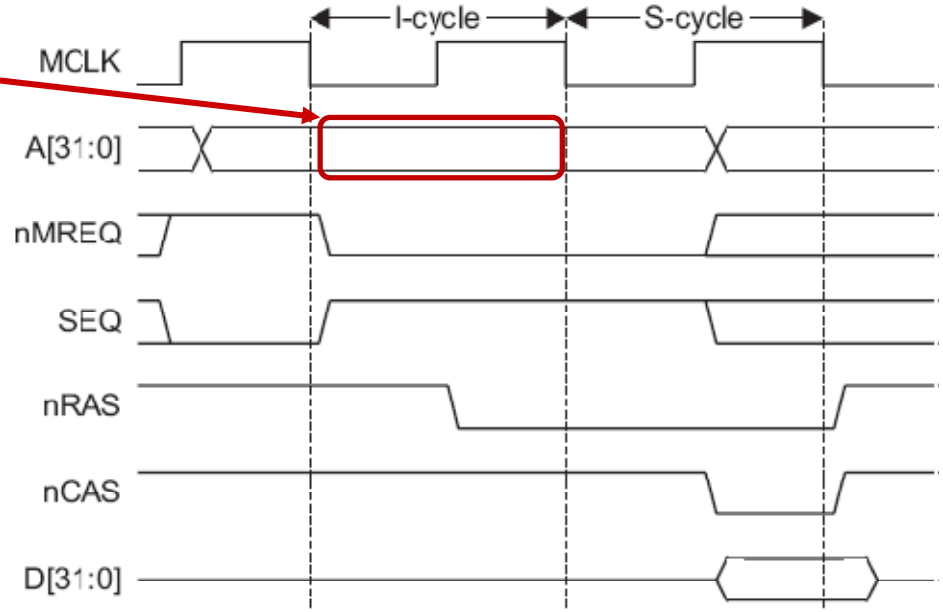
- El ARM realiza una operación interna y no requiere acceso a memoria
- El controlador de memoria no debe permitir ninguna operación
- La señal *seq* señaliza que no se trata de un acceso secuencial por lo que se debe activar la decodificación de la dirección
- En el ejemplo, el siguiente ciclo realiza un acceso secuencial
- De ser posible, el procesador genera durante el ciclo interno, la dirección para el siguiente ciclo, lo que acelera la ejecución de un programa



Ciclos de bus: fusión de ciclo interno y secuencial

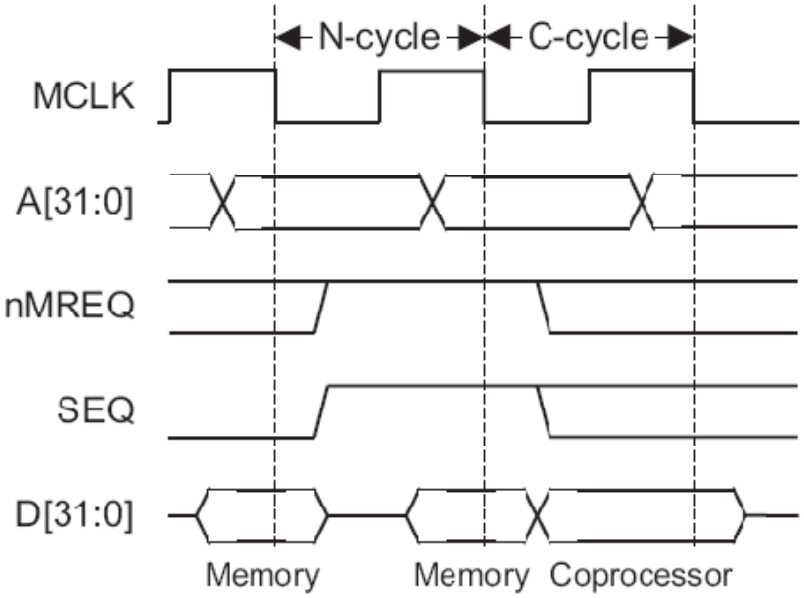
IS-cycle

- El ARM trata siempre de optimizar el uso de los ciclos internos, habilitando un tiempo extra para la decodificación de la dirección de memoria
- Cuando esto ocurre, la dirección de acceso se mantiene durante todo el ciclo interno, permitiendo al controlador de memoria la decodificación de la dirección
- Durante el ciclo interno el controlador no debe realizar ningún acceso a memoria
- El siguiente ciclo se trata siempre de un ciclo secuencial



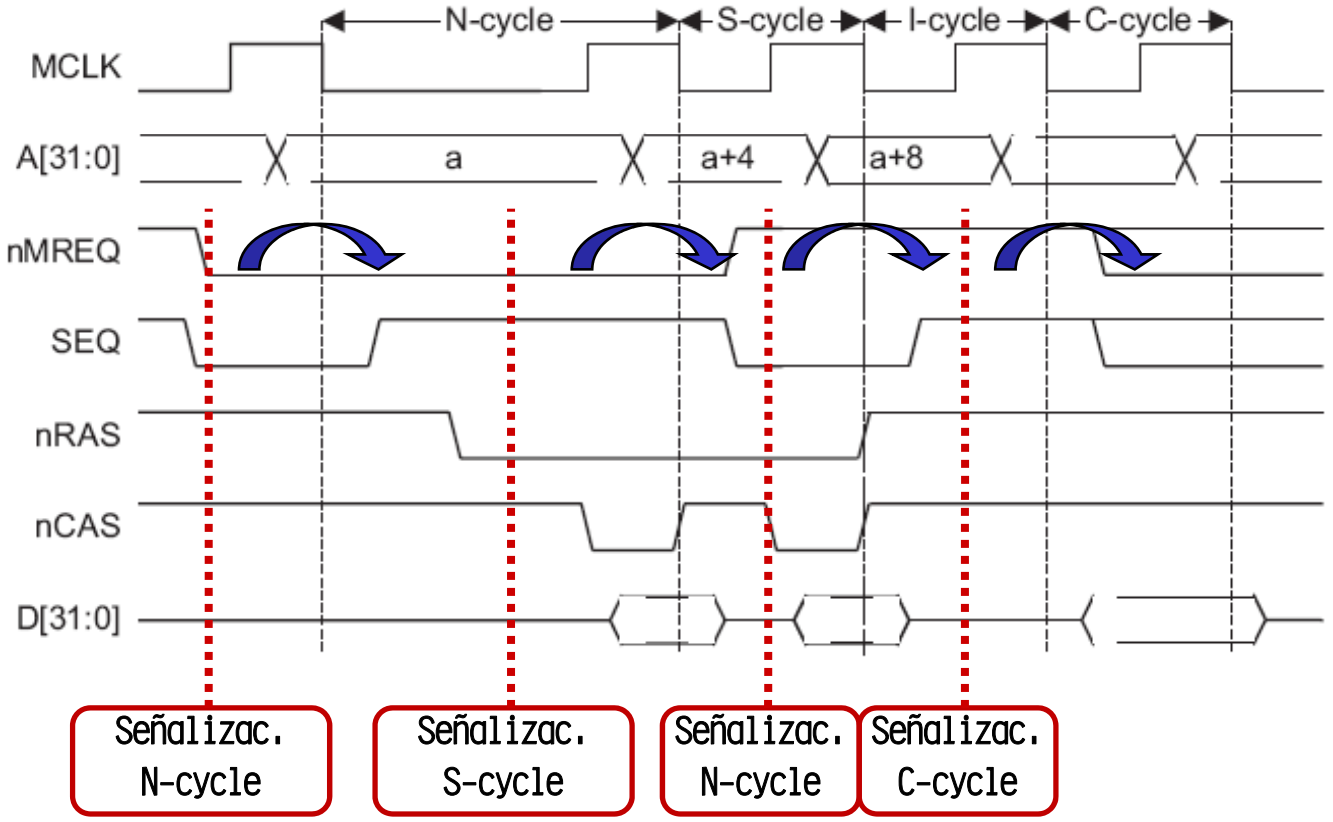
Ciclos de bus: ciclo de transferencia con coprocesador

