Datapath Design, Coding Standards, and Lab 2

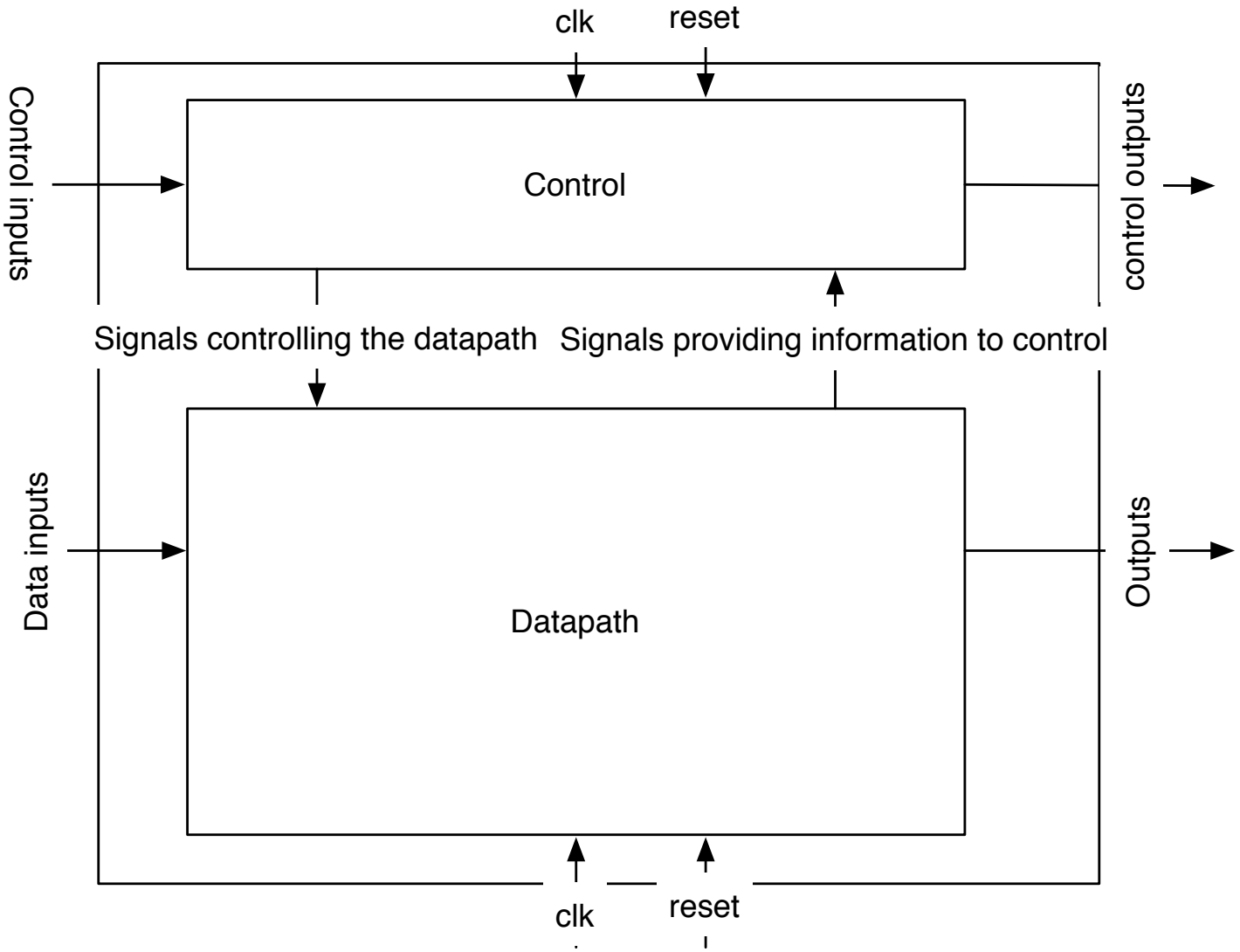
Separating Control From Data

- The datapath is where data moves from place to place.
  - Computation happens in the datapath
  - No decisions are made here.
  - Things you should find in a datapath
    - Muxes
    - Registers
    - ALUs
    - Wide busses (34 bits for data. 17 bits for instructions)
    - These components are physically large
    - In a real machine, their spatial relationship are important.
  - Mostly about wiring things up.

Separating Control From Data

- Control is where decisions are made
  - Things you will there
    - State machines
    - Random lots of complex logic
    - Little state (maybe just a single register)
  - Spatial relationships are harder to reason about or exploit.
- Because they are qualitatively so different, we will use different coding styles for each.
  - These are best practices from people who build real chips.
  - Following them will save you lots of pain
  - If you don't follow them, and you have problem, the TAs and I will tell you to go fix the coding style issues first.

Basic Design



Separating Design from Implementation

- As you will learn, debugging hardware is slow
- Design first
  - Draw your schematic in complete detail.
  - Signal names and everything.
  - Design the state machine for your control
  - Write out the truth tables for your control signals.
- The implement
  - Our coding standards are recipe for implementing datapath and control.
  - Writing Verilog is really just about translating your design into Verilog.
  - It should be almost completely mechanical.

Designing the Datapath

- Designing datapaths is easier than it may seem.
- Design
  - You start with a specification of the algorithm your circuit should implement
  - Figure out what operations need to be performed on the data and how data will flow between operations
  - Draw the schematic
  - Remember: the datapath does not make decisions.
    - It generates data needed to make the decisions
    - It provides the flexibility implement decisions that the control might make.
- Implement
  - Instantiate those components and connect them with wires.
  - Test.

Example: Greatest Common Devisor

- See second set of slides from Arvind.
- The code in the slides is buggy.
- The source code for a correct implementation is available on the course web site.

Datapath Coding Standards

- Non-leaf nodes should contain only
  - Module instantiations
  - Wires
  - Simple assigns for renaming: assign foo = bar (and not many of these)
- Leaf nodes are either stateful or not.
- Stateful leafs
  - Registers
  - Register files
  - Memory modules.
  - Need to have clk and reset.
  - May contain always @(posedge clk), always @(*), and '<=' assignments
- Non-stateful leafs
  - May contain always @(*), and '=' asignments
  - No clk or reset input.

Datapath Coding Standards

- Consistently use a good naming conventions
- Label all inputs and outputs
  - e.g. foo_in, foo_out
- Include module types in their names
  - A_mux -- the instantiated mux
- Give control lines descriptive and consistent names
  - A_mux_sel_in -- the input that controls the mux
  - A_mux_sel -- should not exist since it would be a control line (and would come from the control path)
  - The control unit would have a corresponding A_mux_sel_out

Build Useful Modules

- Parameterize!
- You should only ever write code for one
  - Register (of any width)
  - 2-input mux (of any width)
  - 3-input mux
  - etc.
- Give your modules descriptive names
  - my3Mux
  - my4Mux
  - myFF
  - gcd_control
  - gcd_datapath
  - gcd -- top-level.