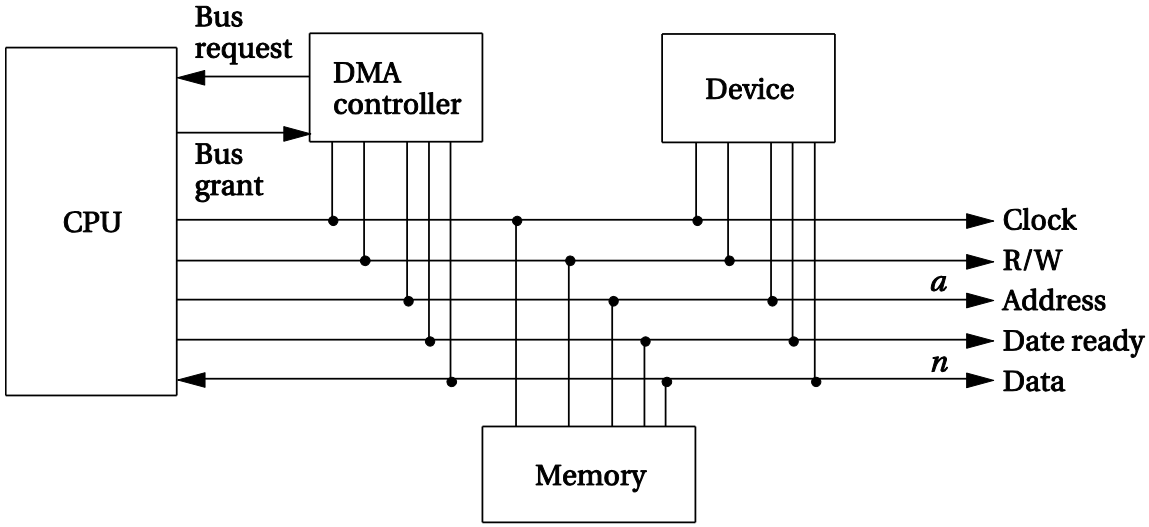
DMA (Direct Memory Access)

Direct memory access (DMA) performs data transfers without executing instructions.

- CPU sets up transfer.
- DMA engine fetches, writes.

DMA controller is a separate unit.



Bus Grant

By default, CPU is bus master and initiates transfers.

DMA must become bus master to perform its work.

- CPU can’t use bus while DMA operates.

Bus mastership protocol:

- Bus request.
- Bus grant.

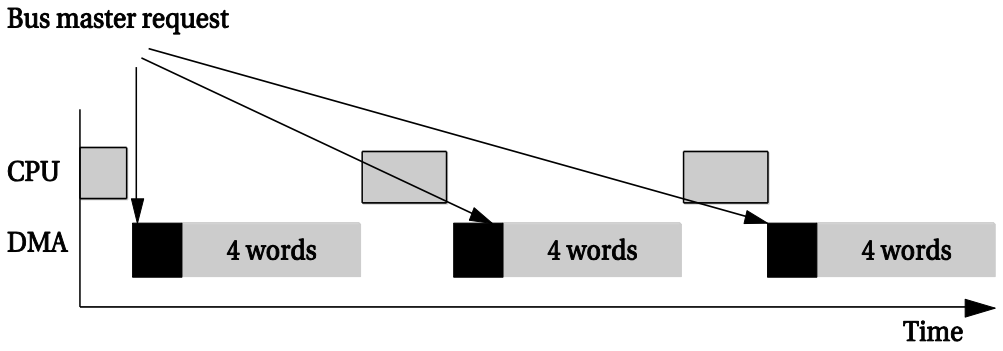
DMA Operation

CPU sets DMA registers for start address, length.

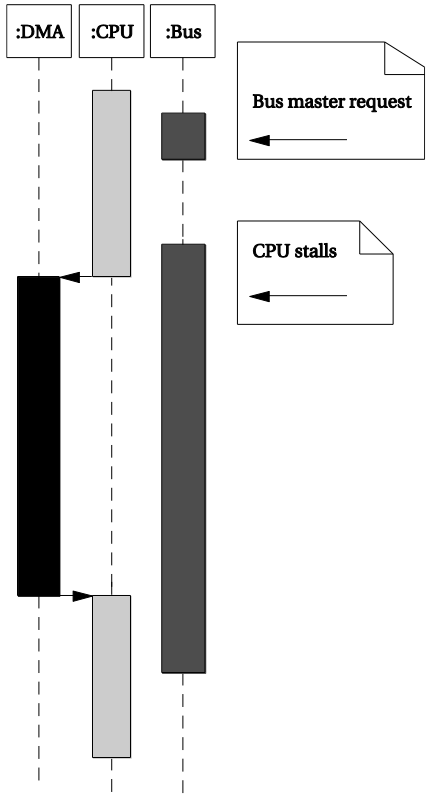
DMA status register controls the unit.

Once DMA is bus master, it transfers automatically.

- May run continuously until complete.
- May use every n^{th} bus cycle.



