Bus Protocols and Interfacing

- Bus basics
- I/O transactions
- MPC823 bus

Reference:
Chapter 13 of “White Book”

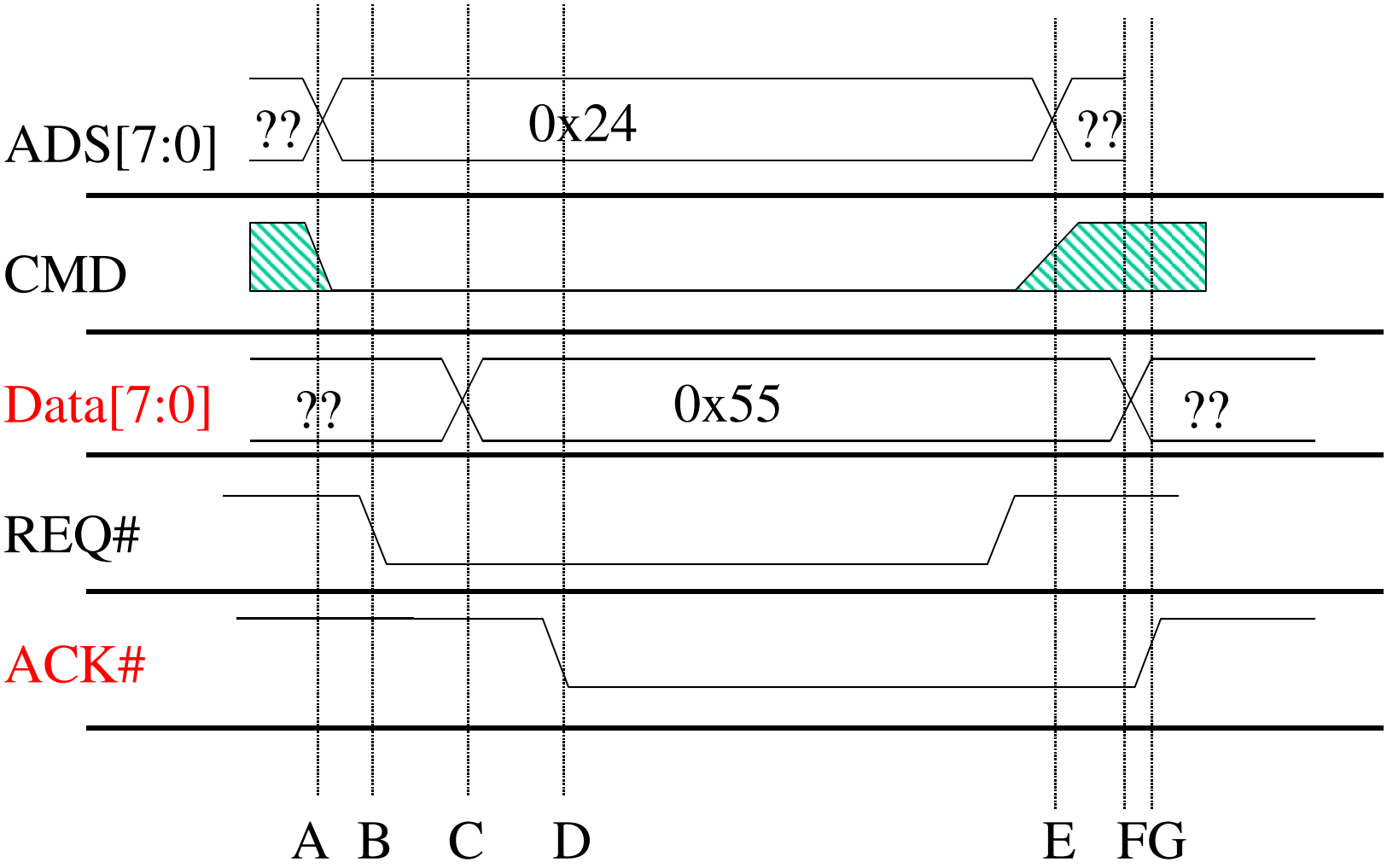
Basic example

- Discuss a basic bus protocol
  - Asynchronous (no clock)
  - Initiator and Target
  - REQ#, ACK#, Data[7:0], ADS[7:0], CMD
    - CMD=0 is read, CMD=1 is write.
    - REQ# low means initiator is requesting something.
    - ACK# low means target has done its job.

A read transaction

- Say initiator wants to read location 0x24
  - Initiator sets ADS=0x24, CMD=0.
  - Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  - Target sees read request.
  - Target drives data onto data bus.
  - Target *then* sets ACK# to low.
  - Initiator grabs the data from the data bus.
  - Initiator sets REQ# to high, stops driving ADS and CMD
  - Target stops driving data, sets ACK# to low terminating the transaction