- *C-cycle*
  - Este ciclo se utiliza para la transferencia de información del ARM con otros procesadores
  - Las transferencias se realizan utilizando el bus de datos del procesador, por este motivo el controlador de memoria no debe permitir el acceso a memoria durante el ciclo



Ciclos de bus: ejemplo de ciclos combinados

- El siguiente cronograma representa un ejemplo de ciclos combinados

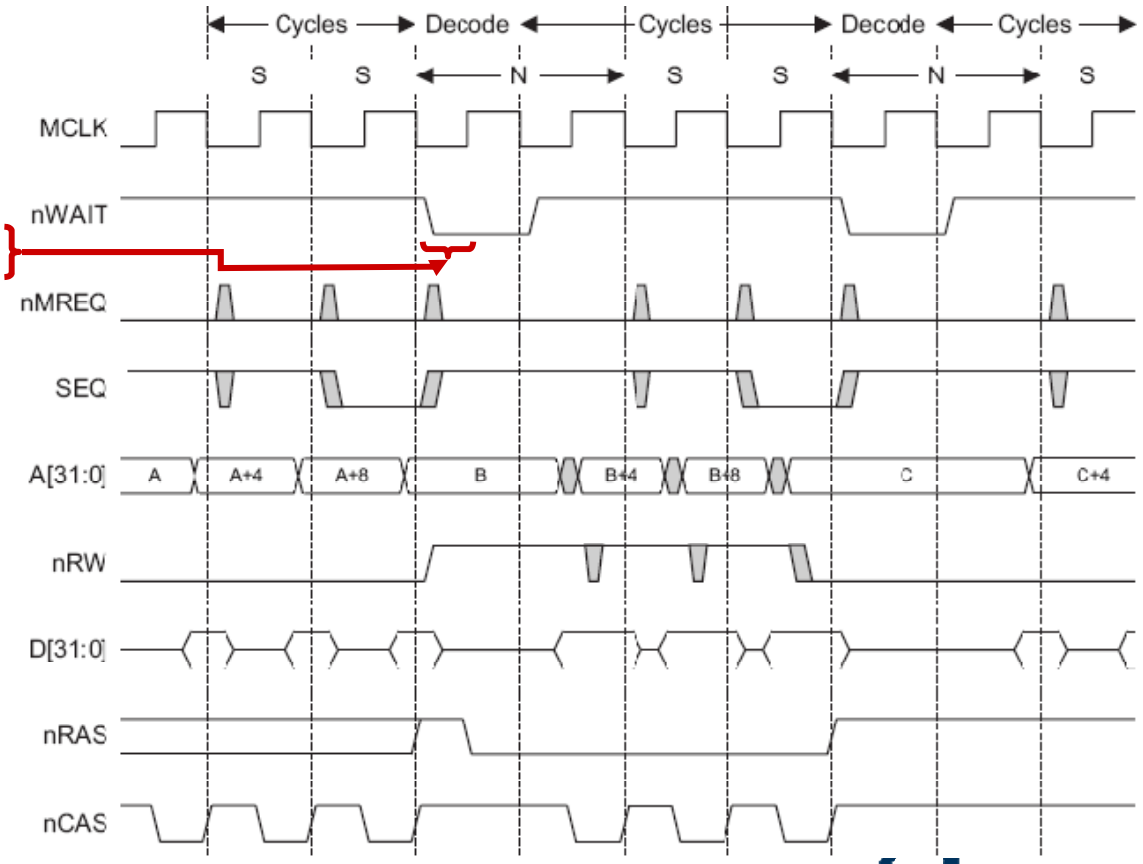


Ciclos de bus: uso de *wait** para controlar los ciclos de bus

Ciclos de reloj vs. Ciclos de bus

- *wait** se utiliza para alargar la duración del ciclo de bus
  - El ciclo de bus finaliza cuando en el flanco de subida de *mclk* la señal *wait** está a nivel alto

- *wait** solo puede modificarse durante la fase a “0” de *mclk*
- A la hora de diseñar un controlador de memoria, hay que muestrear las señales *seq* y *mreq** cuando *wait* está a nivel alto



Resumen de líneas de interfaz 1/#12

A[31:0] Addresses	O	This is the 32-bit address bus. **ALE**, **ABE**, and **APE** are used to control when the address bus is valid.
ABE Address bus enable	IC	The address bus drivers are disabled when this is LOW, putting the address bus into a high impedance state. This also controls the LOCK, **MAS[1:0]**, **nRW**, **nOPC**, and **nTRANS** signals in the same way. **ABE** must be tied HIGH if there is no system requirement to disable the address drivers.
ABORT Memory abort	IC	The memory system uses this signal to tell the processor that a requested access is not allowed.
ALE Address latch enable	IC	This signal is provided for backwards compatibility with older ARM processors. For new designs, if address retiming is required, ARM Limited recommends the use of **APE**, and for **ALE** to be connected HIGH. The address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nOPC**, and **nTRANS** signals are latched when this is held LOW. This enables these address signals to be held valid for the complete duration of a memory access cycle. For example, when interfacing to ROM, the address must be valid until after the data has been read.
APE Address pipeline enable	IC	Selects whether the address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** signals operate in pipelined (**APE** is HIGH) or depipelined mode (**APE** is LOW). Pipelined mode is particularly useful for DRAM systems, where it is desirable to provide the address to the memory as early as possible, to allow longer periods for address decoding and the generation of DRAM control signals. In this mode, the address bus does not remain valid to the end of the memory cycle. Depipelined mode can be useful for SRAM and ROM access. Here the address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** signals must be kept stable throughout the complete memory cycle. However, this does not provide optimum performance. See *Address timing* on page 3-14 for details of this timing.

Resumen de líneas de interfaz 2/#12

BIGEND Big endian configuration	IC	Selects how the processor treats bytes in memory: • HIGH for big-endian format • LOW for little-endian format.
BL[3:0] Byte latch control	IC	The values on the data bus are latched on the falling edge of **MCLK** when these signals are HIGH. For most designs these signals must be tied HIGH.
BREAKPT Breakpoint	IC	A conditional request for the processor to enter debug state is made by placing this signal HIGH. If the memory access at that time is an instruction fetch, the processor enters debug state only if the instruction reaches the execution stage of the pipeline. If the memory access is for data, the processor enters debug state after the current instruction completes execution. This enables extension of the internal breakpoints provided by the EmbeddedICE-RT logic. See *Behavior of the program counter in debug state* on page B-30 for details on the use of this signal.
BUSDIS Bus disable	O	When INTEST is selected on scan chain 0, 4, or 8 this is HIGH. It can be used to disable external logic driving onto the bidirectional data bus during scan testing. This signal changes after the falling edge of **TCK**.
BUSEN Data bus configuration	IC	A static configuration signal that selects whether the bidirectional data bus (**D[31:0]**) or the unidirectional data busses (**DIN[31:0]** and **DOUT[31:0]**) are used for transfer of data between the processor and memory. When **BUSEN** is LOW, **D[31:0]** is used; **DOUT[31:0]** is driven to a value of zero, and **DIN[31:0]** is ignored, and must be tied LOW. When **BUSEN** is HIGH, **DIN[31:0]** and **DOUT[31:0]** are used; **D[31:0]** is ignored and must be left unconnected. See Chapter 3 *Memory Interface* for details on the use of this signal.

