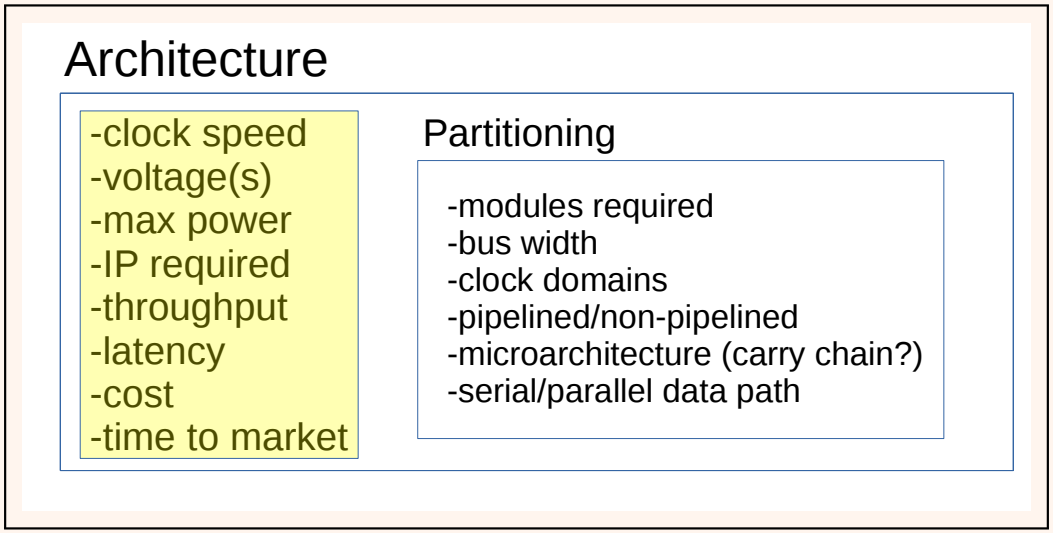
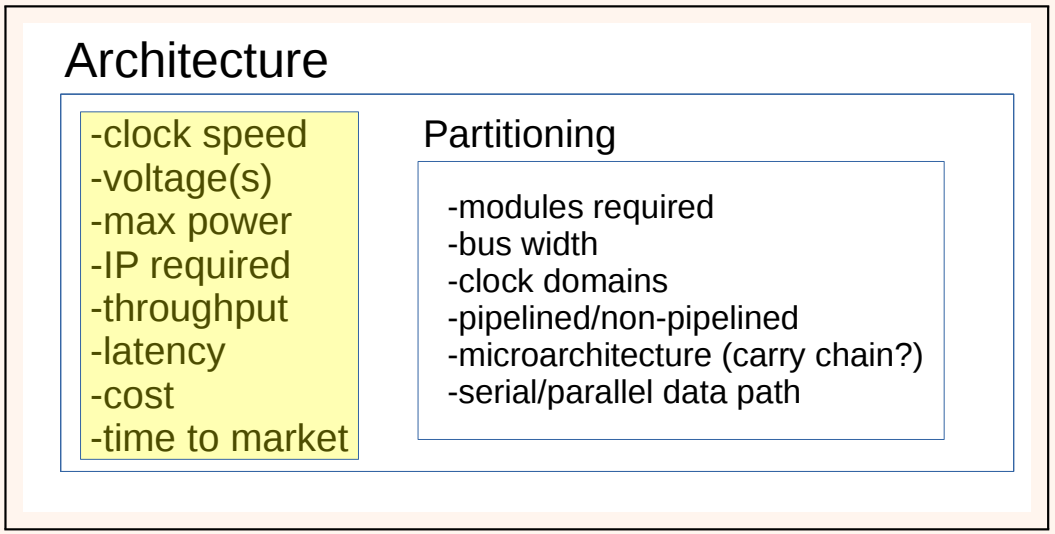
Architecture and Partitioning - Architecture

- Architecture
  - Management, Marketing and Senior Project Engineers work together to define product requirements. The product requirements will set cost, size and market window. These subsequently form the loose boundaries for the project's high-level architecture.
  - At the highest architectural level, a group of senior engineers derrive the requsite power consumption, cooling, size, and any IP required to meet schedule.

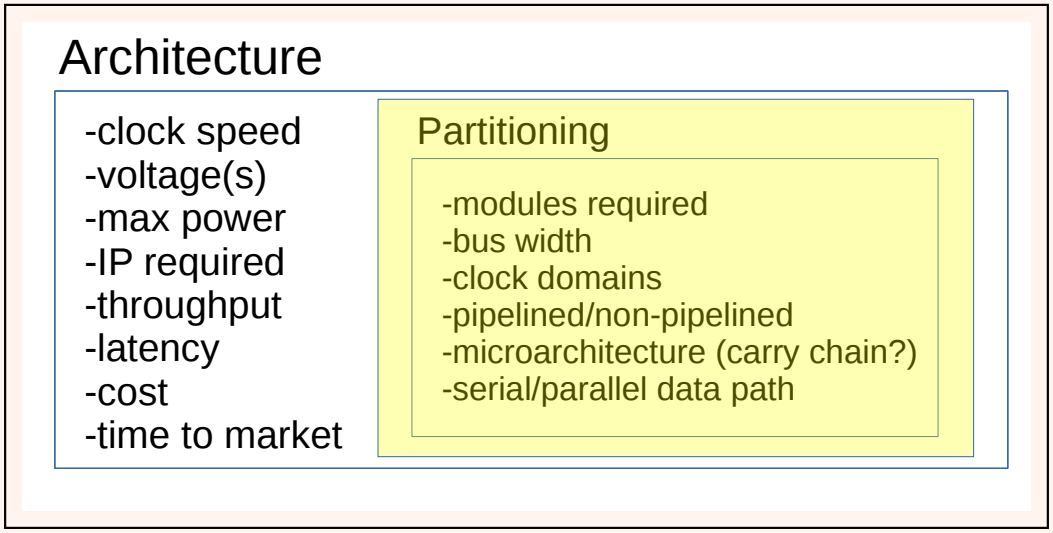


Architecture and Partitioning - Architecture

- Architecture
  - An experienced set of engineers will develop a good basic architecture that will inform much of the partitioning.
  - We won't focus much on these high-level architectural tasks. This job is for guys with lots of experience.



Architecture and Partitioning - Partitioning



- Partitioning is the process of determining how an architecture is physically structured and logically operates.
- Physical structuring defines a set of functional blocks, what cach does, and how they are interconnected. It is forged using block diagrams and lots of erasing.
- Logical operation provides the steps that must be taken to manipulate the data. It is made clear with state diagrams.
- The step-by-step sequence that the logical operations are performed is clarified by using a timing diagram.

Architecture and Partitioning - Partitioning

- Well-done partitioning is iterative and messy, filled with lots of questions, false starts, and possibilities explored
- Partitioning is about structure. What's in each box? How are the boxes connected?

Often, beginning SV users write code like its "C", simply describing a logical sequence of operations. But wait..., "variables" don't really exist; only flip-flops. Also, how do you make a "for loop" in hardware?. With SV, you must provide structure and temporal ordering. The synthesis tool cannot guess what you had in mind.

Architecture and Partitioning - Data and Control

- First cut partitions are made considering data and control paths, clock domains, power supply domains and the availability of IP blocks.
- We will focus mostly on partitioning by data and control structures.
- Later on we will look at partitioning via clock domains and how to cross between clock domains.

Architecture and Partitioning - Data Path

- Data path/functional block structuring usually comes first.
- We will initially consider the case of a synchronous, single clock domain design.
- Data Path Creation Procedure
  - Ask: From in to out, what operations need to be done to the data?
  - Ask: To perform the operations, what blocks are needed?
  - Don't think in terms of algorithms, we are not writing software.
  - Think: adders, shifters, muxes, registers, combinatorial logic, data busses
  - Then: How are the blocks connected to move the data appropriately?
  - Annotate data path elements with control signal stubs as needed.
  - Then, make a data-path-only timing diagram to show how the data moves between blocks over the busses with time.

Architecture and Partitioning - Data Path

- As you draw the functional blocks and data busses, name them with meaningful names.
- Use lower case letters and the underscore only
- Blocks have distinct block names and pin names at their inputs and outputs
- Later, your Verilog modules or "always blocks" will have exactly the same name, and have the same input, and output pin names.
- Busses have a name and should indicate the bus width for good bookeeping.
- The busses will have signal names in your Verilog as well.
- Control signal stubs should be given clear, unambigious names that indicate what their function is.

Architecture and Partitioning - Control Structures

- Now we consider what control signals we need to move the data through the structure we have created.
- We create these signals via state machines.
- We verify our thinking about state machine operation with timing diagrams.
- Ask: What sequence of steps are required to guide the data through the data path to create the function desired?
- This is initially a hard process. You will often move between state machines and timing diagrams. The timing diagram is used as a bookkeeping mechanism to keep temporaly track the control signals.
- Leverage the initial data path timing diagram to provide a starting place.

Architecture and Partitioning - Control Structures

- Draw state diagrams freehand on paper or whiteboard. You need to focus strictly on creation of the correct control sequence, not making a pretty picture with colors and nice arcs. Once its correct, then make it pretty.
- Likewise, do timing diagrams on paper or whiteboard. You will make too many mistakes to keep trying to do it over and over on a computer.
- Work back and forth between timing diagrams and state machine diagrams. After a while you will become more comfortable with the process and develop your own "flow".
- Its better to use several simple state machines than to make one big one. Also, simple state machines are far easier to validate.

Architecture and Partitioning - Clock Regions

- State machines with common clocks reside within the same module that has exactly one clock and one reset input.
- Data paths and their state machines both reside within a higher-level module that has exactly one clock and one reset.
- Whenever data or control passes between clock domains, great care must be taken to avoid metastability or mis-clocking.
- Clock domain crossing modules may have multiple clock and reset inputs
- Clock crossing modules need to be explictly coded and fully documented
- Each clock domain has its own reset signal that is synchronous to its particular clock.

Architecture and Partitioning - IP

- The availabilty of IP within FPGAs or available from ASIC vendors can strongly effect the design from the architectural level.
- IP may have its own clock, reset, or power requirements.
- IP will by necessity be within its own module and will probably be a black box to your design.
- FPGA megafunctions or IP can greatly simplify your design.

Architecture and Partitioning - Recap

- Partition into clearly defined functions
  - i.e., FIFO, adder, shift register, state machine
- Partition to minimize the number of interfaces between blocks
- **Never, never, never** mix control and data
  - i.e., embedding a state machine in an adder
- Keep to the highest level description you can wrap your brain around
- If you can't understand the operation of a module, break it up
- Only go to gate level when its really necessary to make intent clear
  - i.e., clock crossing, special delay elements
- HDL code usually dosen't expose the architecture. If you don't have an architecture, how do you know you got what you coded?
- Blocks created from partitioning usually become Verilog modules