Read transaction

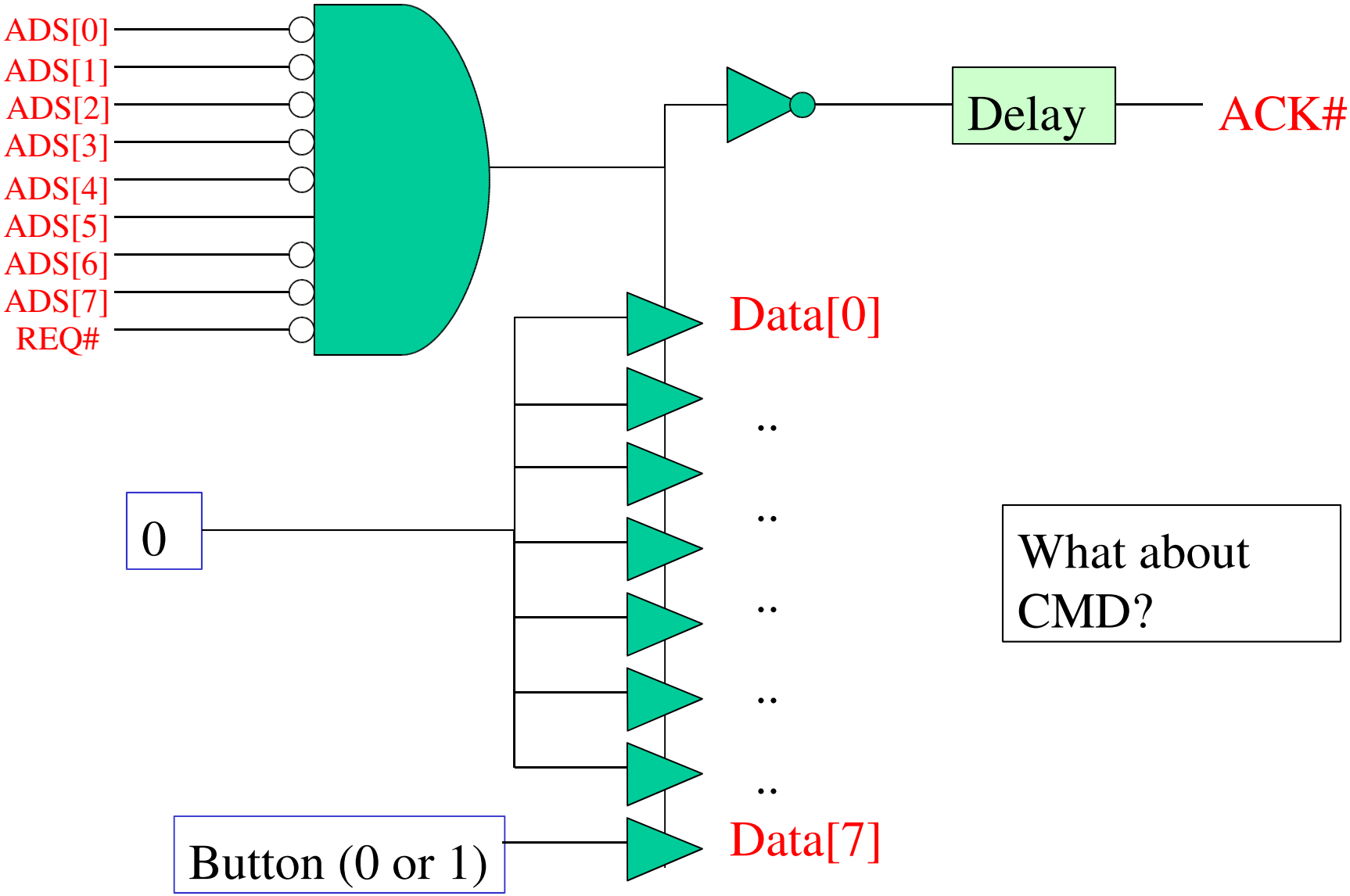


A write transcation
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target *then* sets ACK# to low.
- Initiator sets REQ# to high & stops driving other lines.
- Target sets ACK# to high terminating the transaction

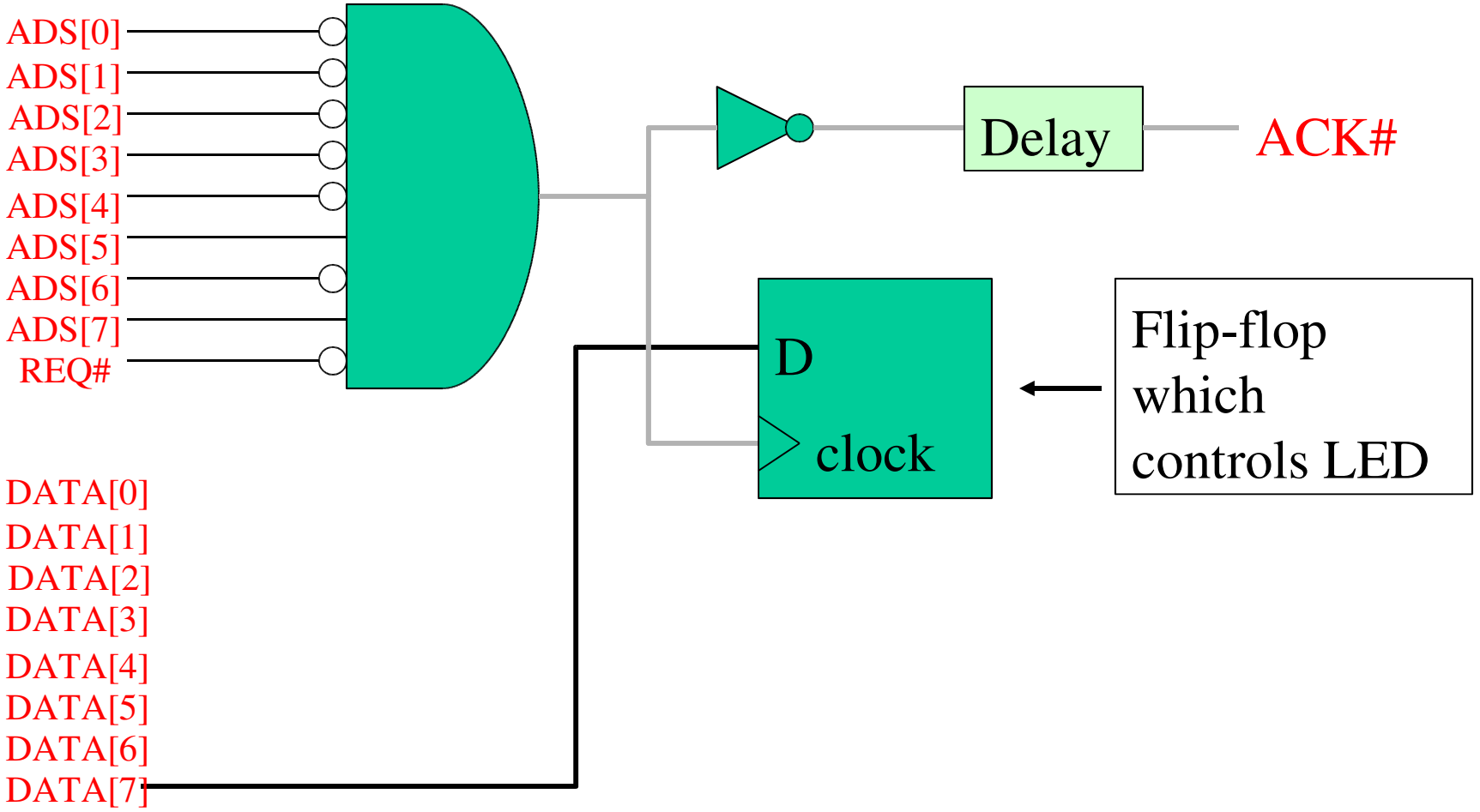
The push-button

(if ADS=0x04 write 0 or 1 depending on button)