Resumen de líneas de interfaz 3/#12

COMMRX Communications channel receive	O	When the communications channel receive buffer is full this is HIGH. This signal changes after the rising edge of **MCLK**. See *Debug Communications Channel* on page 5-17 for more information.
COMMTX Communications channel transmit	O	When the communications channel transmit buffer is empty this is HIGH. This signal changes after the rising edge of **MCLK**. See *Debug Communications Channel* on page 5-17 for more information.
CPA Coprocessor absent	IC	Placed LOW by the coprocessor if it is capable of performing the operation requested by the processor.
CPB Coprocessor busy	IC	Placed LOW by the coprocessor when it is ready to start the operation requested by the processor. It is sampled by the processor when **MCLK** goes HIGH in each cycle in which **nCPI** is LOW.
D[31:0] Data bus	IC O	Used for data transfers between the processor and external memory. During read cycles input data must be valid on the falling edge of **MCLK**. During write cycles output data remains valid until after the falling edge of **MCLK**. This bus is always driven except during read cycles, irrespective of the value of **BUSEN**. Consequently it must be left unconnected if using the unidirectional data buses. See Chapter 3 *Memory Interface*.
DBE Data bus enable	IC	Must be HIGH for data to appear on either the bidirectional or unidirectional data output bus. When LOW the bidirectional data bus is placed into a high impedance state and data output is prevented on the unidirectional data output bus. It can be used for test purposes or in shared bus systems.

Resumen de líneas de interfaz 4/#12

DBGACK Debug acknowledge	O	When the processor is in a debug state this is HIGH.
DBGEN Debug enable	IC	A static configuration signal that disables the debug features of the processor when held LOW. This signal must be HIGH to enable the EmbeddedICE-RT logic to function.
DBGRQ Debug request	IC	This is a level-sensitive input, that when HIGH causes ARM7TDMI core to enter debug state after executing the current instruction. This enables external hardware to force the ARM7TDMI core into debug state, in addition to the debugging features provided by the EmbeddedICE-RT logic. See Appendix B *Debug in Depth*.
DBGRQI Internal debug request	O	This is the logical OR of **DBGRQ** and bit [1] of the debug control register.
DIN[31:0] Data input bus	IC	Unidirectional bus used to transfer instructions and data from the memory to the processor. This bus is only used when **BUSEN** is HIGH. If unused then it must be tied LOW. This bus is sampled during read cycles on the falling edge of **MCLK**.
DOUT[31:0] Data output bus	O	Unidirectional bus used to transfer data from the processor to the memory system. This bus is only used when **BUSEN** is HIGH. Otherwise it is driven to a value of zero. During write cycles the output data becomes valid while **MCLK** is LOW, and remains valid until after the falling edge of **MCLK**.

Resumen de líneas de interfaz 5/#12

DRIVEBS Boundary scan cell enable	O	Controls the multiplexors in the scan cells of an external boundary-scan chain. This must be left unconnected, if an external boundary-scan chain is not connected.
ECAPCLK EXTEST capture clock	O	Only used on the ARM7TDMI test chip, and must otherwise be left unconnected.
ECAPCLKBS EXTEST capture clock for boundary-scan	O	Used to capture the device inputs of an external boundary-scan chain during EXTEST. When scan chain 3 is selected, the current instruction is EXTEST and the TAP controller state machine is in the CAPTURE- DR state, then this signal is a pulse equal in width to **TCK2**. This must be left unconnected, if an external boundary-scan chain is not connected.
ECLK External clock output	O	In normal operation, this is simply **MCLK**, optionally stretched with **nWAIT**, exported from the core. When the core is being debugged, this is **DCLK**, which is generated internally from **TCK**.
EXTERN0 External input 0	IC	This is connected to the EmbeddedICE-RT logic and enables breakpoints and watchpoints to be dependent on an external condition.
EXTERN1 External input 1	IC	This is connected to the EmbeddedICE-RT logic and enables breakpoints and watchpoints to be dependent on an external condition.
HIGHZ High impedance	O	When the HIGHZ instruction has been loaded into the TAP controller this signal is HIGH. See Appendix B *Debug in Depth* for details.

Resumen de líneas de interfaz 6/#12

ICAPCLKBS INTEST capture clock	O	This is used to capture the device outputs in an external boundary-scan chain during INTEST. This must be left unconnected, if an external boundary-scan chain is not connected.
INSTRVALID Instruction valid	O	Indicates that the instruction in the Execute stage of the pipeline was valid and has been executed (unless it failed its conditions codes).
IR[3:0] TAP controller instruction register	O	Reflects the current instruction loaded into the TAP controller instruction register. These bits change on the falling edge of **TCK** when the state machine is in the UPDATE-IR state. The instruction encoding is described in *Public instructions* on page B-9.
ISYNC Synchronous interrupts	IC	Set this HIGH if **nIRQ** and **nFIQ** are synchronous to the processor clock. Set it LOW for asynchronous interrupts.
LOCK Locked operation	O	When the processor is performing a locked memory access this is HIGH. This is used to prevent the memory controller allowing another device to access the memory. It is active only during the data swap (SWP) instruction. This is one of the signals controlled by **APE**, **ALE** and **ABE**.
MAS[1:0] Memory access size	O	Used to indicate to the memory system the size of data transfer (byte, halfword or word) required for both read and write cycles, become valid before the falling edge of **MCLK** and remain valid until the rising edge of **MCLK** during the memory cycle. The binary values 00, 01, and 10 represent byte, halfword, and word respectively (11 is reserved). This is one of the signals controlled by **APE**, **ALE**, and **ABE**.

Resumen de líneas de interfaz 7/#12

MCLK Memory clock input	IC	This is the main clock for all memory accesses and processor operations. The clock speed can be reduced to enable access to slow peripherals or memory. Alternatively, the **nWAIT** can be used with a free-running **MCLK** to achieve the same effect.
nCPI Not coprocessor instruction	O	LOW when a coprocessor instruction is processed. The processor then waits for a response from the coprocessor on the **CPA** and **CPB** lines. If **CPA** is HIGH when **MCLK** rises after a request has been initiated by the processor, then the coprocessor handshake is aborted, and the processor enters the undefined instruction trap. If **CPA** is LOW at this time, then the processor enters a busy-wait period until **CPB** goes LOW before completing the coprocessor handshake.
nENIN NOT enable input	IC	This must be LOW for the data bus to be driven during write cycles. Can be used in conjunction with **nENOUT** to control the data bus during write cycles. See Chapter 3 *Memory Interface*.
nENOUT Not enable output	O	During a write cycle, this signal is driven LOW before the rising edge of **MCLK**, and remains LOW for the entire cycle. This can be used to aid arbitration in shared bus applications. See Chapter 3 *Memory Interface*.
nENOUTI Not enable output	O	During a coprocessor register transfer C-cycle from the EmbeddedICE-RT communications channel coprocessor to the ARM core, this signal goes LOW. This can be used to aid arbitration in shared bus systems.
nEXEC Not executed	O	This is HIGH when the instruction in the execution unit is not being executed because, for example, it has failed its condition code check.

Resumen de líneas de interfaz 8/#12

nFIQ Not fast interrupt request	IC	Taking this LOW causes the processor to be interrupted if the appropriate enable in the processor is active. The signal is level-sensitive and must be held LOW until a suitable response is received from the processor. **nFIQ** can be synchronous or asynchronous to **MCLK**, depending on the state of **ISYNC**.
nHIGHZ Not **HIGHZ**	O	When the current instruction is **HIGHZ** this signal is LOW. This is used to place the scan cells of that scan chain in the high impedance state. This must be left unconnected, if an external boundary-scan chain is not connected.
nIRQ Not interrupt request	IC	As **nFIQ**, but with lower priority. Can be taken LOW to interrupt the processor when the appropriate enable is active. **nIRQ** can be synchronous or asynchronous, depending on the state of **ISYNC**.
nM[4:0] Not processor mode	O	These are the inverse of the internal status bits indicating the current processor mode.
nMREQ Not memory request	O	When the processor requires memory access during the following cycle this is LOW.
nOPC Not op-code fetch	O	When the processor is fetching an instruction from memory this is LOW. This is one of the signals controlled by **APE**, **ALE**, and **ABE**.