Bus Transfer – Sequence Diagram

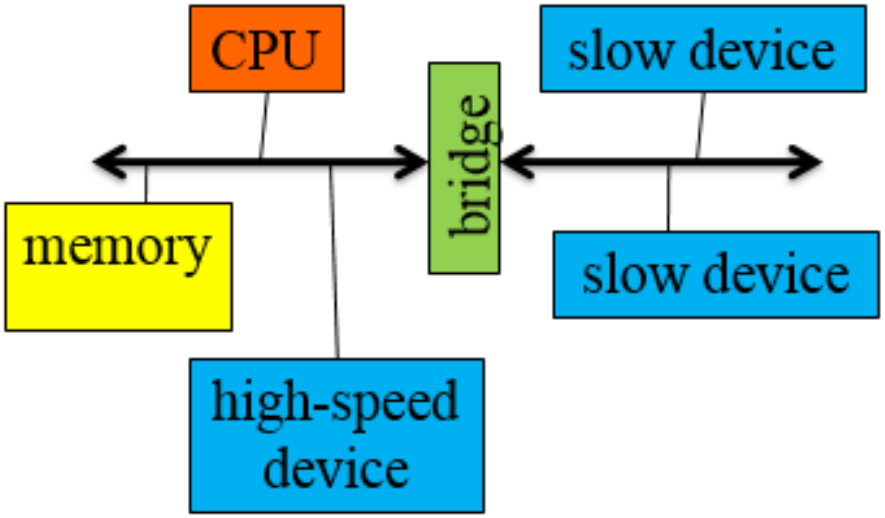


System Bus Configurations

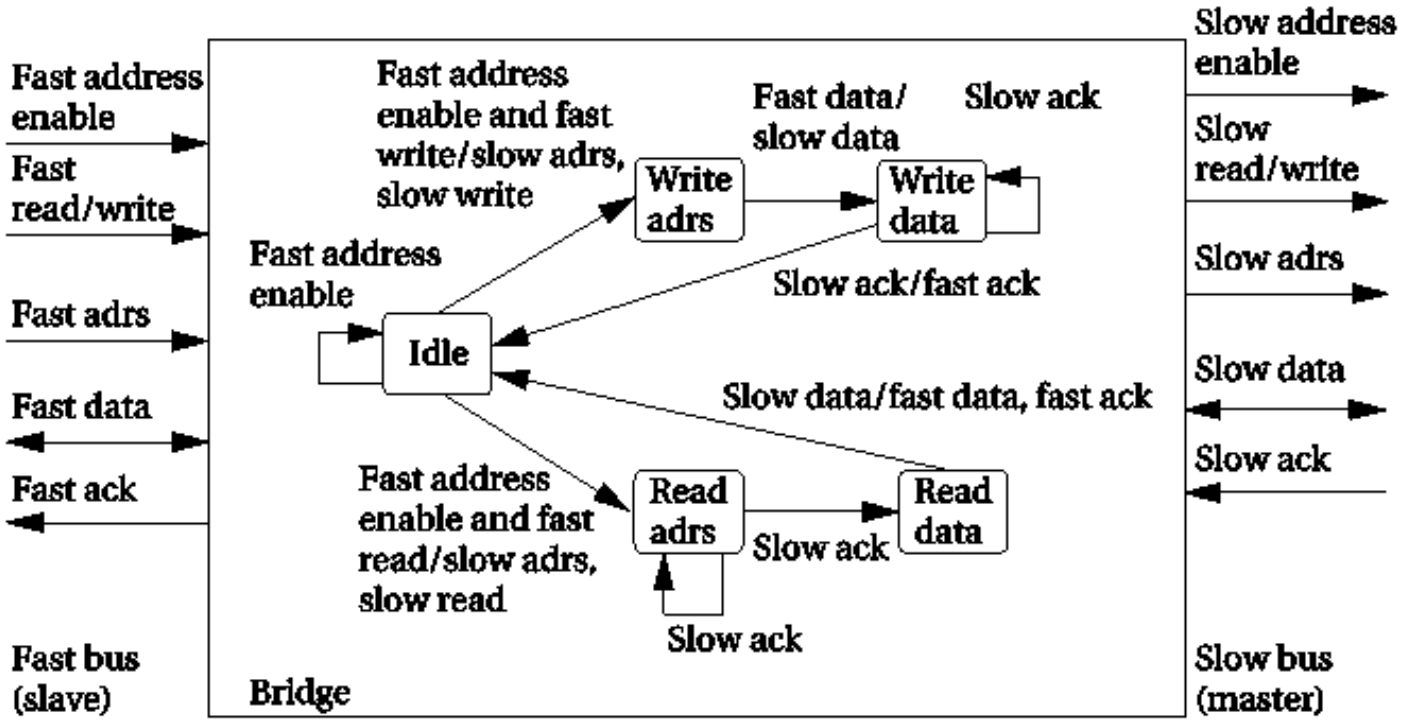
Multiple busses allow parallelism:

- Slow devices on one bus.
- Fast devices on separate bus.

A bridge connects two busses



Bridge State Diagram



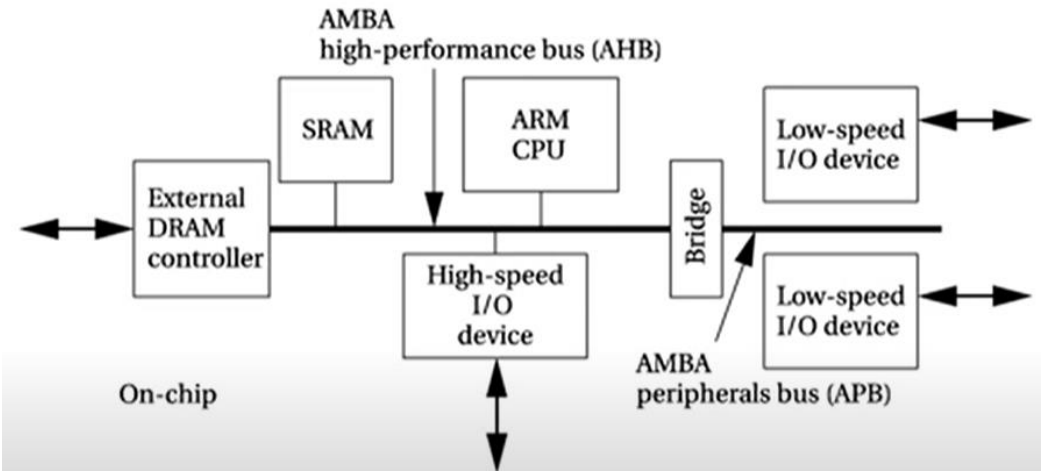
ARM Bus Architecture -- AMBA

Two varieties:

- AHB is high-performance.
- APB is lower-speed, lower cost.