The LED

(1 bit reg written by LSB of address 0x05)



MPC823 Bus

The basic function of the MPC823 bus is similar, though slightly more complicated. (Chapter 13 of "White Book")

- Timing is controlled by a global clock; all signals are in reference to the rising edge of this clock.
- 32-bit data bus D[0:31]
- 26-bit address bus A[6:31]
  - A[0:5] not sent off chip
  - On-chip peripherals still see 32 address bits
- Basic control lines:
  - RD/#WR
  - #TS (transfer start)—like #REQ but only asserted on first clock cycle of transaction
  - #TA (transfer acknowledge)—like #ACK

MPC823 Read

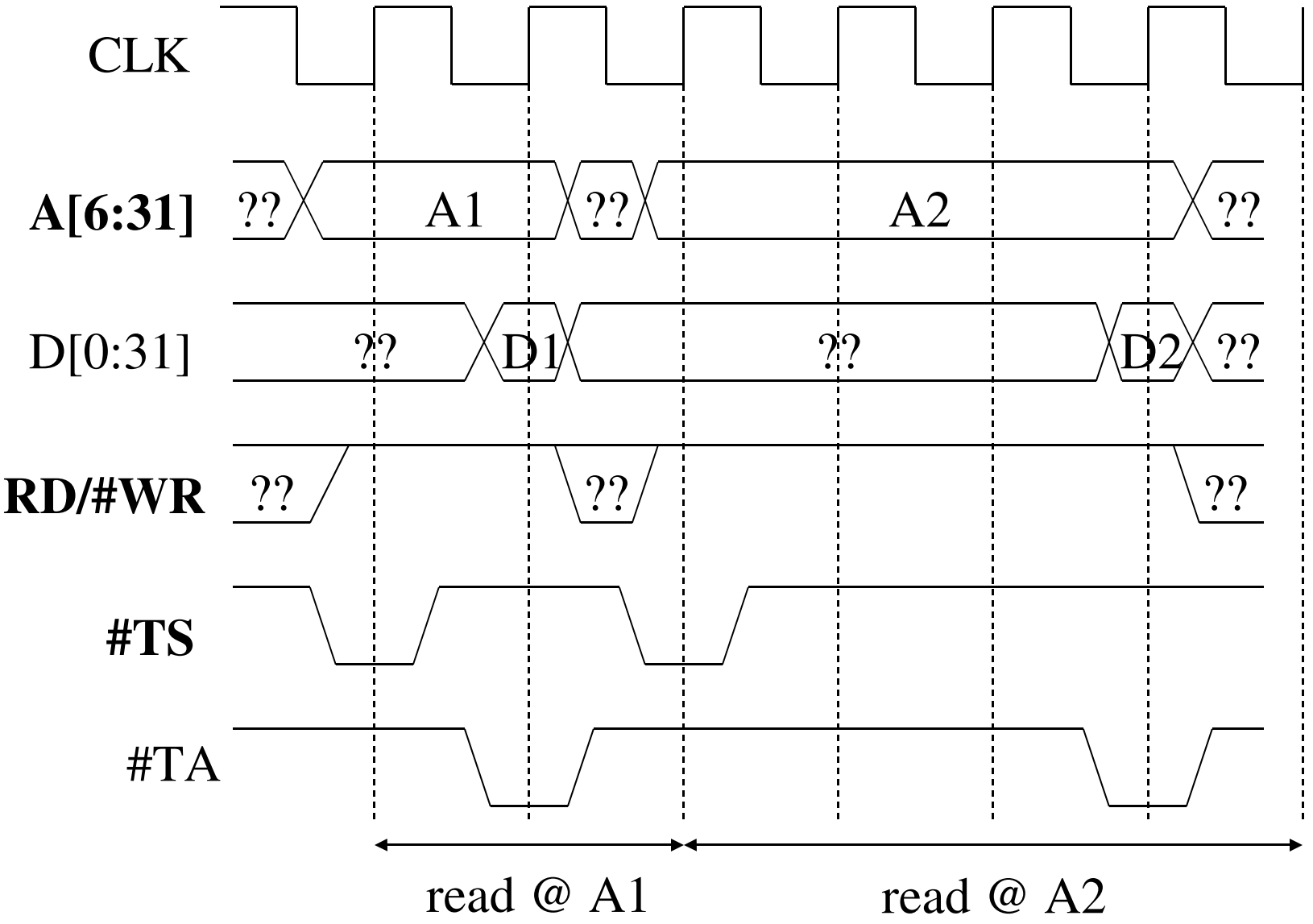
• Master drives address, RD/#WR, and #TS to initiate a read transaction. Address and RD/#WR guaranteed valid at same rising clock edge that TS' is asserted. Master de-asserts #TS after one cycle, but keeps driving address and RD/#WR until it sees #TA.

• Slaves look at address and RD/#WR when #TS asserted. **One** of them will drive data and assert #TA. Master samples data on same rising clock edge when #TA is asserted.

• Minimum transactions takes two clock cycles.

• Transactions can take longer; slow slaves add wait states by not asserting #TA.