Resumen de líneas de interfaz 9/#12

nRESET Not reset	IC	Used to start the processor from a known address. A LOW level causes the instruction being executed to terminate abnormally. This signal must be held LOW for at least two clock cycles, with **nWAIT** held HIGH. When LOW the processor performs internal cycles with the address incrementing from the point where reset was activated. The address overflows to zero if **nRESET** is held beyond the maximum address limit. When HIGH for at least one clock cycle, the processor restarts from address 0.
nRW Not read, write	O	When the processor is performing a read cycle, this is LOW. This is one of the signals controlled by **APE**, **ALE**, and **ABE**.
nTDOEN Not **TDO** enable	O	When serial data is being driven out on **TDO** this is LOW. Usually used as an output enable for a **TDO** pin in a packaged part.
nTRANS Not memory translate	O	When the processor is in User mode, this is LOW. It can be used either to tell the memory management system when address translation is turned on, or as an indicator of non-User mode activity. This is one of the signals controlled by **APE**, **ALE**, and **ABE**.
nTRST Not test reset	IC	Reset signal for the boundary-scan logic. This pin must be pulsed or driven LOW to achieve normal device operation, in addition to the normal device reset, **nRESET**. See Chapter 5 *Debug Interface*.
nWAIT Not wait	IC	When LOW the processor extends an access over a number of cycles of **MCLK**, which is useful for accessing slow memory or peripherals. Internally, **nWAIT** is logically ANDed with **MCLK** and must only change when **MCLK** is LOW. If **nWAIT** is not used it must be tied HIGH.

Resumen de líneas de interfaz 10/#12

PCLKBS Boundary scan update clock	O	This is used by an external boundary-scan chain as the update clock. This must be left unconnected, if an external boundary-scan chain is not connected.
RANGEOUT0 EmbeddedICE-RT RANGEOUT0	O	When the EmbeddedICE-RT watchpoint unit 0 has matched the conditions currently present on the address, data, and control buses, then this is HIGH. This signal is independent of the state of the watchpoint enable control bit. **RANGEOUT0** changes when **ECLK** is LOW.
RANGEOUT1 EmbeddedICE-RT RANGEOUT1	O	As **RANGEOUT0** but corresponds to the EmbeddedICE-RT watchpoint unit 1.
RSTCLKBS Boundary scan Reset Clock	O	When either the TAP controller state machine is in the RESET state or when **nTRST** is LOW, then this is HIGH. This can be used to reset external boundary-scan cells.
SCREG[3:0] Scan chain register	O	These reflect the ID number of the scan chain currently selected by the TAP controller. These change on the falling edge of **TCK** when the TAP state machine is in the UPDATE-DR state.
SDINBS Boundary scan serial input data	O	This provides the serial data for an external boundary-scan chain input. It changes from the rising edge of **TCK** and is valid at the falling edge of **TCK**.
SDOUTBS Boundary scan serial output data	IC	Accepts serial data from an external boundary-scan chain output, synchronized to the rising edge of **TCK**. This must be tied LOW, if an external boundary-scan chain is not connected.

Resumen de líneas de interfaz 11/#12

SEQ Sequential address	O	When the address of the next memory cycle is closely related to that of the last memory access, this is HIGH. In ARM state the new address can be for the same word or the next. In THUMB state, the same halfword or the next. It can be used, in combination with the low-order address lines, to indicate that the next cycle can use a fast memory mode (for example DRAM page mode) or to bypass the address translation system.
SHCLKBS Boundary scan shift clock, phase one	O	Used to clock the master half of the external scan cells and follows **TCK1** when in the SHIFT-DR state of the state machine and scan chain 3 is selected. When not in the SHIFT-DR state or when scan chain 3 is not selected, this clock is LOW.
SHCLK2BS Boundary scan shift clock, phase two	O	As **SHCLKBS** but follows **TCK2** instead of **TCK1**. This must be left unconnected, if an external boundary-scan chain is not connected.
TAPSM[3:0] TAP controller state machine	O	These reflect the current state of the TAP controller state machine. These bits change on the rising edge of **TCK**. See Figure B-2 on page B-5.
TBE Test bus enable	IC	When LOW, **D[31:0]**, **A[31:0]**, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** are set to high impedance. Similar in effect as if both **ABE** and **DBE** had been driven LOW. However, **TBE** does not have an associated scan cell and so enables external signals to be driven high impedance during scan testing. Under normal operating conditions **TBE** must be HIGH.

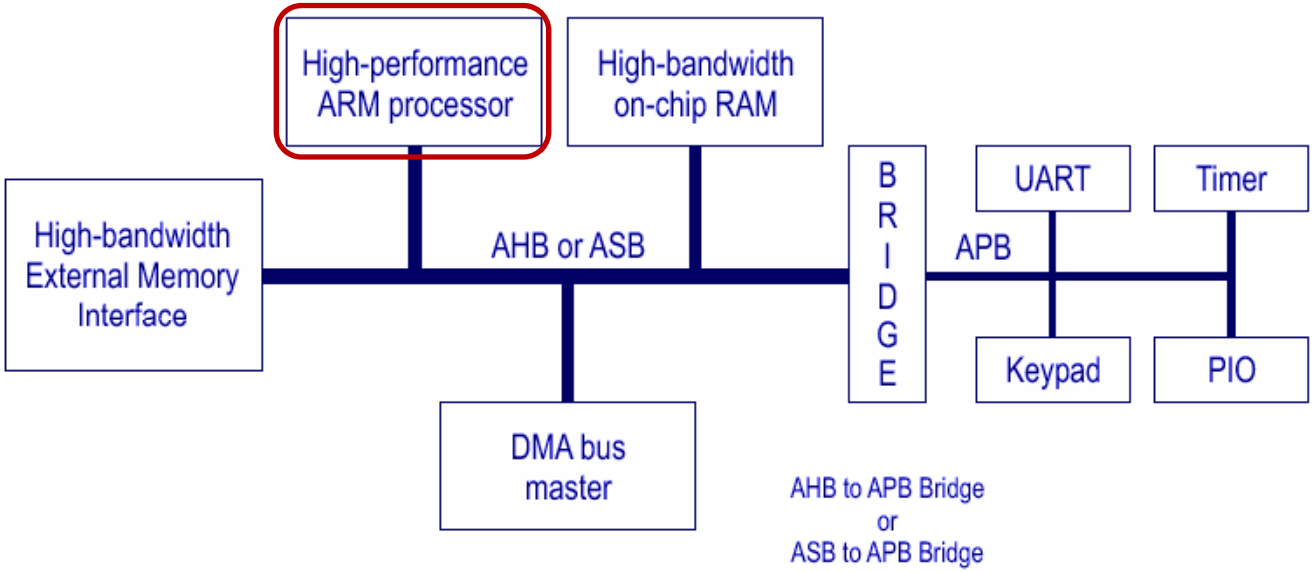
Resumen de líneas de interfaz 12/#12

TBIT	O	When the processor is executing the THUMB instruction set, this is HIGH. It is LOW when executing the ARM instruction set. This signal changes in phase two in the first execute cycle of a BX instruction.
TCK	IC	Clock signal for all test circuitry. When in debug state, this is used to generate **DCLK**, **TCK1**, and **TCK2**.
TCK1 **TCK**, phase one	O	HIGH when **TCK** is HIGH (slight phase lag because of the internal clock non-overlap).
TCK2 **TCK**, phase two	O	HIGH when **TCK** is LOW (slight phase lag because of the internal clock non-overlap). It is the non-overlapping complement of **TCK1**.
TDI	IC	Serial data for the scan chains.
TDO Test data output	O	Serial data from the scan chains.
TMS	IC	Mode select for scan chains.
VDD Power supply	P	Provide power to the device.
VSS Ground	P	These connections are the ground reference for all signals.

