Busses

Move from here to there

Bus Architecture

- A bus is an interconnection structure between units
  - Such as: CPU, memory, I/O interfaces
  - Even between I/O interfaces and devices
    - Example the USB bus
- There are various bus architectures
  - Design choices relate to physical, electrical, signaling and protocol issues
- A few points to consider ...
- Busses may be parallel or serial, as discussed earlier

Bus Architecture

- A few points to consider …
  - Busses may be parallel or serial, as discussed earlier
  - Many busses include the possibility that >2 units may be connected. Such busses can be “multidrop” (all components connected to the same lines and electrically parallel), or daisy chained. Hubs are also possible, as in USB. Serial busses are more likely to be daisy chained.
- Busses may be:
  - Synchronous ⟵ transfers timed by a clock
  - Asynchronous ⟵ transfers synchronized using handshake signals

Synchronous Bus

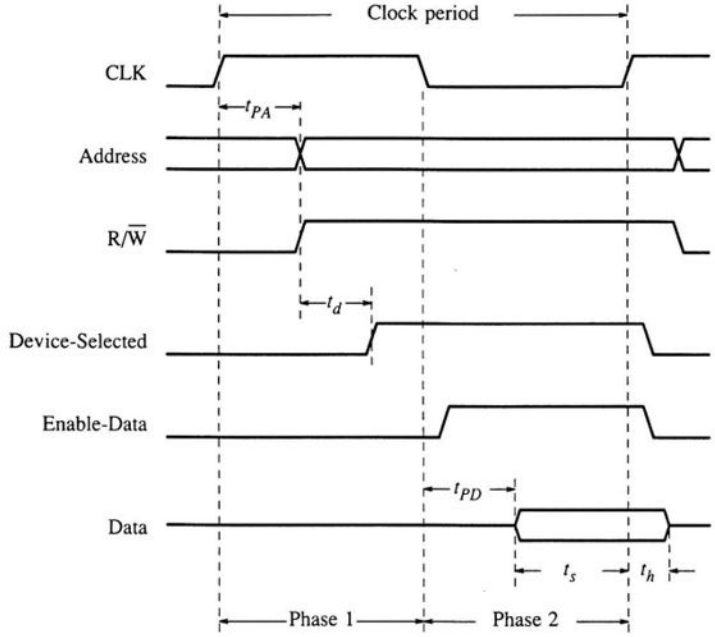


Figure 6.8 A detailed timing diagram for an input operation.

Microcomputer Structures, Z.G. Vranesic & S.G. Zaky, Holt, Rinehart & Winston, 1989

Asynchronous Bus

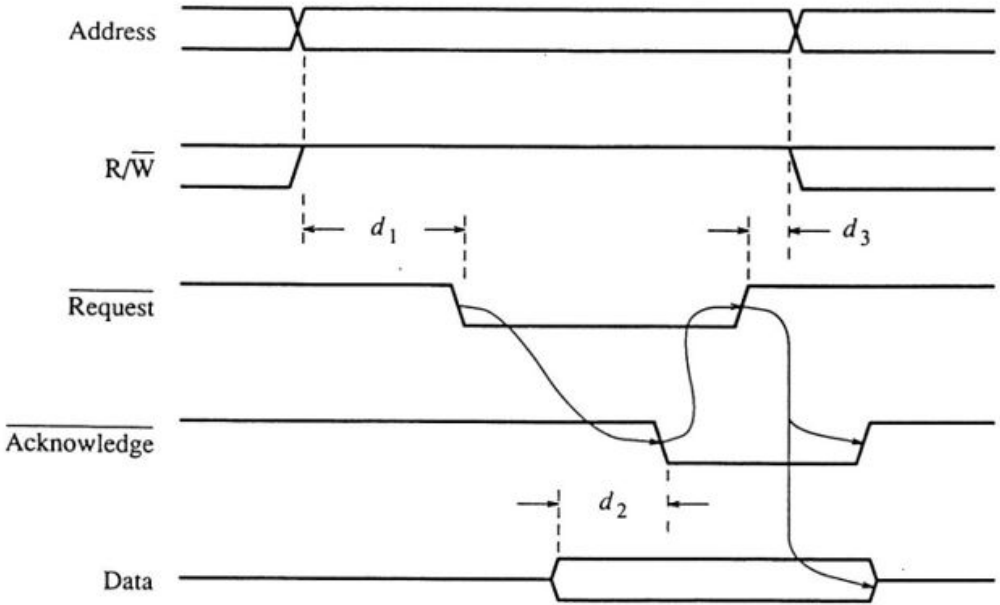


Figure 6.47 Timing of an input operation on an asynchronous bus.

Microcomputer Structures, Z.G. Vranesic & S.G. Zaky, Holt, Rinehart & Winston, 1989

Bus Architecture

- Synchronous busses require that the clock be slow enough to accommodate worst case transfer times
    - Example: due to worst case propagation delay
- On some busses, a unit can assert a “wait” signal to claim extra bus cycles for its transfers
- Asynchronous busses sidestep these issues.
- The memory bus may be separate (logically or physically) from the I/O bus, as discussed earlier

Bus Architecture

- Bus lines may be dedicated (one purpose), or time division multiplexed
  - Example: a set of lines may carry an address during the first bus cycle, and then the data during the next bus cycle
  - This saves pins and bus lines, but at the expense of protocol and buffering complexity
- A bus may support more than one bus master
  - In this case, bus arbitration must be performed in either a centralized or distributed fashion
  - In a centralized scheme, a bus controller (arbiter) grants control of the bus to one master at a time.

Bus Transfer Types

- A bus may support multiple transfer types, e.g.
  - Interrupt acknowledge
  - Memory read/write
  - I/O read/write
  - Atomic read-modify-write operation
  - “Burst” – a read or write operation where the address is transferred in the first clock cycle, and then a sequence of data values is transferred in subsequent clock cycles

Connecting Busses

- For performance reasons, systems may employ a multiple-bus hierarchy. “Bridges” connect the busses. Some advantages:
  - Bridges decouple bus architecture from CPU architecture
  - High priority devices can be “closer” to the CPU and memory
  - Busses can operate at different speeds
  - Bus lengths can be kept short – reduces propagation delay
  - Reduces bus contention – bridges can buffer data and then forward it when possible

Sample PC Implementation

- CPU may be connected to “northbridge” via its “front side bus” (address and data bus)
    - Possibly through the “back side bus” to a cache
- Northbridge may be connected to memory, graphics card, PCI bus and “southbridge”
    - The latter possibly via the PCI bus
- Southbridge may be connected to IDE, USB bus, etc.
- Other functionality may be provided onboard the bridges

PCI Bus

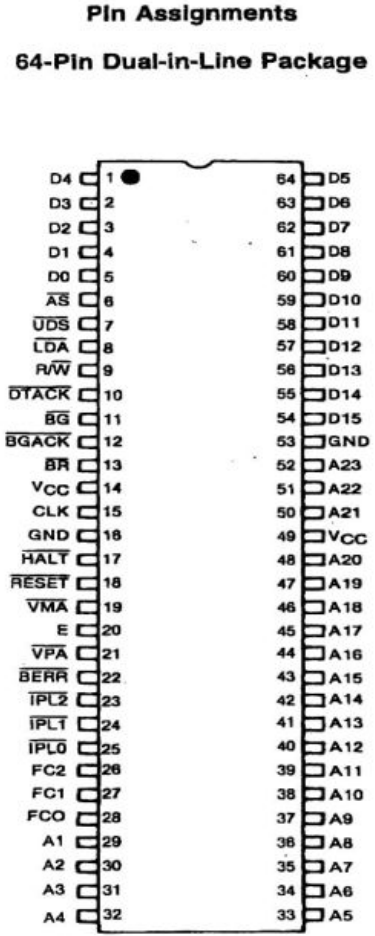
- Parallel (32 or 64 bit)
- Synchronous
- Logically separates memory from I/O addressing
- Address/data lines multiplexed
- Centralized arbitration
    - Each connected unit supports bus request/grant signals
- Supports many transfer types, including bursts
- Includes support for cache protocols
- PCI-E Bus is a serial PCI Bus

68000 Bus

- Parallel
- Asynchronous
- Memory mapped I/O
- Dedicated address/data lines
- Supports multiple bus masters
    - CPU can grant bus to another unit
    - Example: for DMA

68000 Bus Pin Assignment

D0-16	i/o	data bus
A1-23	o	address bus (no A0)
FC0-2	o	function codes: output CPU state and transfer type
R/W	o	read/write signal
UDS/LDS	o	upper/lower data strobe: indicates if transfer is odd byte, even byte or word (serves as A0)
AS	o	address strobe: “address valid” handshake signal
DTACK	i	data transfer acknowledged: the other handshake signal
BERR	i	bus error