MPC823 Read



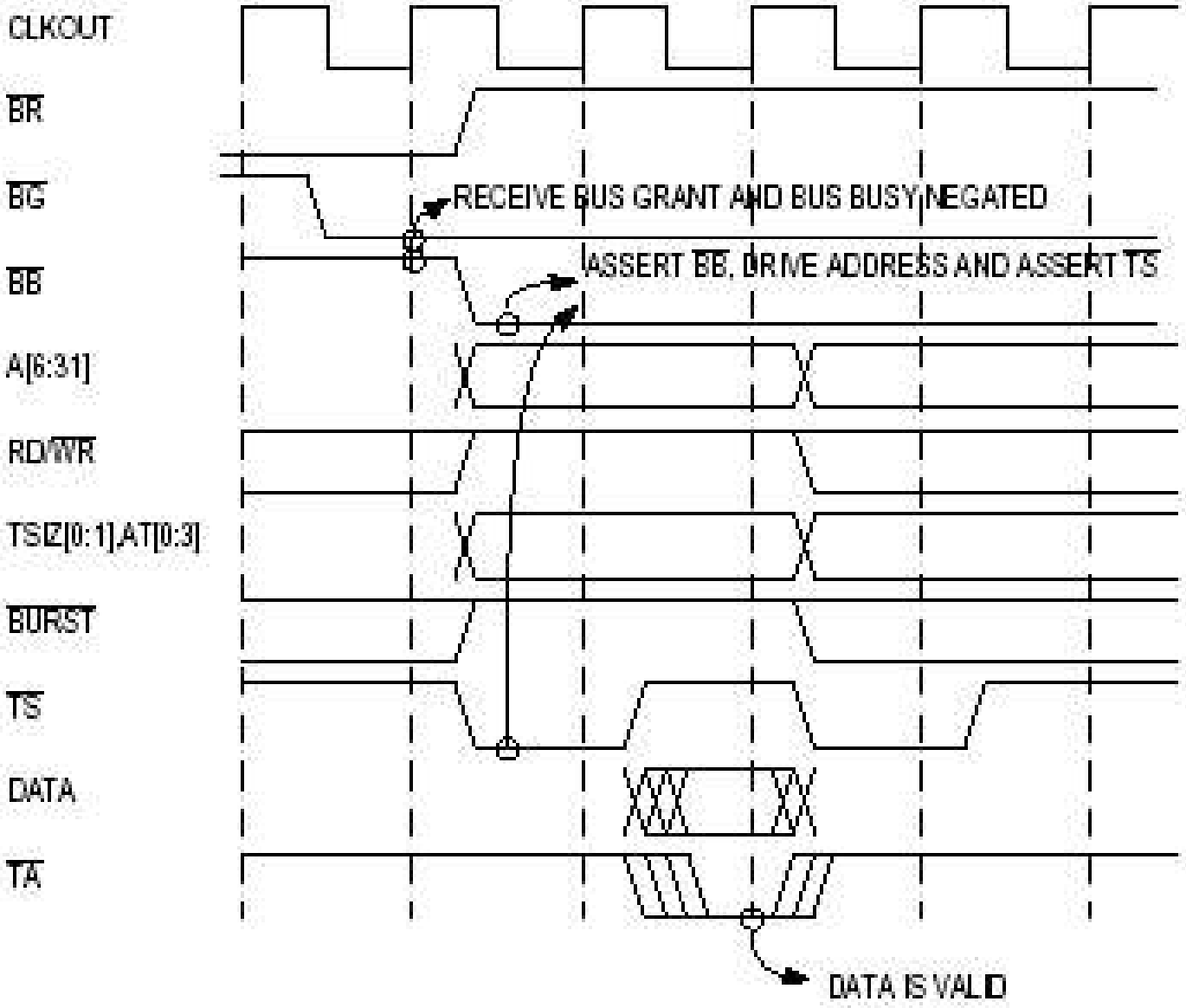


Figure 13-4. Single Beat Read Cycle–Basic Timing–Zero Wait States

MPC823 Write

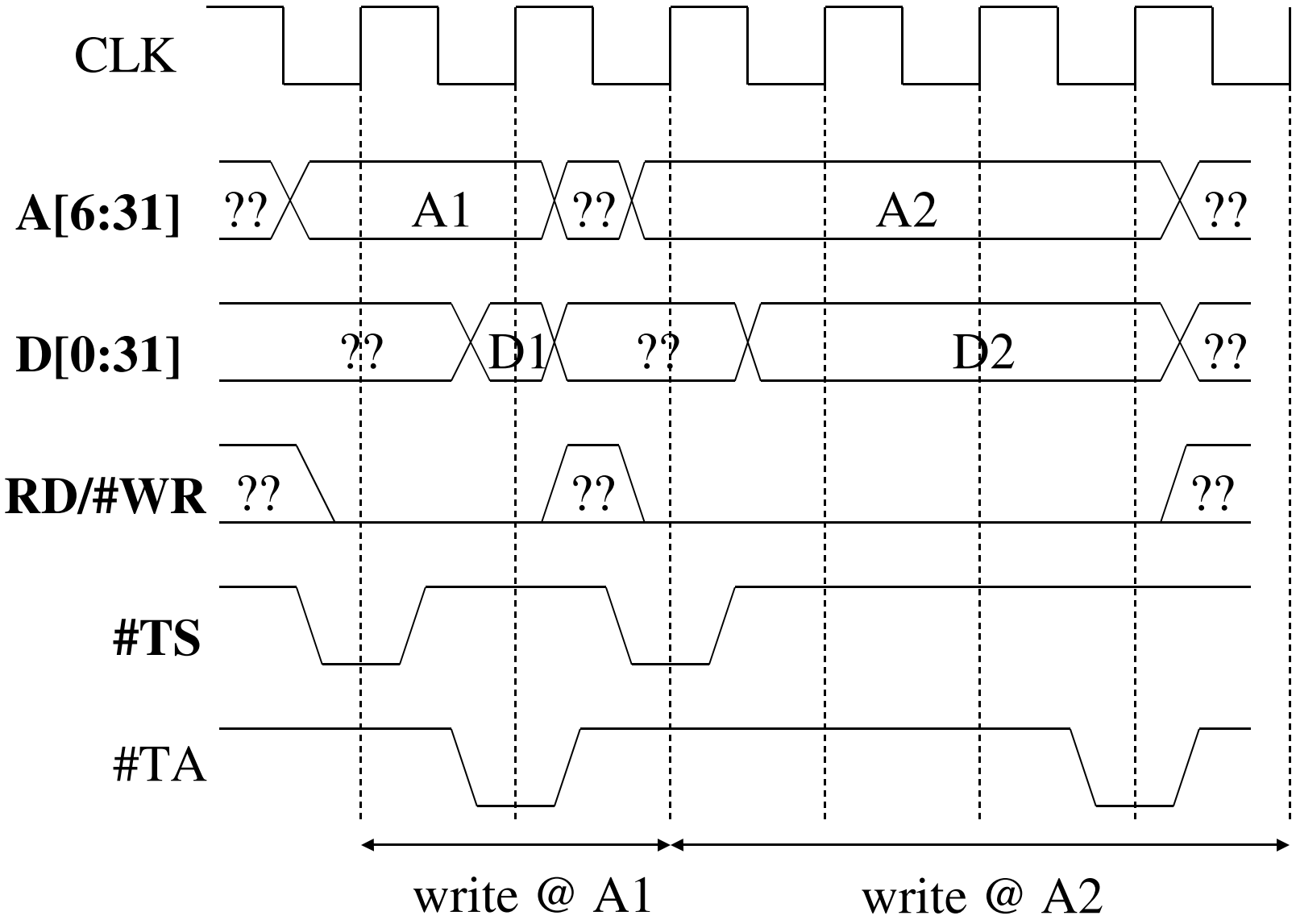
- Same start as for a read, except for polarity of RD/WR#.