El bus AMBA: introducción

The Advance Microcontroller Bus Architecture

- Bus estándar diseñado por ARM para comunicación *on-chip*



AMBA AHB

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

AMBA ASB

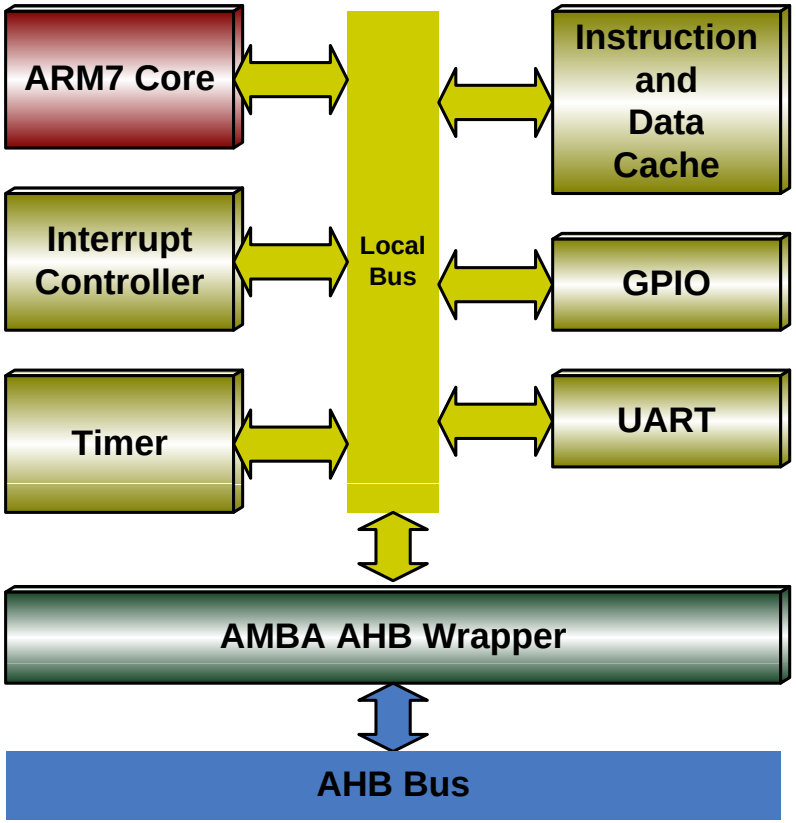
* High performance
* Pipelined operation
* Multiple bus masters

AMBA APB

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

El bus AMBA: *AHB Wrapper*

- Necesidad de un módulo de adaptación (*wrapper*) para conectar el bus local con el bus ASB/AHB



- Adaptación de las señales del interfaz del ARM a las señales especificadas por el protocolo del bus AMBA
- ARM7:
  - *A[31:0], Din[31:0], Dout[31:0], mreq*, seq, mas[1:0], wait**
- AMBA:
  - *HBUSREQ, HGRANT, HADDR[31:0], HTRANS[1:0]*

El bus AMBA: líneas de interfaz AHB 1/#2

HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of **HCLK.**
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.

El bus AMBA: líneas de interfaz AHB 2/#2

HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELx Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require **HREADY** as both an input and an output signal.
HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

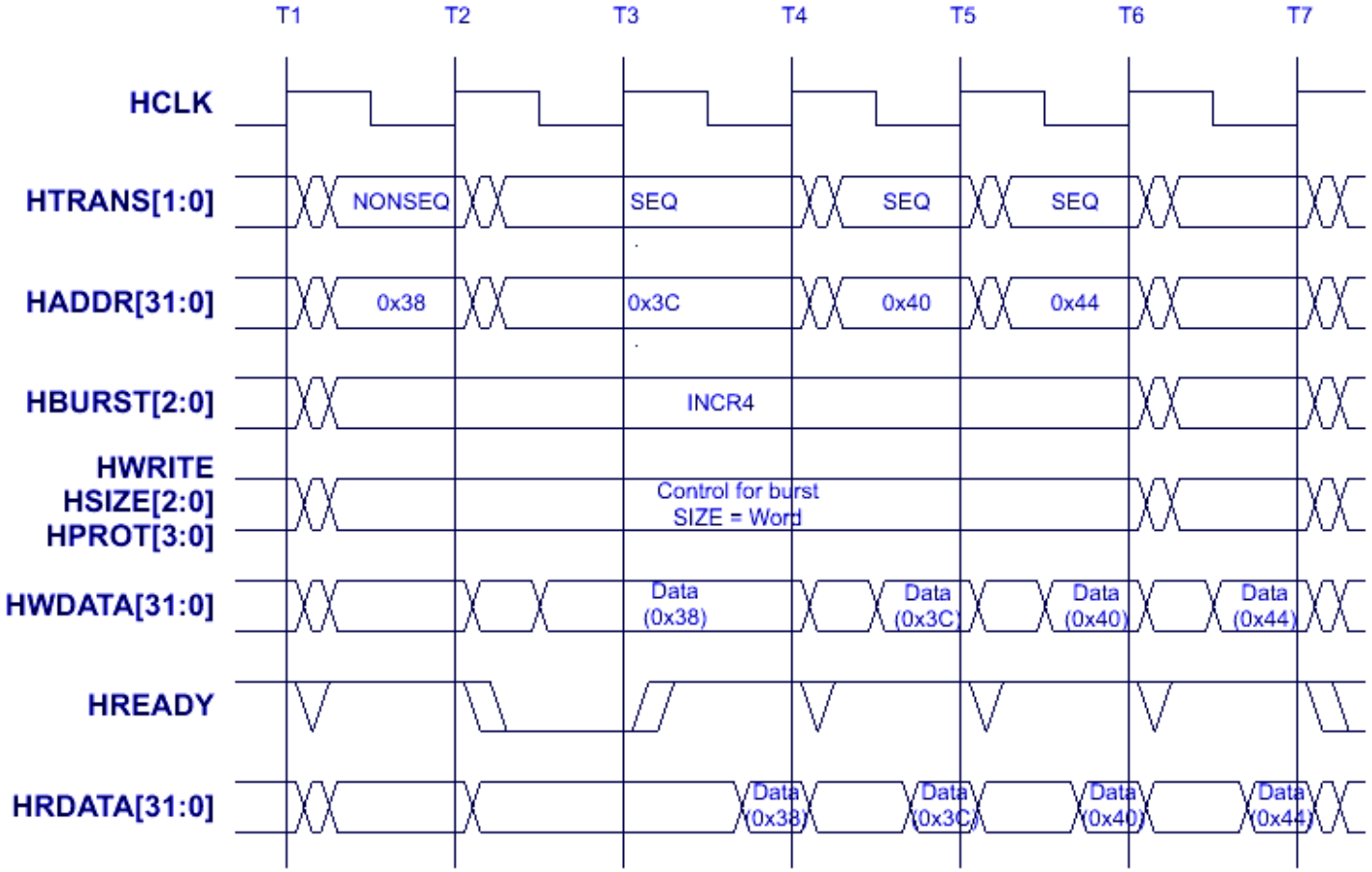
El bus AMBA: ciclo de bus

Secuencia de operaciones

- Las operaciones en el bus se realizan de forma síncrona ***HCLK*** (usualmente corresponde con *mclk*)
- El ciclo comienza cuando un maestro *x* (*master*) envía una petición (*request*) ***HBUSREQx*** al árbitro (*arbiter*)
- Si el bus está libre, el árbitro concede (*grants*) ***HGRANTx*** el bus al maestro *x*
  - El árbitro no podrá conceder el bus si éste se encuentra bloqueado ***HMASTLOCK***
  - El estándar AMBA especifica el protocolo de comunicación pero no el esquema de prioridad
- Una vez concedido el bus a un determinado maestro comienza la transferencia (ciclo de lectura/escritura)
  - *Write data bus*: transferencia *Master* → *Slave*
  - *Read data bus*: transferencia *Slave* → *Master*

El bus AMBA: *AHB Master transfer cycle*

Ciclos de bus

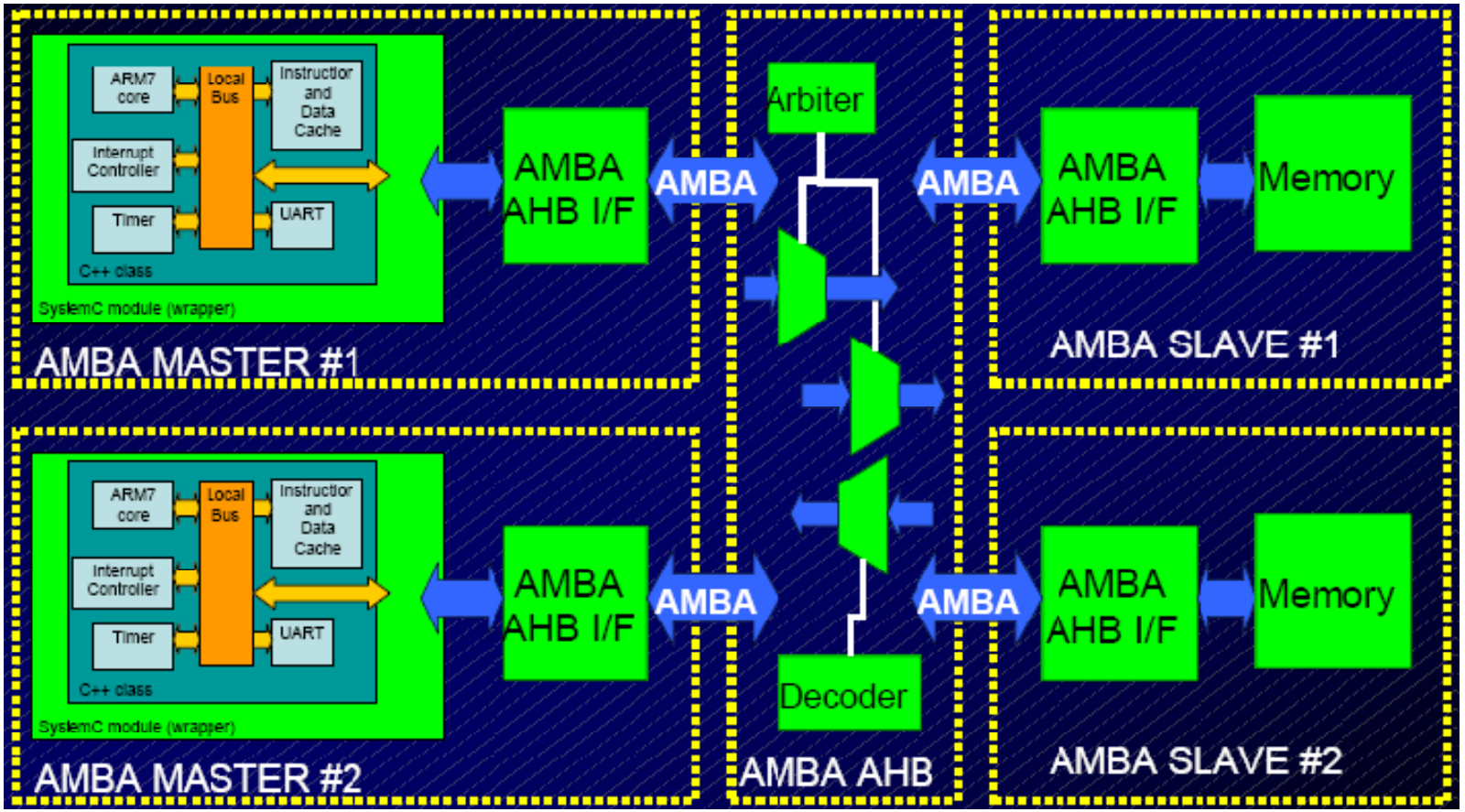


El bus AMBA: *AHB Slave responses*

- Una vez el *master* comienza la transferencia, el *slave* determina el progreso de la misma
  - Okay: la transferencia se completa de forma inmediata
  - Wait: es necesario insertar uno o más ciclos de espera (*wait cycles*) para completar la transferencia
  - Error: se activa la señal de error indicando que la transferencia ha fallado
  - Retry or Split: indica que la transferencia se retarda y permite tanto al maestro como al esclavo desconectarse del bus, dando la posibilidad de que se inicien otras transferencias (especialmente útil cuando se trata de dispositivos de entrada/salida lentos). Este modo mejora el rendimiento del bus AHB

El bus AMBA: sistema con bus multiplexado I

- Uso de interfaz AHB en *masters* y *slaves*

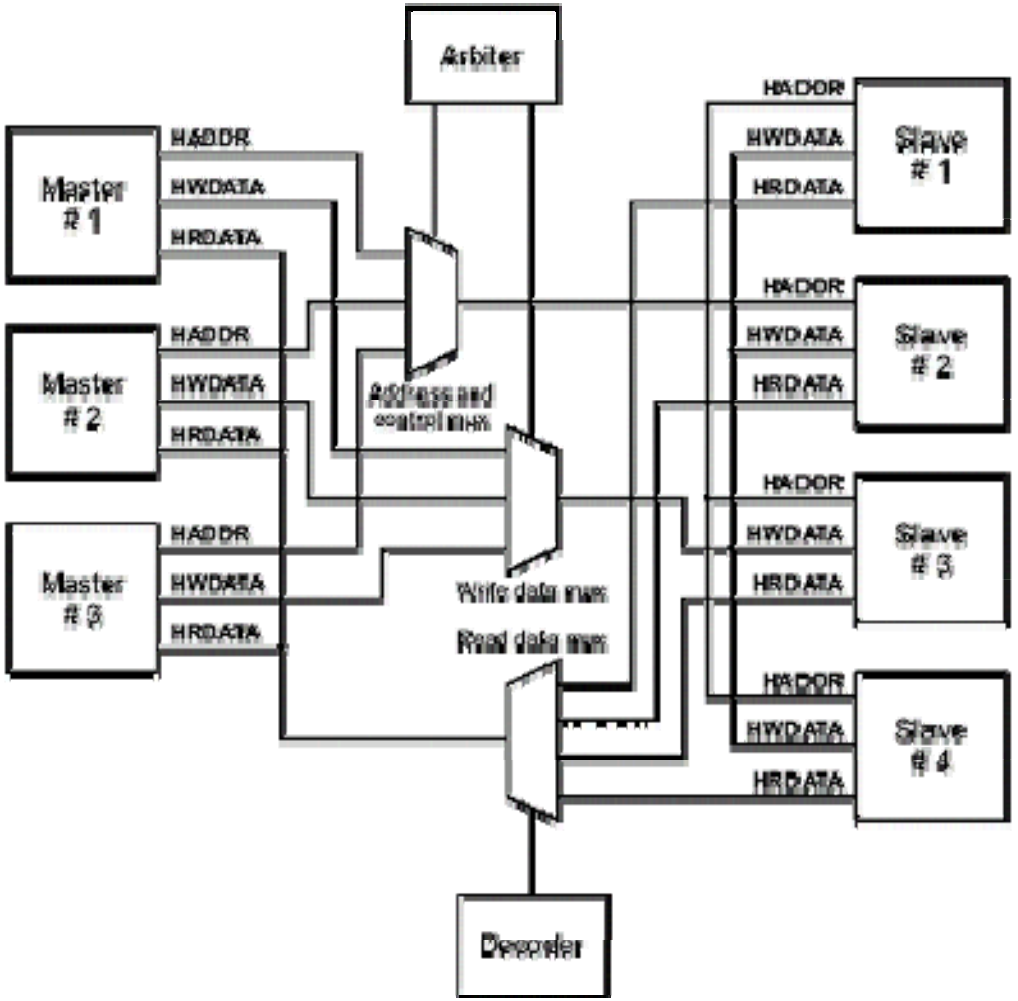


El bus AMBA: líneas de interfaz AHB *arbitration* 1/#1

HBUSREQx Bus request	Master	A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus. There is an **HBUSREQx** signal for each bus master in the system, up to a maximum of 16 bus masters.
HLOCKx Locked transfers	Master	When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.
HGRANTx Bus grant	Arbiter	This signal indicates that bus master x is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when **HREADY** is HIGH, so a master gets access to the bus when both **HREADY** and **HGRANTx** are HIGH.
HMASTER[3:0] Master number	Arbiter	These signals from the arbiter indicate which bus master is currently performing a transfer and is used by the slaves which support SPLIT transfers to determine which master is attempting an access. The timing of **HMASTER** is aligned with the timing of the address and control signals.
HMASTLOCK Locked sequence	Arbiter	Indicates that the current master is performing a locked sequence of transfers. This signal has the same timing as the **HMASTER** signal.
HSPLITx[15:0] Split completion request	Slave (SPLIT-capable)	This 16-bit split bus is used by a slave to indicate to the arbiter which bus masters should be allowed to re-attempt a split transaction. Each bit of this split bus corresponds to a single bus master.

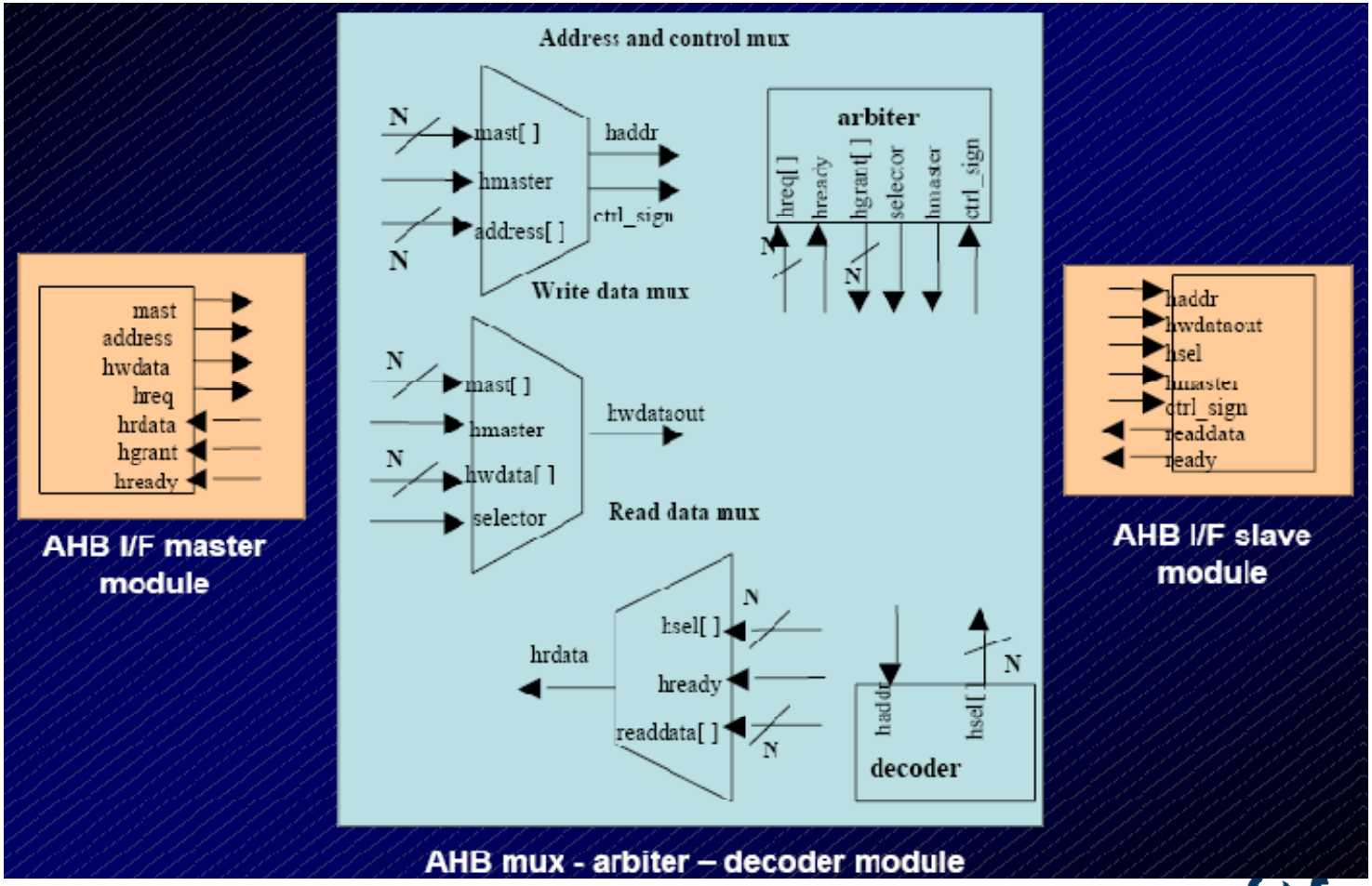
El bus AMBA: sistema con bus multiplexado II

