Bus Topics

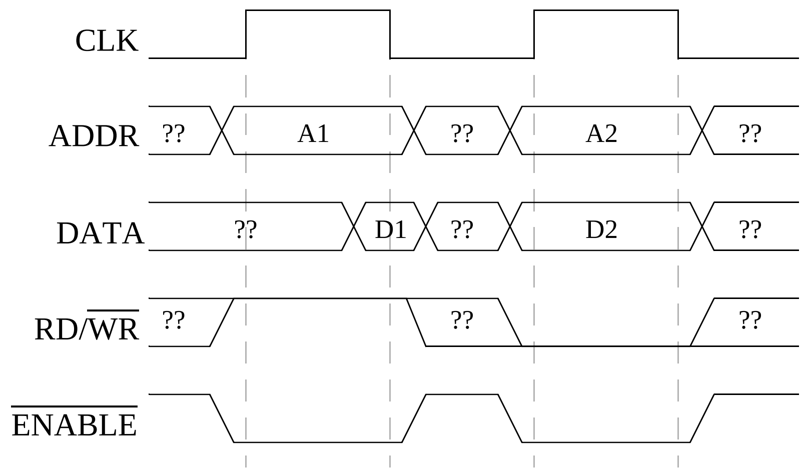
You should be familiar by now with the basic operation of the MPC823 bus. In this section, we will discuss alternative bus structures and advanced bus operation.

- Synchronization styles
- Arbitration: supporting multiple masters
- Burst transfers
- Pipelining
- Address/data multiplexing
- Split transactions

Synchronization: How do master & slave agree on when data is valid, or when transaction is over?

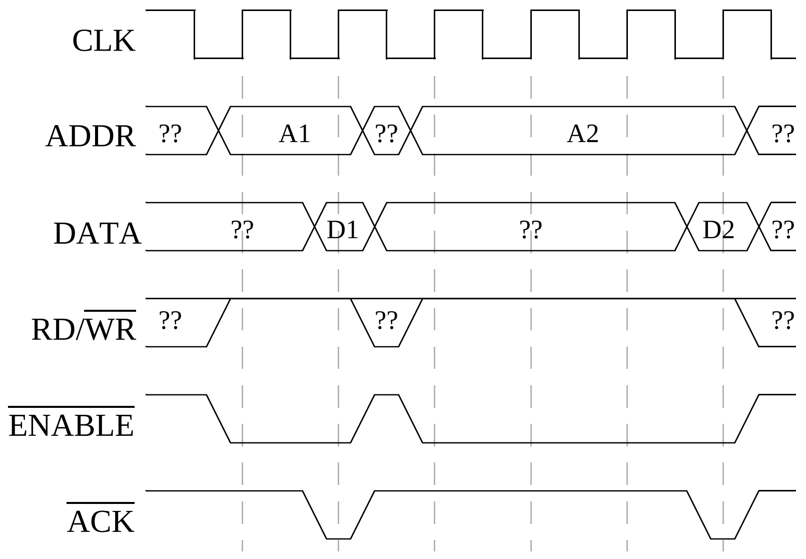
1. Synchronous ("fully" synchronous)
2. Semi-synchronous
3. Asynchronous

Synchronous Bus



- Everything synchronized to bus clock, every transaction takes one clock cycle
- All master outputs valid on rising edge of CLK, stay valid through falling edge of CLK; slave output (for read) valid by falling edge of CLK
- Setup & hold times part of bus specification
- Advantage:
- Disadvantages:

Semisynchronous Bus

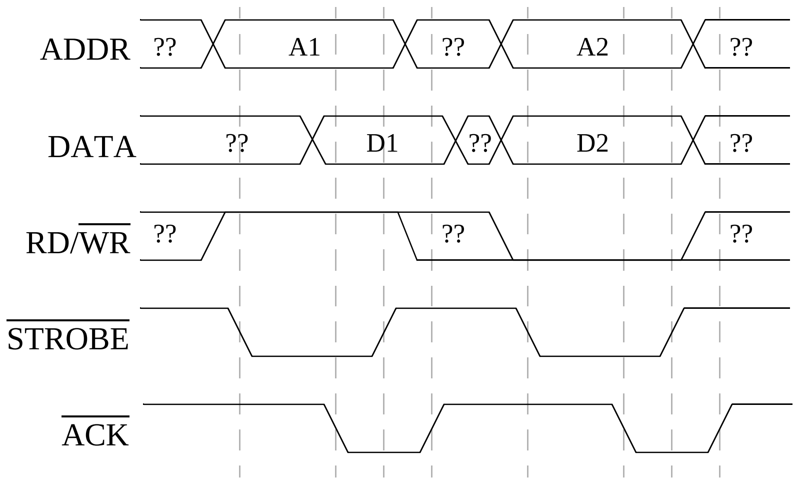


- Everything synchronized to bus clock, but transactions take *variable number* of cycles
- Slave asserts $\overline{ACK}$ to indicate that data is valid at clock edge (must be synchronized to clock)
- Control signal edges convey no timing information
- Minimum transaction usually multiple cycles (2 in example, on MPC823; 4 on 8086)
- Extra clock cycles between start and end (due to late $\overline{ACK}$) are called *wait states*

Semisync Bus (cont'd)

- Variation: replace $\overline{ACK}$ with $\overline{WAIT}$; assume minimum transaction time unless device asserts $\overline{WAIT}$
- $\overline{ACK}$ or $\overline{WAIT}$ can be driven by:
  - device itself (if it provides such a thing)
  - "wait state generator": dedicated counter/FSM logic, delays $\overline{ENABLE}$ by fixed number of cycles to generate $\overline{ACK}$ (based on device response time)

Asynchronous Bus



- No clock: all timing based on control signal edges:
    - $\overline{\text{STROBE}}$: master to slave valid signal (slave reads on falling edge)
    - $\overline{\text{ACK}}$: slave to master valid signal (master reads on falling edge)
- fully interlocked: no device timing dependencies

1. master asserts $\overline{\text{STROBE}}$ until it sees $\overline{\text{ACK}}$
2. slave asserts $\overline{\text{ACK}}$ until it sees $\overline{\text{STROBE}}$ deasserted

Asynchronous Bus (cont'd)

- M68000 can be operated in this fashion (slaves can ignore clock and use control signal edges only)
    - key: $\overline{\text{ACK}}$ is asynchronous; no setup time w.r.t. clock
    - synchronized internally to CPU
- partially interlocked:
    - bus specifies minimum, maximum pulse widths for $\overline{\text{STROBE}}$ & $\overline{\text{ACK}}$
    - relies on devices to meet spec, breaks if pulses too short or too long
    - PC parallel (printer) interface is like this
- Advantages:
- Disadvantages:

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

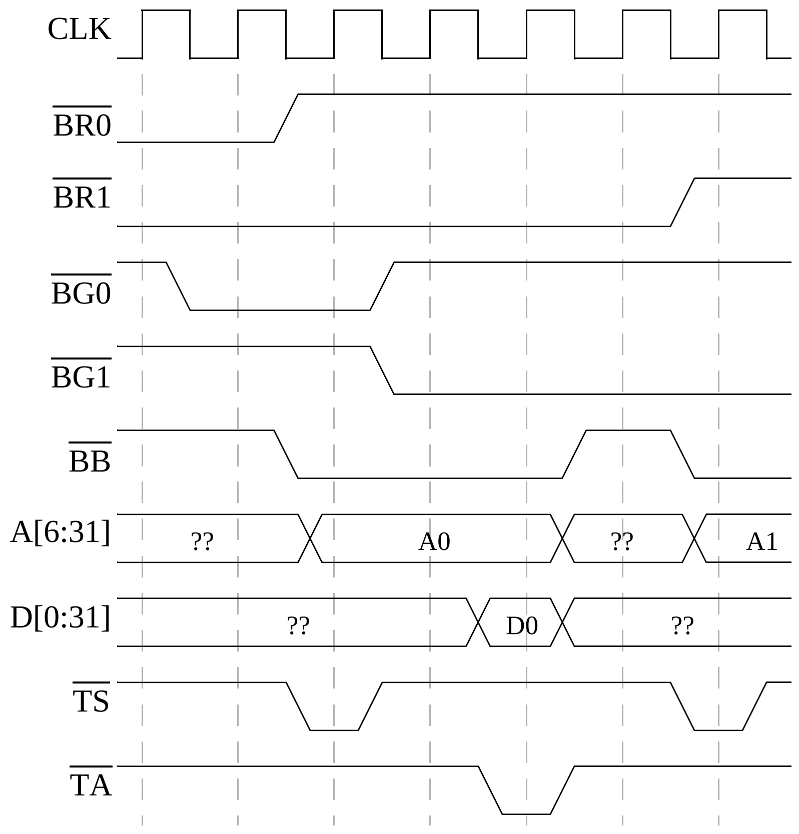
Arbiter looks at all $\overline{\text{BR}}$ signals, asserts exactly one $\overline{\text{BG}}$

- may use priorities or round-robin

To become master:

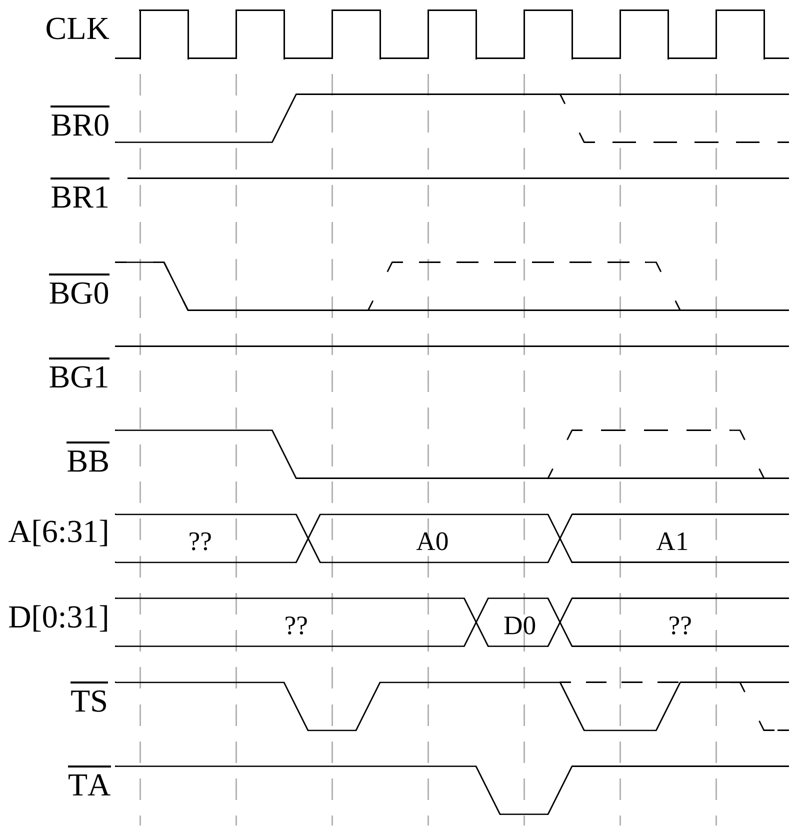
1. assert $\overline{\text{BR}}$
2. wait for ($\overline{\text{BG}}$ and not $\overline{\text{BB}}$)
3. deassert $\overline{\text{BR}}$, assert $\overline{\text{BB}}$
4. do transaction
5. deassert $\overline{\text{BB}}$