AHB supports pipelining, burst transfers, split transactions, multiple bus masters.

All devices are slaves on APB.



Platform-level performance

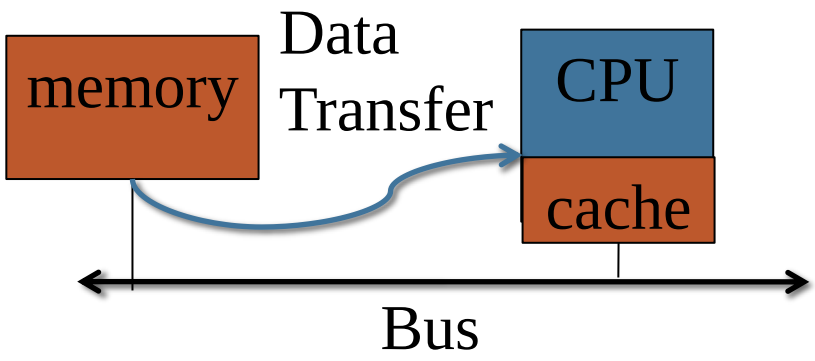
- Bus-based systems add another layer of complication to performance analysis Platform-level performance involves much more than the CPU.
- We often focus on the CPU because it processes instructions, but any part of the system can affect total system performance.
- We want to move data from memory to the CPU to process it. To get the data from memory to the CPU we must:
  - Read from the memory;
  - Transfer over the bus to the cache;
  - Transfer from the cache to the CPU.

System-level performance analysis

Multiple components contribute to overall performance of an embedded or a computer system.

Performance depends on all the elements of the system:

- CPU.
- Cache.
- Main memory.
- I/O device.
- Bus.



Platform-level data flows and performance

Bandwidth as performance

- The most basic measure of performance we are interested in is bandwidth -the rate at which we can move data.
- Ultimately, if we are interested in real-time performance, we are interested in real-time performance measured in seconds.
- But often the simplest way to measure performance is in units of clock cycles.
- However, different parts of the system will run at different clock rates.
- We have to make sure that we apply the right clock rate to each part of the performance estimate when we convert from clock cycles to seconds.

Bandwidth as performance

Bandwidth applies to several components:

- Memory.
- Bus.
- CPU fetches.

Different parts of the system run at different clock rates.

Different components may have different widths (bus, memory).

Bandwidth and data transfers

- Bandwidth questions often come up when we are transferring large blocks of data. For simplicity, let us start by considering the bandwidth provided by only one system component, the bus.
- Consider an image of 320x240 pixel video frame
  - Each pixel has 3 byte of information (3 primary colors)
  - This gives a video frame: 320 x 240 x 3 = 230,400 bytes.
  - We want to check if we can push one frame through the system within 1/30 sec.
- If the transfer rate is 1 Mbps;
- Transfer 1 byte/μsec, 0.23 sec per frame
  - Too slow.
- Solution :Increase bandwidth
  - Increase bus width.
  - Increase bus clock rate.

Bus bandwidth

T: # bus cycles for transfer.

P: time/bus cycle **or** bus clock period.

Total time for transfer:

- t = TP.

D: Data Payload Time

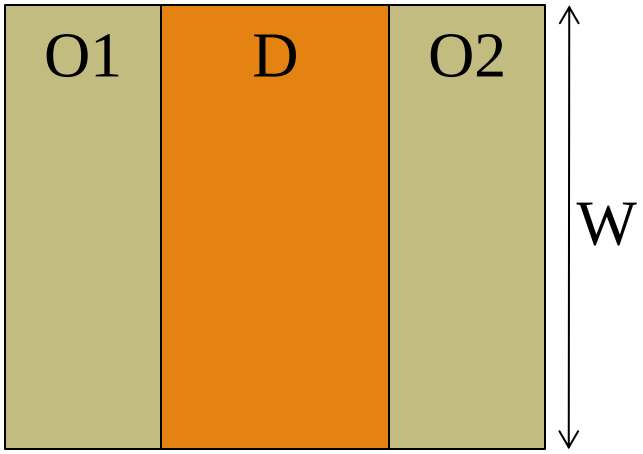
[in (clock cycles – waits)/clock cycles].

