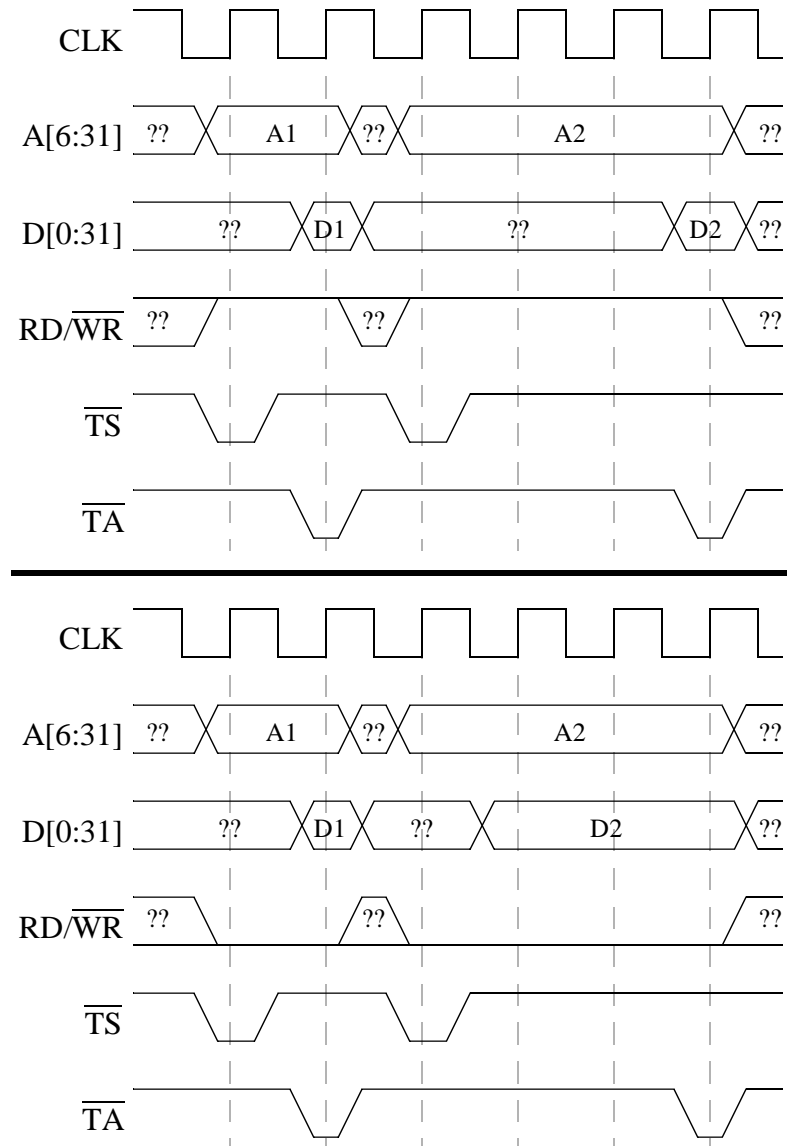


## MPC823 Bus

See Chapter 13 of the data book.

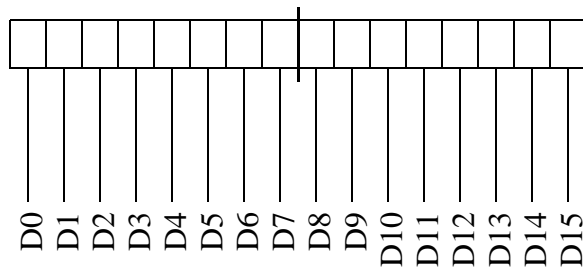
- semi-synchronous
- 32-bit data bus D[0:31]
- 26-bit address bus A[6:31]
  - A[0:5] not sent off chip
  - on-chip peripherals still see 32 address bits
- basic control lines:
  - $\overline{RD/\overline{WR}}$
  - $\overline{TS}$  (transfer start)
  - $\overline{TA}$  (transfer acknowledge)
  - $\overline{TEA}$  (transfer error acknowledge)
- key differences from generic semisync bus:
  - $\overline{TS}$  driven only on first cycle
  - for writes, data not valid until second cycle

## MPC823 Bus Examples



## Dealing with Wider Busses

- 32-bit data bus can read/write one word per transaction
- What about halfword or byte accesses?
- Example: 16-bit device on 16-bit bus:



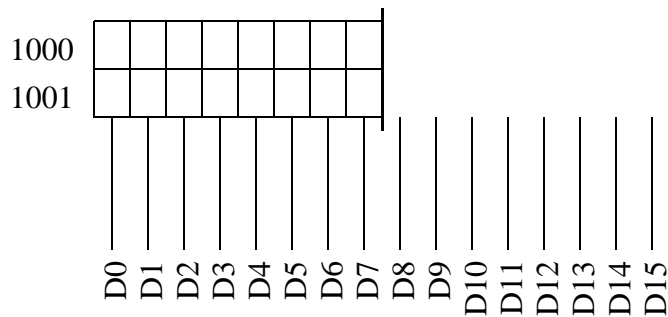
- Assume r4 has device address. What happens on:
  - `lhz r3, 0(r4)`
  - `lbz r3, 0(r4)`
  - `lbz r3, 1(r4)`
- What does this imply about where CPU expects data on data bus?

## Wider Busses (cont'd)

- What about `lhz r3, 1(r4)`?
- If all bus transactions are aligned, can we ignore least-significant address bit (A[31])?
- Most wider busses provide separate byte enable lines:
  - Moto 68000 (16-bit bus):  $\overline{LDS}$ ,  $\overline{UDS}$  (no address LSB)
  - 32-bit busses: replace low 2 addr bits w/4 byte enables
- MPC823 does not:
  - full byte address provided (incl. A[30] & A[31])
  - size (byte, halfword, word) encoded on two control lines (TSIZ[0-1], see Table 13-4)
  - equivalent, but messier
  - internal memory controller can generate byte enables & more for you

## Wide Busses and Narrow Devices

- An 8-bit device with two registers on a 16-bit bus:



- Why are only D0-D7 connected?
- What happens when we do `lbz r3, 1(r4)`?
- What address is the second register *really* at?

## Options for Narrow Devices

1. Live with it in software
  - not really that bad for I/O devices
  - good idea to check access size in hardware (use  $\overline{TEA}$ )
  - doesn't work for memory (esp. code)
2. Fix it in hardware
  - Requires two things:
    1. convert one access to multiple smaller accesses
    2. get data on correct data bus wires (data steering)
  - Some CPUs do this for you (called *dynamic bus sizing*)
    - slave indicates port size during transaction (extra inputs)
    - 486, Pentium will take care of #1 but not #2
    - 68020 takes care of #1 and #2
  - MPC823 bus does not do this, but internal memory controller does

## Bus Arbitration

Who gets to be master next?

MPC823 protocol is typical, see section 13.4.6. Three control signals are used:

- $\overline{\text{BR}}$  (Bus Request)
  - from master to arbiter: I want bus
  - one per master:  $\overline{\text{BR0}}$ ,  $\overline{\text{BR1}}$ , etc.
- $\overline{\text{BG}}$  (Bus Grant)
  - from arbiter to master: you can have it *next*
  - $\overline{\text{BG0}}$ ,  $\overline{\text{BG1}}$ , etc.
- $\overline{\text{BB}}$  (Bus Busy)
  - shared among masters (open-collector wired-OR)

## Arbitration Protocol

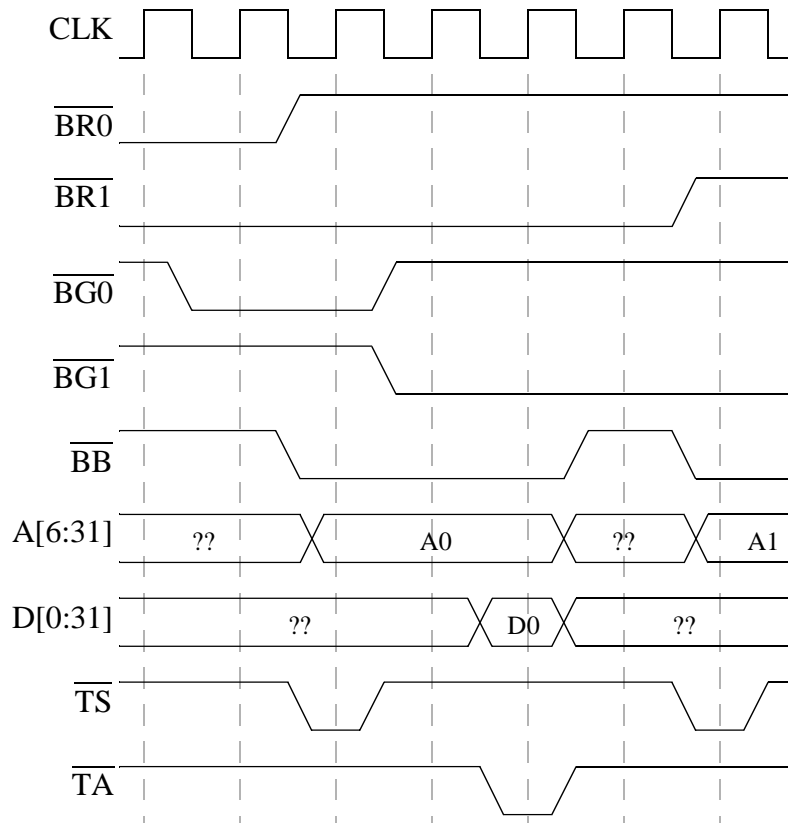
Arbiter looks at all  $\overline{\text{BR}}$  signals, asserts exactly one  $\overline{\text{BG}}$

- may use priorities or round-robin

To become master:

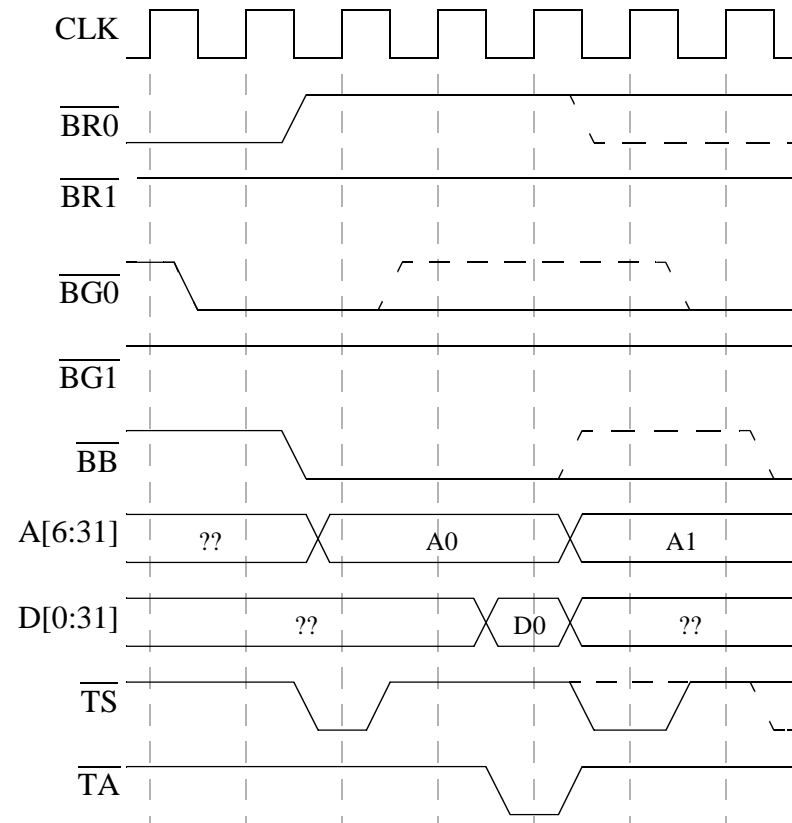
1. assert  $\overline{\text{BR}}$
2. wait for ( $\overline{\text{BG}}$  and not  $\overline{\text{BB}}$ )
3. deassert  $\overline{\text{BR}}$ , assert  $\overline{\text{BB}}$
4. do transaction
5. deassert  $\overline{\text{BB}}$

## Bus Arbitration Example



- arbitration overlaps with previous transaction
- still lose one cycle to switch masters

## Bus Parking



- “bus parking”: arbiter leaves  $\overline{BG}$  on last master if no  $\overline{BR}$ 
  - check  $\overline{BG}$  &  $\overline{!BB}$  same cycle as assert  $\overline{BR}$
  - save a cycle (or more) if no switch in masters
  - no need to wait on  $\overline{BB}$  if you’re driving it yourself