

- mechanism of communication between CPU, mem, dev
 - wires and a protocol : clk, control, addr, data lines
 - throughput** (datawidth * bits) - overhead

Types

- serial - 1 bit conveyed at a time (UART, I2C, SPI, USB)
- parallel - N bits (Memory, PCI)
- on-chip - connecting cores within a dev (SoC/FPGA)
- off-chip - connecting modules on PCB
- synchronous - separate clk and data signals
- asynchronous - no clk signal (dev provides own clk)

Controller/Device Config

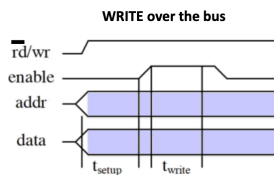
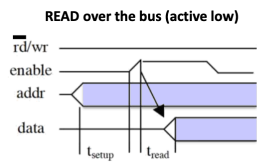
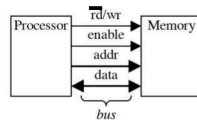
- bus operates using **controller/device** config
 - controller handles requests, ACK
 - bus can have multiple controllers (needs arbitration on who controls the bus)

Bus Protocol

- bus protocol defines what controller/dev should do
 - controller initiates data transfer
 - dev responds to init request (processor/peripheral/mem)
- protocol defines
 - data direction** receive(RX) transmit (TX)
 - how data shares wires
 - addr indicates data src and dest

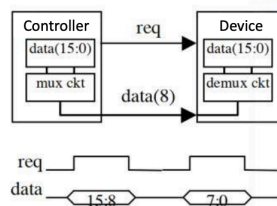
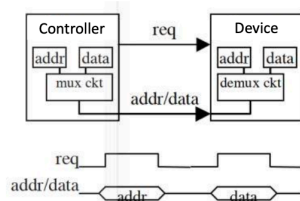
Protocol is described by a timing diagram

- Control lines -> high/low
- Data/address -> valid/not valid



Bus Splitting

- to prevent bus from getting very large, share line between addr and data OR splitting data across multiple transfers



Bus Control Methods

The controller knows that device is done via...

Strobe

- controller asserts req
- dev puts data on bus within deadline
- controller receives data and de-asserts req
- dev ready for next req

Pro/Con

- when dev response time is known, can avoid the extra line for ACK (can set DDL)
- can complicate logic of controller, since it waits a fixed number of cycles internally

Handshake

- controller asserts req
 - dev puts data on bus and ACK the data is ready. controller reads data
 - controller de-asserts req
 - dev stops transmitting data and de-asserts ACK (requires **extra line for ACK**)
 - both ready for next req
- UART is handshake

Pro/Con

- extra line for ACK
- may be slower than strobe - requires controller to detect ACK

Burst Transfers


Motivation: Reading a series of consecutive memory locations; avoid putting each on the line

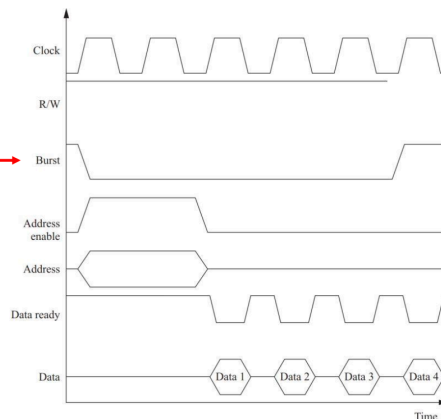
- can read from a contiguous sequence of addr at once
- start addr given by CPU; **bursts of data** are received from **n** successive locations

Pro/Con

- uses resources efficiently (continuous stream without interrupts)
- sending consecutive blocks = **maximize throughput reduces latency**
- however, one dev can occupy bus for long period, blocking another dev's access to bus

Burst transfers

- In this example, there is a dedicated **burst** signal. 
- To indicate a burst transfer, this is held low.
- Burst line is released after data3 in order to stop receiving data at the end of data4



Bus Arbitration

- when multiple dev simultaneously request the bus, need to decide who controls the bus
- handled by **arbiter** which receives all the req and grants req based on **arbitration scheme**

Priority-based Arbitration

- each dev sends *separate req* to hardware arbiter, which schedules requests
 - **fixed priority/round-robin**
 - RR - dev used bus most recently gets lowest priority
 - prevents overusing the bus but complicates arbiter design

- a system with 3 peripherals
- different enable signal for each device (the address accumulates)

- for N devices, solve N truth tables to design hardware

Optimized Scheme

- look at top few bits to know which device is being addressed
- PRO - simpler address decoder (& directly use lower bits to address register inside dev)
- CON - wastes addresses