O1+O2 = overhead,O

W is the width of the bus

N words

D is the number of clock cycles needed to transfer the data.



$$T_{basic}(N) = (D+O)N/W$$

Times and data volumes in a basic bus transfer

Bus burst transfer bandwidth

T: # bus cycles.

P: time/bus cycle.

Total time for transfer:

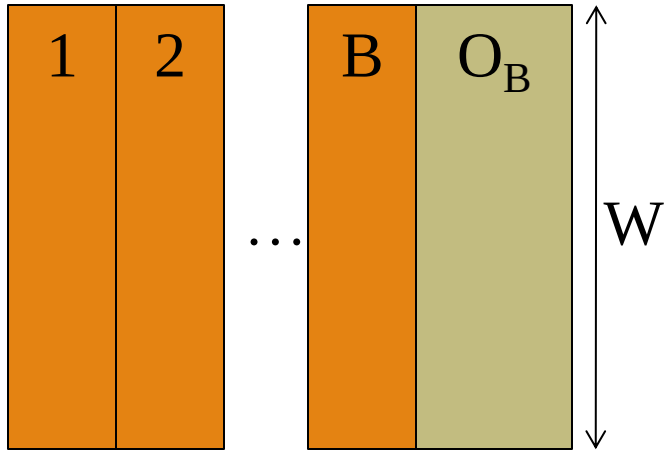
- t = TP.

B: Burst transaction length

D: data payload time.

O1+O2 = overhead, O_B

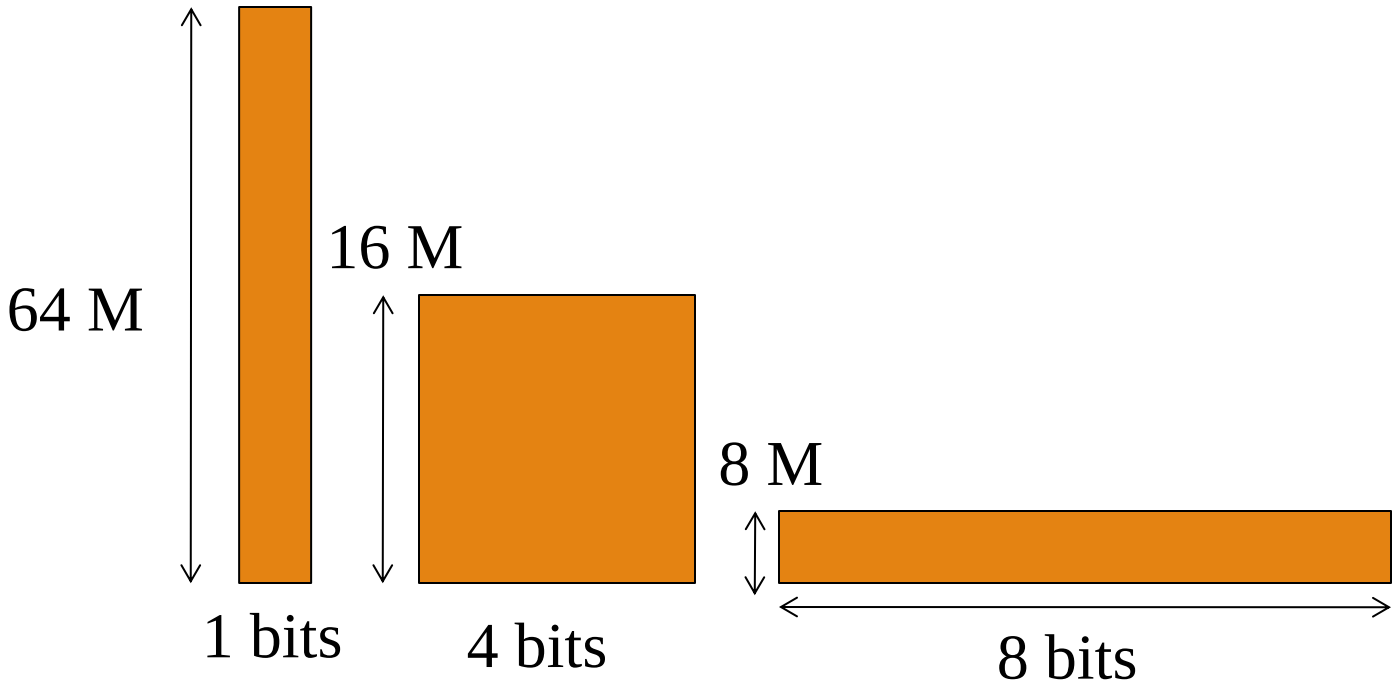
If burst transfer is allowed more than one bus clock cycle is allowed for data transfer. If B is the burst transaction length then the overhead occurs once per bus burst transaction.