- Ejemplo de conexionado con múltiples *masters/slaves*



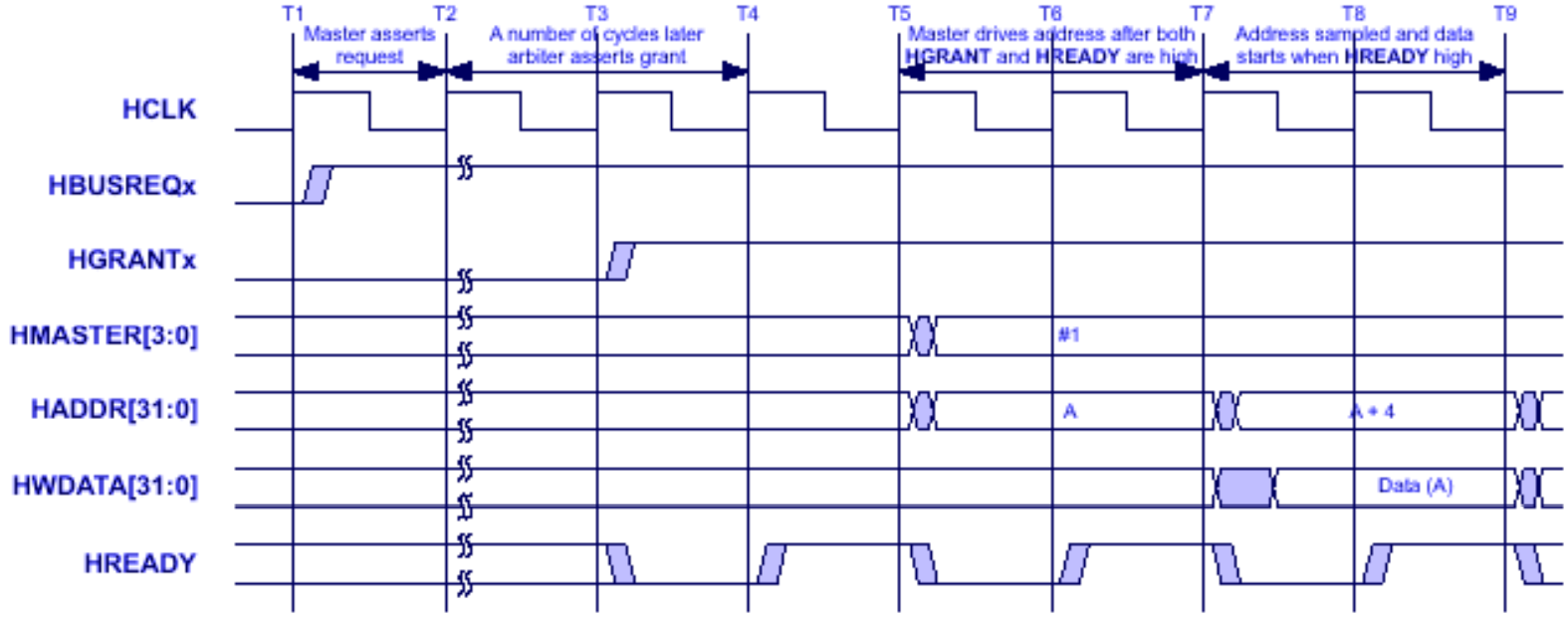
El bus AMBA: módulos del bus AMBA AHB

- Descripción de las señales de los módulos de interfaz en el bus AMBA AHB



El bus AMBA: *AHB arbitration and bus Lock*

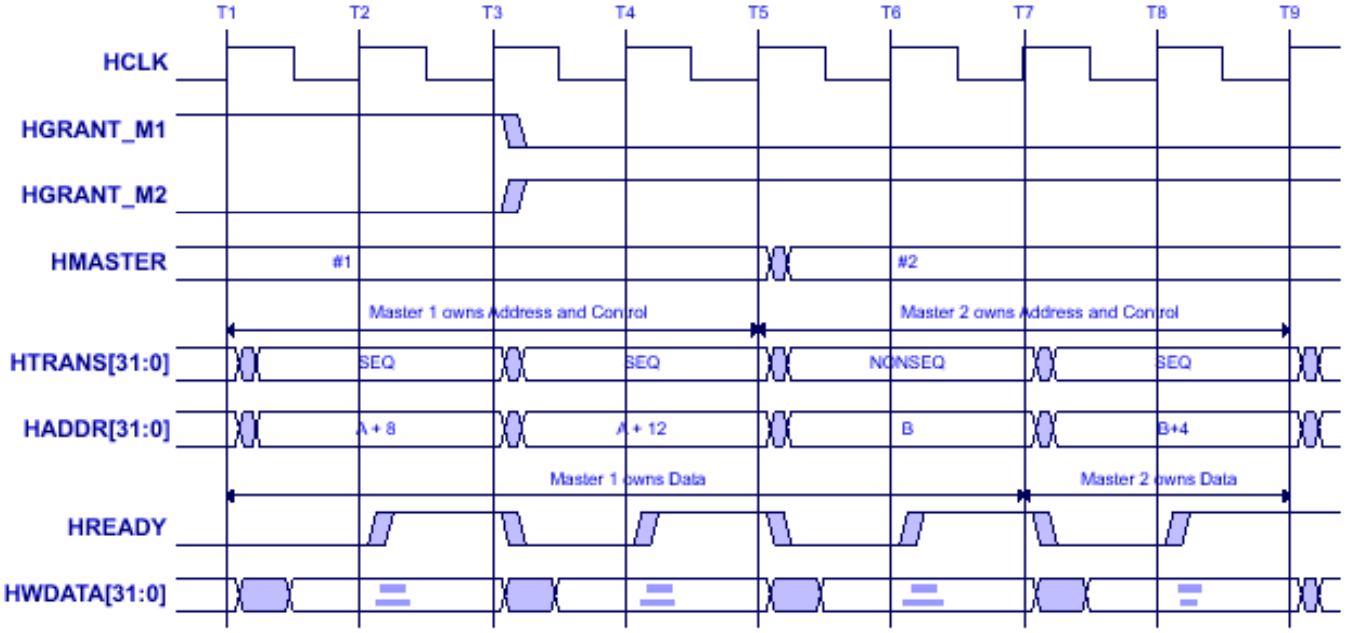
- HBUSREQx → *arbitration* → HGRANTx (& HREADY) → acceso al bus



- Secuencia de bloqueo del bus: HLOCKx
  - El bus sólo se concede si no está bloqueado por ningún maestro (ej. operación *read-modify-write*)

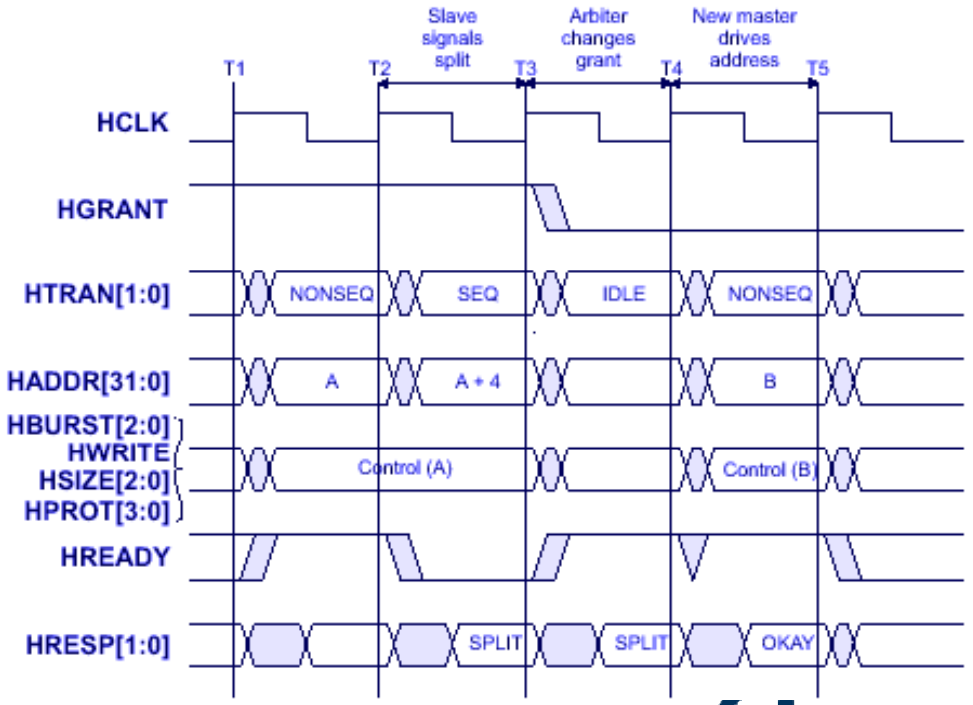
El bus AMBA: *AHB bus Ownership*

- Consideraciones en la concesión del bus
  - La concesión del bus se produce cuando HGRANTx y HREADY están activas
  - El bus HMASTER[3:0] indica el maestro en curso
  - El maestro en curso continua controlando el bus hasta completar la transferencia



El bus AMBA: *AHB split transfers I*

- Cuando se accede a un periférico lento
  - Éste puede señalar la disponibilidad del bus antes de finalizar la transferencia, HRESP[1:0] = SPLIT
  - Es posible realizar otras transferencias antes de finalizar la anterior
- El *slave* debe
  - Señalizar el SPLIT
  - Informar en cada momento al árbitro y al maestro cuándo está listo, HSPLITx, HMASTER[3:0]



El bus AMBA: *AHB split transfers II*

Consideraciones

- El árbitro debe enmascarar la petición de acceso al bus del *master* hasta que el *slave* esté listo y enviar la señal de *grant* en ese momento
- Un maestro únicamente puede tener una transferencia pendiente (*split transfer*)
- Un *slave* puede tener múltiples peticiones de transferencia pendientes