Meta-principles:

- Holism: Hardware design is highly coupled and global in nature - design choice in one place have implications to the rest of the system
- Duality: Software and Hardware is like Yin and Yang - both adhere to “Tao”. Many design patterns and solutions in software is just as applicable, and in fact actually used, in hardware as well (subject to suitable modification to suit the unique constraints of hardware)

Bus (The all important but mysteriously missing topic in undergrad curriculum):

- Basically like the Ethernet at Layer 2: we want to have a simple, homogeneous medium of communication through shared access.
- Problem of 1) coordinating action and 2) avoiding conflict.
- Require a somewhat exotic hardware primitive to make it work: The Tristate buffer
    - Start with a buffer (abstractly a single input single output, identity function gate. It’s not trivial because it can restore electrical strength to a signal that may have attenuated due to, say, fan-out)
    - Modify it by adding a second input. If this “enable” switch is off, then the output is in a “high-impedance” state denoted Z, which is a third state in addition to the usual true/false value. (This is another instance of leaky abstraction, or where the original boolean abstraction turns out to be inadequate due to the underlying electrical model) This state is basically like having the wire physically cut off, thus isolating any electrical signal between the two sides.
- We can then construct a “bidirectional buffer”:
- A bus is in general a bunch of elements connected together via wire, in a star topology.
    - Connections are gated by a bidirectional buffer (controlled by respective element) before going into each element
    - Usually depicted in either of these ways:
- Actually it is parallelized: we have the data bus, address bus and control bus. Each bus can have different number of bits (i.e. number of wires in parallel)
- Want a simple model of operation: everything controlled by a single element, called the Master => The rest are Slaves.
    - Analogy to distributed system/network: a simple solution to the coordination problem is to centralize control. Tradeoff is ease of implementation/understanding/less complexity vs less robustness/performance in some case.
    - How it works: Master send signal via the control bus to the slaves to control them. All slave must follow commands from master and open/close the buffer in the read/write direction for address and data bus. Master is responsible for making the appropriate commands, so as to ensure that when sending/receiving, only the intended sender/receiver, and no one else, has the buffer open in the right direction. Obvious that this centralized scheme “work”.
- We still need a protocol.

- More complicated use case: what if we want to be able to change master/hand off the right-to-control on the fly?
    - Add a protocol for granting control.
    - Usually called Bus Arbitration due to the possibility of conflict again (What if more than one element raise a request to become master at the same time?)

The big picture:
(See the figure in https://en.wikipedia.org/wiki/Front-side_bus)
A bunch of VLSI (Very large scale integration) chips connected by various bus. Sort of like a computer network. Uses a hierarchical organization due to difference in natural/most suitable speed for various parts. Trend is to increasingly integrate things so for e.g. Memory controller are now on the same chip as CPU etc.
Protocol used for bus depends on its speed. Internal bus that are connected sort of directly to CPU => high speed custom/proprietary protocol (e.g. HyperTransport), though in the old days the “simplest thing that could possible work”, i.e. a naive memory access protocol should be okay.