- Master drives data by 2nd cycle.

- Slaves look at address and RD/#WR when #TS is asserted. One of them will read data and assert #TA.

- As with reads, minimum transaction length is two cycles. Slaves can take longer, however, by not asserting #TA. Master keeps driving address, RD/#WR, and data until it sees #TA asserted.

MPC823 Write



Of course, things aren't that simple

- **TSI[0:1] –** Specifies the size of the data to be transferred.
- **DP[0:3] –** Data parity
- **#Burst –** Indicates a burst transaction
- **#BDIP –** Burst Data In Progress (more burst stuff)
- **#BI –** Burst Inhibit. Indicates that the slave doesn't support burst transactions
- Many More…

Example of complexity: The Burst Mechanism (13.4.4)

- Burst transfers are used to move 16 bytes at a time
- #BURST must be asserted by master
- #BI must **not** be asserted by slave
- Must be a 16-byte aligned access
- Supports critical word first.

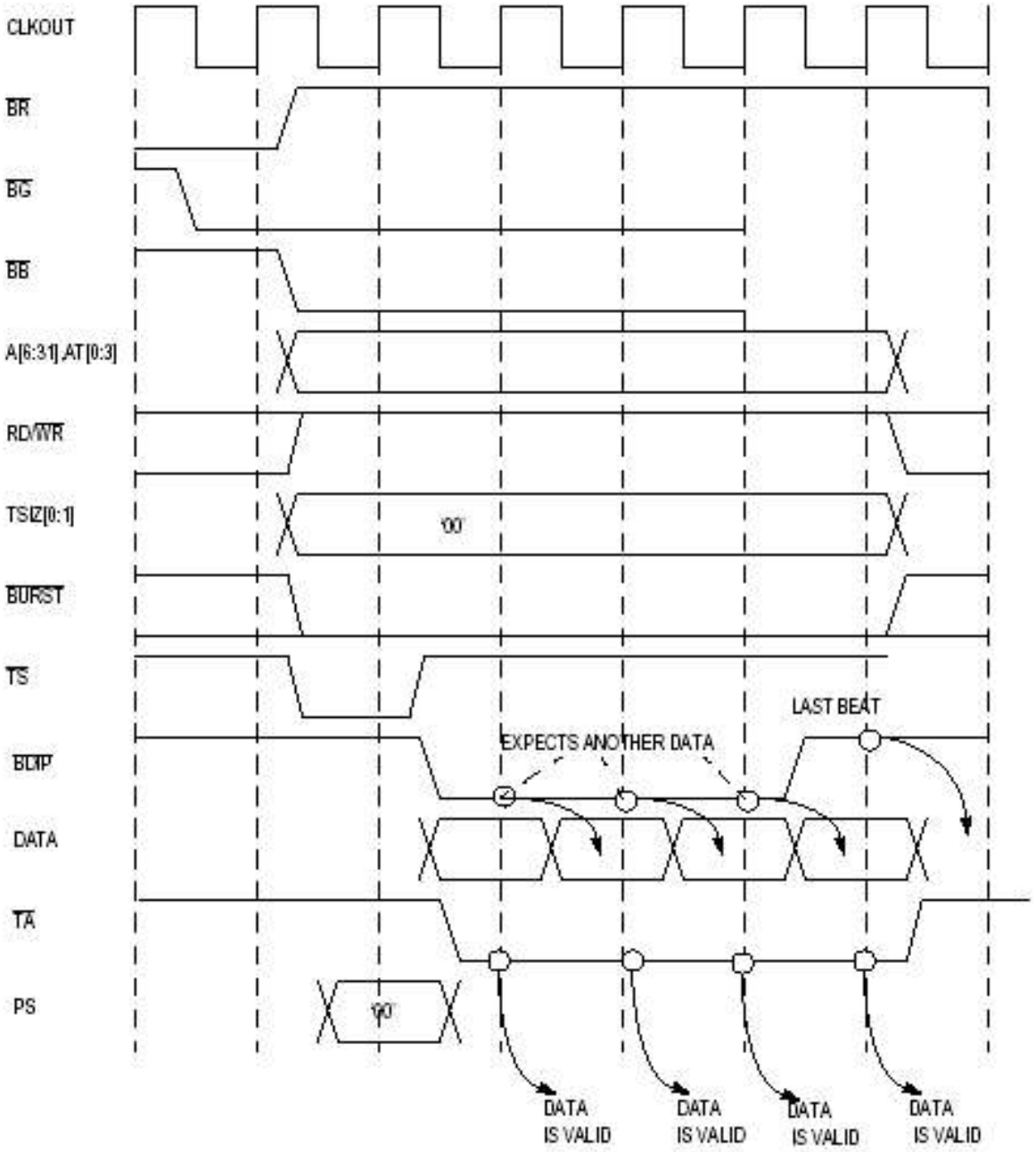
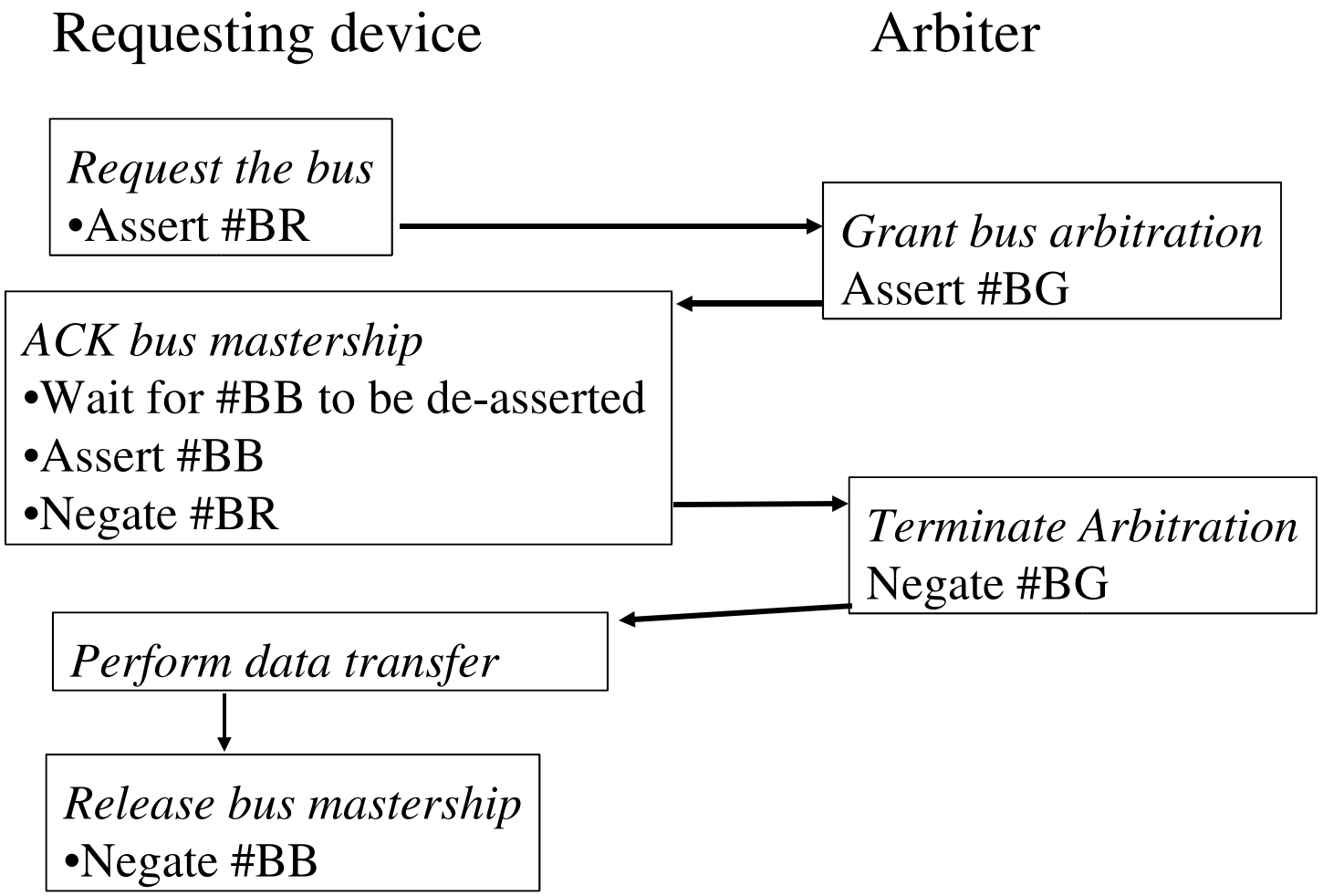
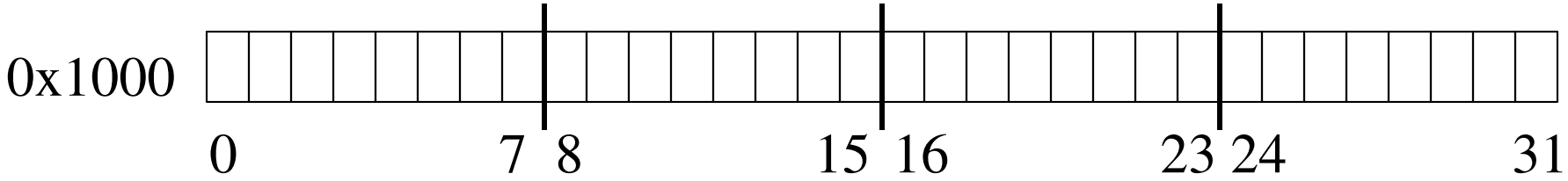


Figure 13-11. Burst-Read Cycle–32-Bit Port Size–Zero Wait State

Arbitration (page 13-28)



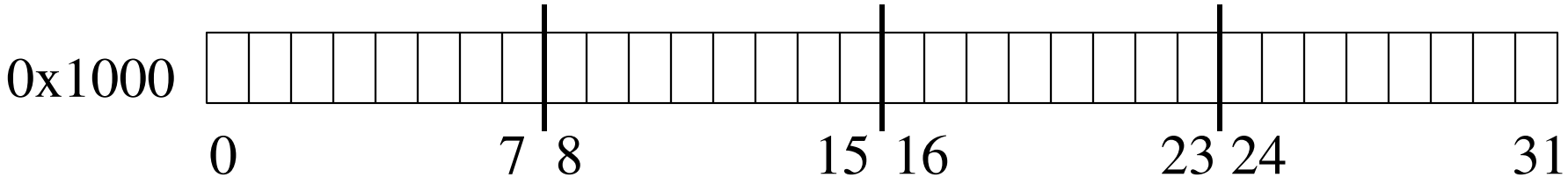
Transfer Alignment



MPC823 external bus supports natural address alignment

- Byte access: Any address alignment
- Half-word access: Address bit 31 equal to 0
- Word access: Address bits 31 and 30 equal to 0

Dealing with Smaller Accesses: Reads



Assume that the word value 0x12345678 is stored at 0x1000 and that r4 contains 0x1000. What happens on the following transfers?

D[0:7] D[8:15] D[16:23] D[24:31]

- lbz r3 , 0 (r4)
- lbz r3 , 1 (r4)
- lbz r3 , 2 (r4)
- lbz r3 , 3 (r4)
- lhz r3 , 0 (r4)
- lhz r3 , 2 (r4)

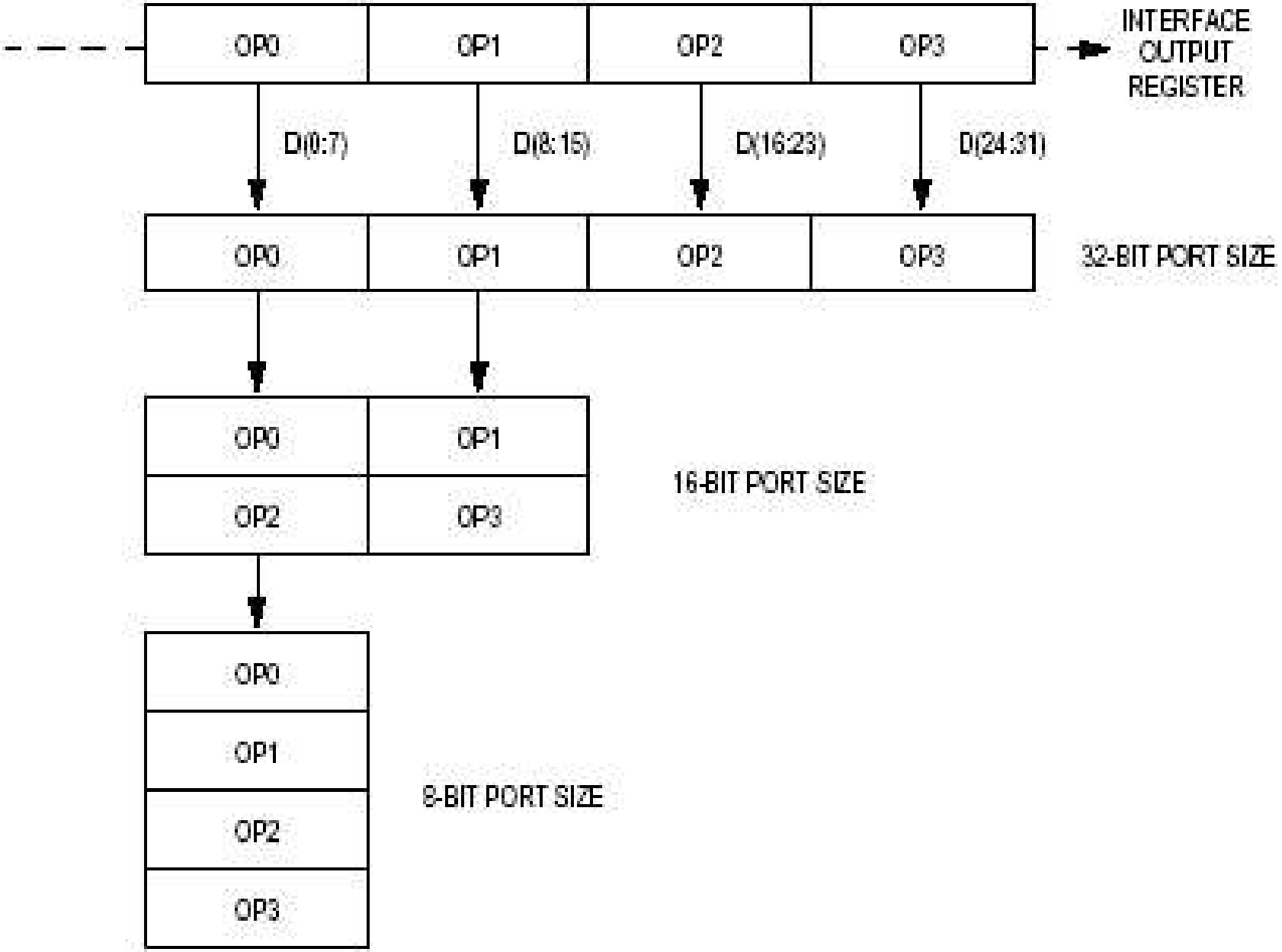


Figure 13-19. Interface To Different Port Size Devices

Table 13-2. Data Bus Requirements For Read Cycles

TRANSFER SIZE	TSIZE [0:1]		INTERNAL ADDRESS		32-BIT PORT SIZE				16-BIT PORT SIZE		8-BIT PORT SIZE
			A30	A31	D0–D7	D8–D15	D16–D23	D24–D31	D0–D7	D8–D15	D0–D7
Byte	0	1	0	0	OP0	—	—	—	OP0	—	OP0
	0	1	0	1	—	OP1	—	—	—	OP1	OP1
	0	1	1	0	—	—	OP2	—	OP2	—	OP2
	0	1	1	1	—	—	—	OP3	—	OP3	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—	OP0	OP1	OP0
	1	0	1	0	—	—	OP2	OP3	OP2	OP3	OP2
Word	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0

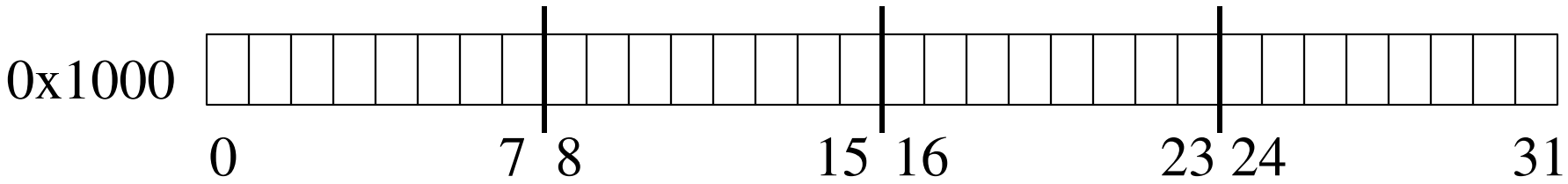
NOTE: — Denotes that a byte is not required during that read cycle.

Table 13-3. Data Bus Contents for Write Cycles

TRANSFER SIZE	TSIZE [0:1]		INTERNAL ADDRESS		EXTERNAL DATA BUS PATTERN			
			A30	A31	D0–D7	D8–D15	D16–D23	D24–D31
Byte	0	1	0	0	OP0	—	—	—
	0	1	0	1	OP1	OP1	—	—
	0	1	1	0	OP2	—	OP2	—
	0	1	1	1	OP3	OP3	—	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—
	1	0	1	0	OP2	OP3	OP2	OP3
Word	0	0	0	0	OP0	OP1	OP2	OP3

NOTE: — Denotes that a byte is not required during that read cycle.

Dealing with Smaller Accesses: Writes



How about the following transfers?

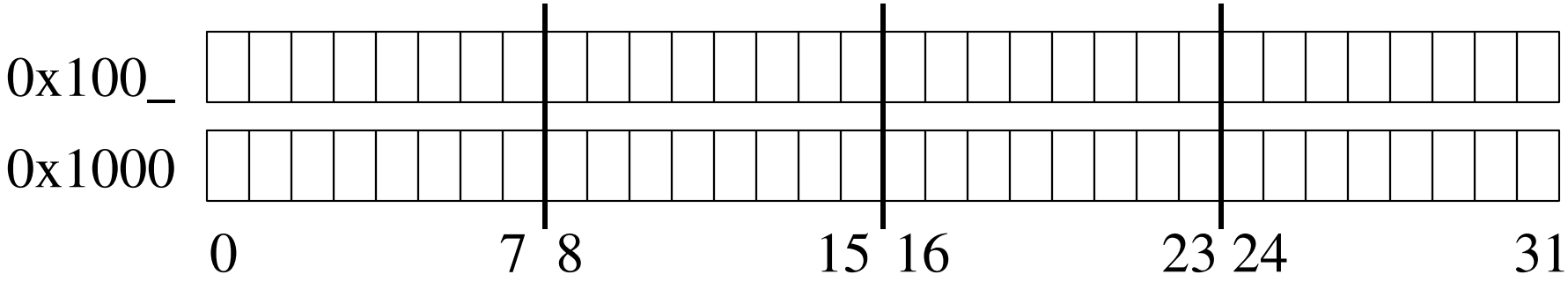
D[0:7] D[8:15] D[16:23] D[24:31]

- stb r2 , 0 (r4)
- stb r2 , 1 (r4)
- stb r2 , 2 (r4)
- stb r2 , 3 (r4)
- sth r2 , 0 (r4)
- sth r2 , 1 (r4)

Dealing with Smaller Accesses

- On a write access, which two factors determine which bits in a 32-bit word are updated?
- On most wide buses, the master drives *byte enable* lines instead of less significant address bits
  - Moto 68000 (16 bits): LDS', UDS' (no address LSB)
  - 32-bit buses: Replace low 2 address bits with 4 byte enables
- MPC823 does not:
  - Full byte address provided
  - Size (byte, halfword, word) encoded on two control lines TSIZ[0-1]

Unaligned Accesses



Consider two adjacent 32-bit memory locations and assume that r4 = 0x1000. What happens when the CPU executes the following instructions?

- lwz r3 , 2 (r4)

- lwz r3 , 1 (r4)

Problems with Unaligned Accesses

What are some problems with unaligned accesses?

Basic bus issues

- What are the basic wires for specifying the transaction and moving the data
  - What are the types of transactions? How are they specified?
  - How is length of data transfer specified?
- Who can delay (insert wait states?)
- How is arbitration done?
- Out-of-order transfers allowed?
  - Any restrictions?
- Error reporting?
- Weirdness?
  - Alignment for example.

Transaction types

- Usually read/write with a length
  - But in a given domain, other info might be important.
    - Data vs. Code access.
    - I/O vs. memory access
    - Hints to target device
  - Length might be arbitrary.

Delaying

- Who can delay and how
  - Usually a target (slave) can delay
  - Sometimes initiator (master) can delay
  - Sometimes initiator can drop the transaction
  - Sometimes the target has options on *how* to delay.

Arbitration

- Fairness
  - Even sharing, priority sharing, weighted sharing
- Mechanism
  - Centralized arbiter
  - Distributed arbiter
  - Combination
- Duration
  - Until done
  - Until someone else requests
  - Until certain time passes.
  - Combination

Out-of-order

- Does the bus allow transactions to complete out-of-order?
  - If so, can increase bandwidth (why?)
  - If so, might have to worry about ordering issues
    - Memory consistency models not a topic for this class (take EECS 570!) but basics are pretty easy to grasp

Out-of-order: Ordering problem

Processor 1

```
write I=1
write J=1
```

- Processor 2

```
Write J=2
Write I=2
```

If both programs are executed in order, is there any setting of J and I which is impossible?