Bus Arbitration Example



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

Bus Parking



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

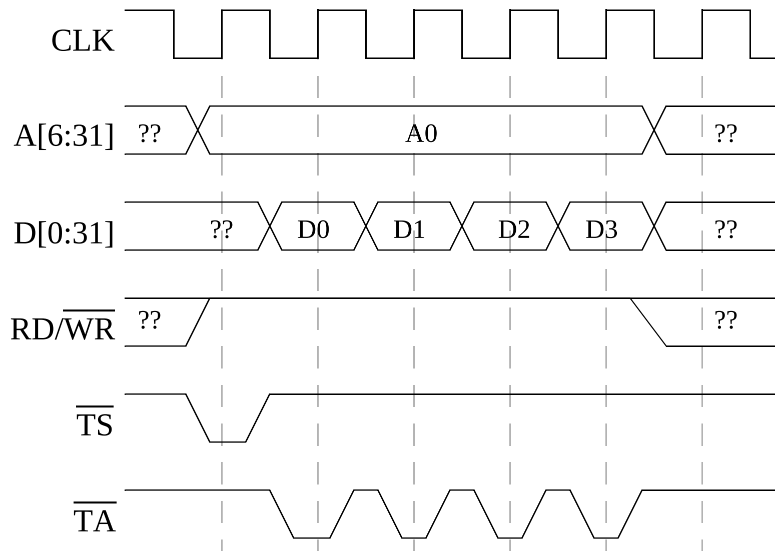
Advanced Bus Techniques

How to make busses work more efficiently or cheaply:

- Burst transfers
- Pipelining
- Address/data multiplexing
- Split transactions

Burst Transfers

- First cycle of every MPC823 bus transaction transfers address; minimum transaction is two cycles
- This limits data transfers to one word every other cycle
- We can beat this limit by transferring *multiple words* in one transaction using one address cycle:



(Some additional control signals are required, not shown in the diagram.)

- What fraction of bus cycles can we transfer data on now?

Burst Transfers (cont'd)

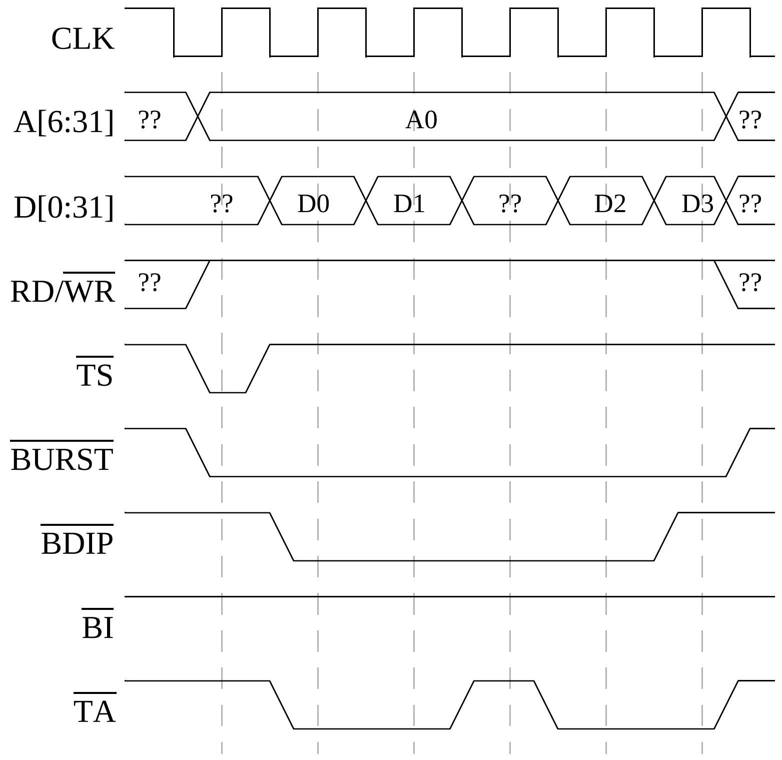
- What addresses are used for additional data words?

- Where do burst requests come from?

- How are burst transfers initiated?