68000 Bus

- UDS/LDS explain address errors. They also explain why 8-bit I/O interface registers are mapped either at odd or even addresses, but not both
- Some pins are three-state – why?
    - They are not continuously driven to a specific state so can be allowed to float
- Some signals are active low – why?
    - They need to be driven to a specific value (state) and it is electrically easier to use an open collector to drive this state

68000 Read Cycle Details

- Note similarity to vector # transfer
- Write cycle is similar

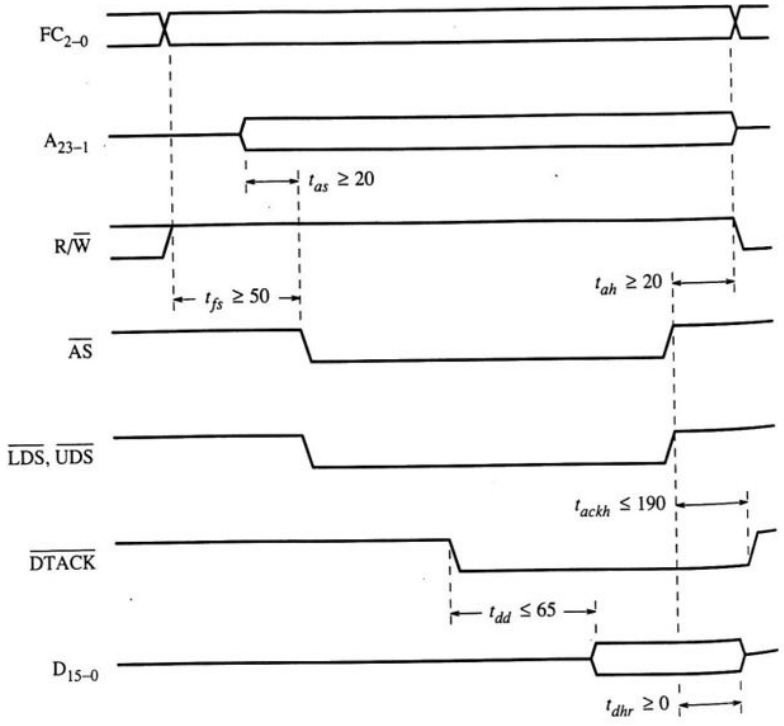


Figure 6.53 Timing of a read operation on the 68000 bus.

68000 - Sequence of States

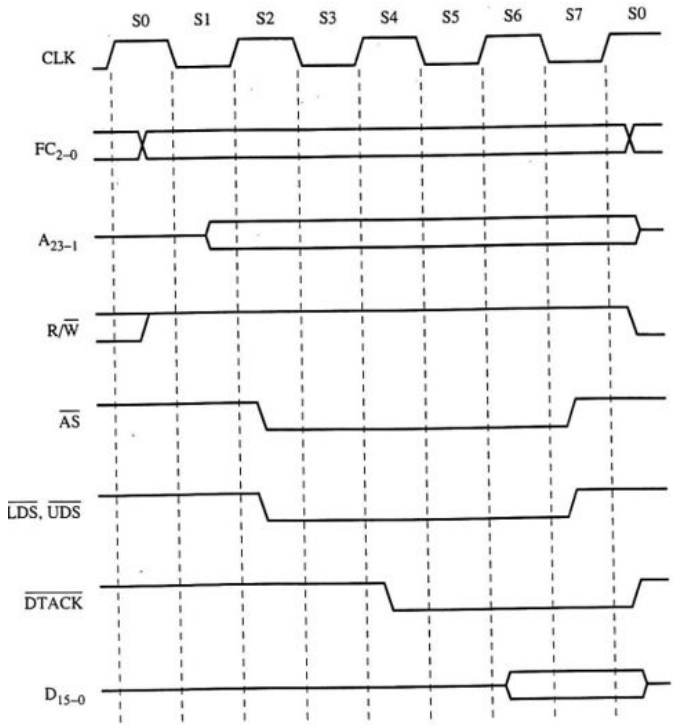


Figure 6.55 Sequence of microprocessor states during a read operation.

Table 6.4 SEQUENCE OF STATES OF THE 68000 MICROPROCESSOR DURING A READ OPERATION

State	Action
S0	Transmit the function code on lines FC_{2-0}, and set R/$\overline{W}$ to 1.
S1	Place address on the address lines.
S2	Activate $\overline{AS}$, $\overline{LDS}$ and $\overline{UDS}$.
S3	No action.
S4	Sample $\overline{DTACK}$ — remain in state S4 until $\overline{DTACK}$ becomes active.
S5	No action.
S6	Load input data into input buffer on the next falling edge of CLK.
S7	Remove address and strobe signals.