$$T_{burst}(N) = (BD+O_B)N/(BW)$$

Times and data volumes in a burst bus transfer

Memory aspect ratios



Memory access times

Memory component access times comes from chip data sheet.

- Page modes allow faster access for successive transfers on same page.

Sometimes data doesn’t fit naturally into physical words

Bus performance bottlenecks

We want to transfer data between the CPU and the memory over the bus. We need to be able to read an HDTV 1920 × 1080, 3 bytes per pixel video frame into the CPU at the rate of 30 frames/s, for a total of 6.2 MB/s.

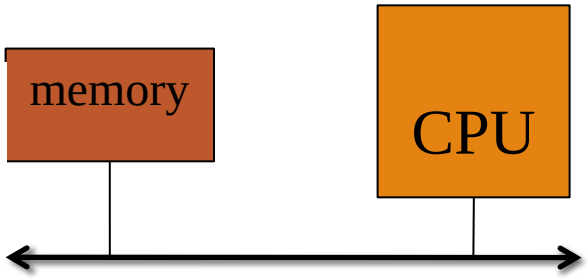
Let us assume that the bus has a **100-MHz** clock rate (period of 10 ns) and **is 2-byte wide**,

with D = 1 and 0 = 3. The 2-byte bus allows us to cut the number of bus operations in half.

This gives a total transfer time of

$$T_{basic(1920x1080)} = (3 + 1).(\frac{6.2\ x\ 10^6}{2}) = 12.4\ x\ 10^6 \text{ Cycles}$$

$$t_{basic} = T_{basic}\ P = 0.124s$$



The access time for this memory is not fast enough for our application. As an alternative, consider a memory that provides a burst mode with **B = 4** and is **2-byte wide**. For this memory, **D = 1** and **O = 4** and assume that it runs at the same clock speed: 10 ns. Then

$$T_{basic\ (1920x1080)} = \left(\frac{6.2\ x\ 10^6/2}{4}\right).(4x1 + 4) = 6.2\ x\ 10^6 \text{ Cycles}$$

$$T_{basic} = T_{basic}\ P = 0.062s$$

Solutions

If bus is the bottleneck, then possible solutions

- Increase bus clock rate (if possible)
- Increase bus width

If memory is the performance bottleneck

- Increase memory banks (adding parallelism)
- Use faster memory chips and or technology

Parallelism

Speed things up by running several units at once.

DMA provides parallelism if CPU doesn't need the bus:

- DMA + bus.
- CPU.