- MPC823 has two primary signals:
  - $\overline{\text{BURST}}$: from master: I want to do burst
  - $\overline{\text{BI}}$: from slave: I can't do bursts (823 will convert to regular "single-beat" requests)
  - Only 16-byte (4-word) bursts supported, so no need to indicate specific size

MPC823 Burst Transfer Example



- Slave can hold $\overline{\text{TA}}$ asserted on adjacent cycles (no need to deassert as in previous example)

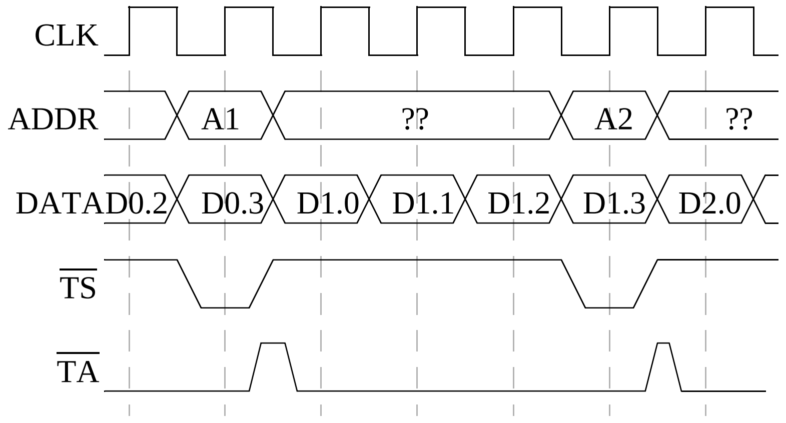
- Slave can still insert wait states by not asserting $\overline{\text{TA}}$

- $\overline{\text{BDIP}}$: from master: I still expect another word (simplifies slave control)

Pipelining

Burst transfers let us transfer data on *n* out of every *n*+1 cycles. How can we do better?

- Notice that data bus is not used during address cycle, and address bus is not really needed during data cycles.
- *Overlap* address cycle of each transaction with data cycles of previous transaction.



- This is called transaction *pipelining*.
- In general, any two phases of a transaction that use a separate set of physical signals (wires) can be pipelined.

Pipelining (cont'd)

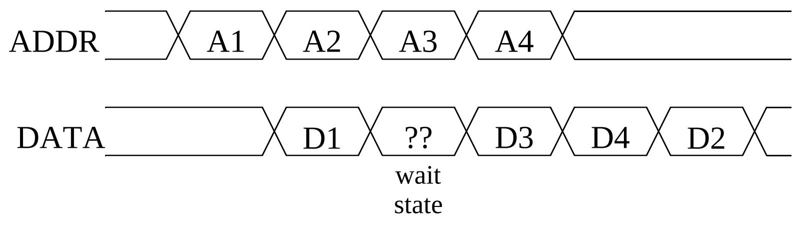
- The MPC823 bus does not support address/data pipelining, but (like most busses) does pipeline the arbitration phase.
- The P6 (Pentium Pro/Pentium II) bus has five pipeline stages (arbitration, request (address), error, snoop, and response/data).
  - By the time transaction *n* is transferring data, transaction *n*+3 may be issuing its request.
  - minimum transaction length = 9 cycles
  - can start a new transaction every 3 cycles
  - with burst transfers (4x 64 bits = 32 bytes), can keep data bus busy 100% of the time
- Disadvantages of pipelining?

Address/Data Multiplexing

- Pipelining takes advantage of the fact that one transaction does not use both address & data busses at the same time.
- If the goal is to reduce cost (not increase performance), how might we exploit this fact differently?

Split Transactions

- Back to high performance... what happens on a pipelined bus when a transaction requires wait states?
- A *split-transaction* bus splits each transaction into two largely independent parts, a request (address part) and a reply (data part for reads). Replies may appear in any order.



- Just as masters arbitrate to initiate a transaction on the address bus, slaves must now arbitrate to put their reply on the data bus.
- Advantage:
- Disadvantages: