FPGA vs. ASIC Design Styles

Certain elements of the ASIC and FPGA design philosophy are different because of the inherent differences in their underlying fabric.

Coding Style

ASIC: Designers tend to write *portable* Verilog/VHDL code, i.e., they don't make use of *named* cells.

FPGA: Designers can make heavy use of instantiated cells, either those provided in a library or those that they handcraft themselves.

Pipelining and Levels of Logic

ASIC: More control over delay in multiple level logic. Although pipelining popular here, more freedom exists w.r.t. performance optimization.

FPGA: Spreading out combinational logic in different LUTs hurts performance. Given each PLB contains LUT + reg, pipelining used more often.

Asynchronous Design Practices

ASIC: Designers may include structures that depend on the relative propagation delays of signals.

Asynchronous Design Practices

FPGA: This style does not work well because routing and associated delays vary widely for each run of the place-and-route tool.

ASIC: Make use of *combinational loops*, where outputs are fed back to the inputs. Can introduce race conditions, routing delays are carefully tracked.

FPGA: Lack of control over propagation delays makes this difficult in FPGAs. All feedback loops should include a register element.

ASIC: *Delay chains* (buffer chains) used to address race conditions, for example in asynchronous designs.

FPGA: Avoid them because propagation delay is difficult to predict.

Clock Considerations

ASIC: Can include a huge number of clock domains.

FPGA: Have a limited number of dedicated global clock resources.

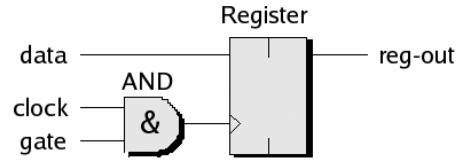
Clock Balancing

ASIC: Designers need to use special techniques to balance clock delays throughout the design.

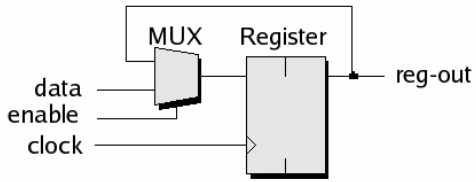
FPGA: Chips already include global, low-skew clock routing resources, FPGA vendor has already taken care of it.

Clock Gating vs. Clock Enabling

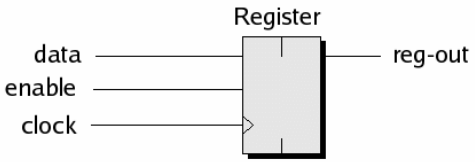
ASIC: Use *gated clocks* to help reduce power dissipation. Gives the design asynchronous characteristics, and care must be taken to avoid glitches.



(a) Clock gating



(b) Clock enabling ("then")



(b) Clock enabling ("now")

Clock Gating vs. Clock Enabling

FPGA: Designers use the technique of *enabling clocks*. Originally, a MUX was used, now FPGA architectures incorporate an *enable* on the register.

PLLs and Clock Conditioning Circuitry

FPGAs: One PLL for each dedicated clock. If used for on-chip clk generation, the design should provide a disable for testing and debugging.

Register and Latch Considerations

ASIC: Often make use of latches.

FPGA: Not recommended in FPGA designs.

ASIC: Libraries usually offer FFs with *both* a *set* and *reset* control.

FPGA: Can easily be configured with one or the other, otherwise the LUT is required.

ASIC: Typically don't allow *all* FFs to be initialized.

FPGA: Registers are programmed with default initial conditions, either 0 or 1, and provide a *global* reset signal.

Resource Sharing (Time-Division Multiplexing)

ASIC: Less effort is spent here on sharing resources.

FPGA: Limited resources forces designers to invest more effort here.

Sometimes it is not needed, in cases where the resources are provided whether you use them or not.

However, using them, e.g., multipliers, costs interconnect resources, so multiplexing them may cost more than instantiating multiple copies.

This is particularly true for simpler functions, e.g., adders, where the MUX may consume more resources than the adder itself.

State Machine Encoding

ASIC: A wider choice of state encoding options exist.

FPGA: Given that each PLB has a LUT and a register, a *one-hot* encoding scheme (one FF/state) is a natural mapping.

Test Methodologies

ASIC: Large effort spent on ensuring testability, e.g., SCAN, BIST, etc., to enable fabrication defects to be detected.

FPGA: Chip is tested by vendor so designer does not need to worry about this for FPGAs.