ECE 551
System on Chip Design

RTL Verification, Synthesis and Optimization

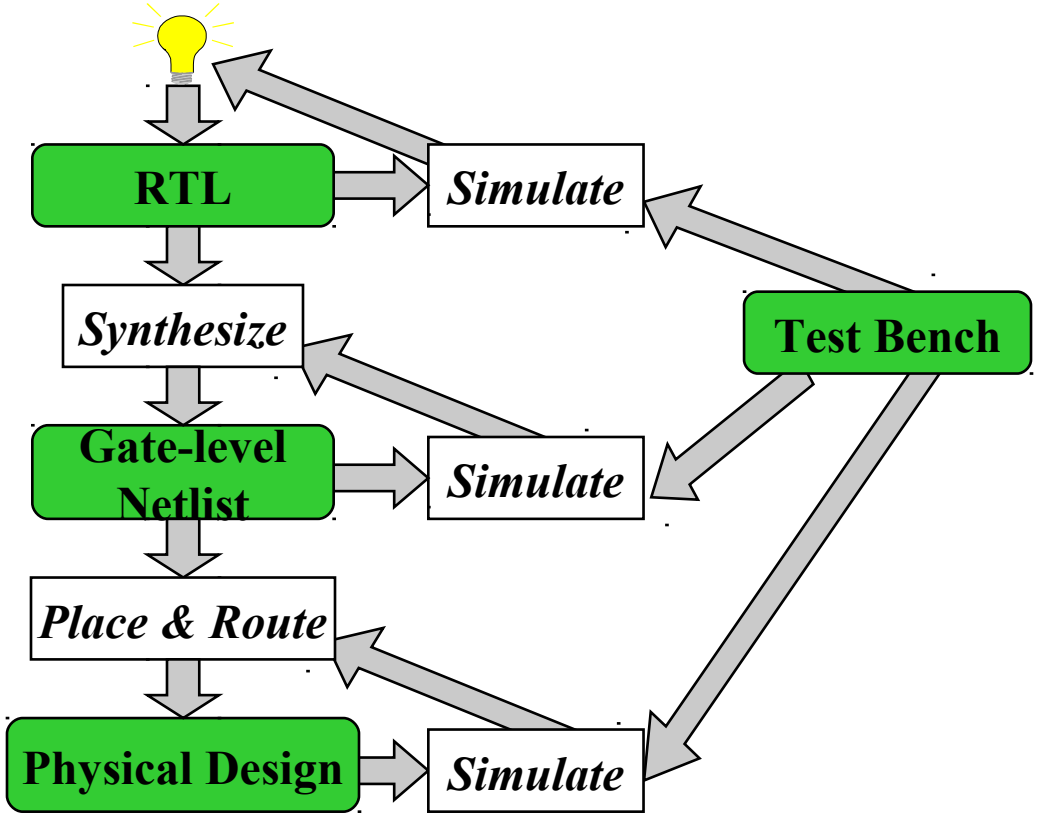
Garrett S. Rose
Fall 2018

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

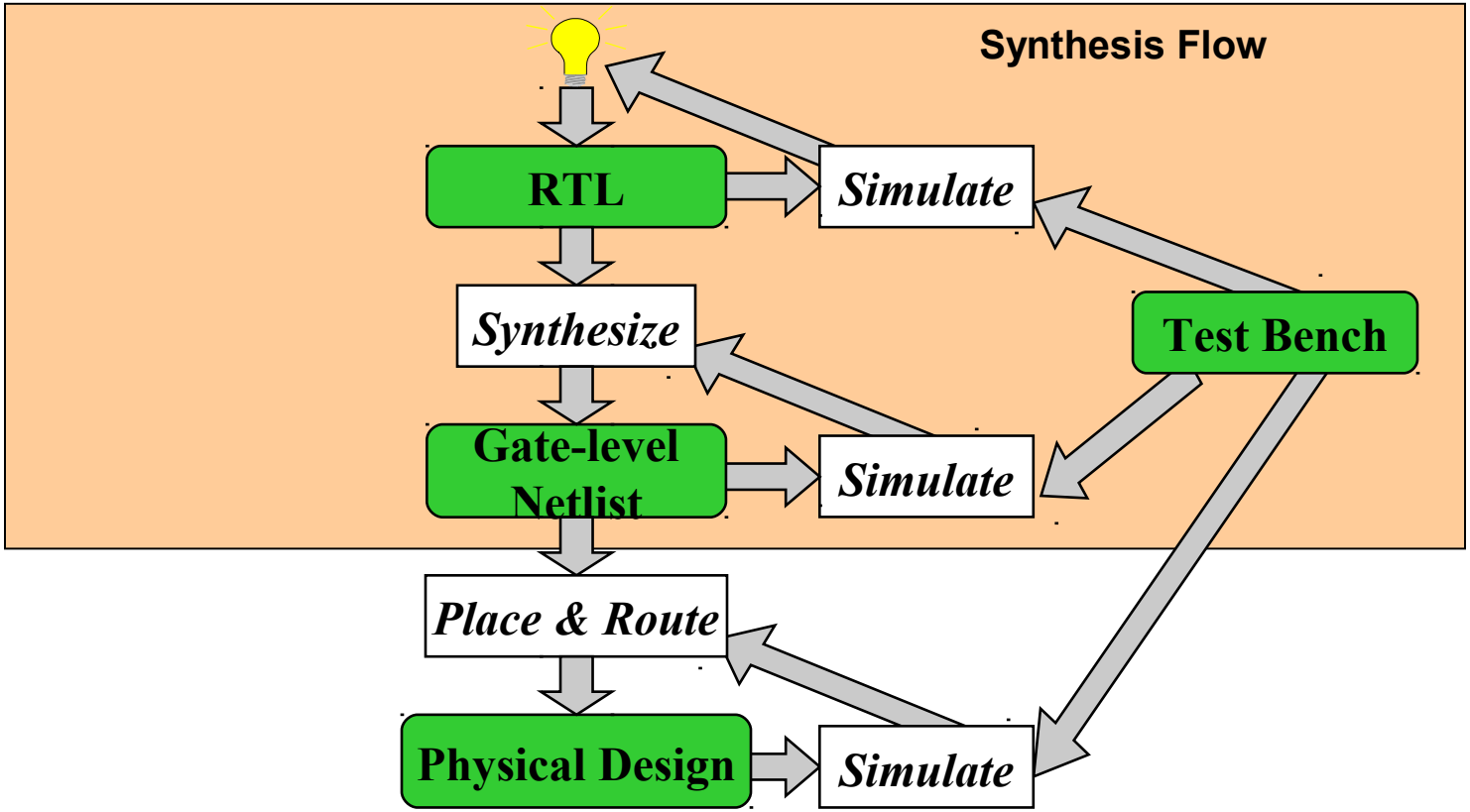
Outline

- Introduction to RTL (Verilog) verification using ModelSim
- Verification of gate level netlist using ModelSim
- Synthesizing RTL design code into a gate level netlist

ASIC/SoC Design Flow



ASIC/SoC Design Flow



RTL Synthesis Design Steps

- Code design in HDL such as VHDL or Verilog
  - Can use ‘gedit’ on Linux servers
- Simulation/verification of HDL description
  - ModelSim (Mentor Graphics) or NCLaunch (Cadence)
  - Use test bench, Verilog or VHDL
- Synthesis of HDL description
  - Use RTL Compiler
  - Output of synthesis is a Verilog gate level netlist
  - Netlist built from standard cells
- Gate level netlist should be simulated using same test bench designed for RTL verification
  - ModelSim or NCLaunch useful here as well

Coding Hardware

- Previous weeks: combinational and sequential logic with SystemVerilog
- Verilog used to describe hardware components in code
- Use structural Verilog (or VHDL) as much as possible

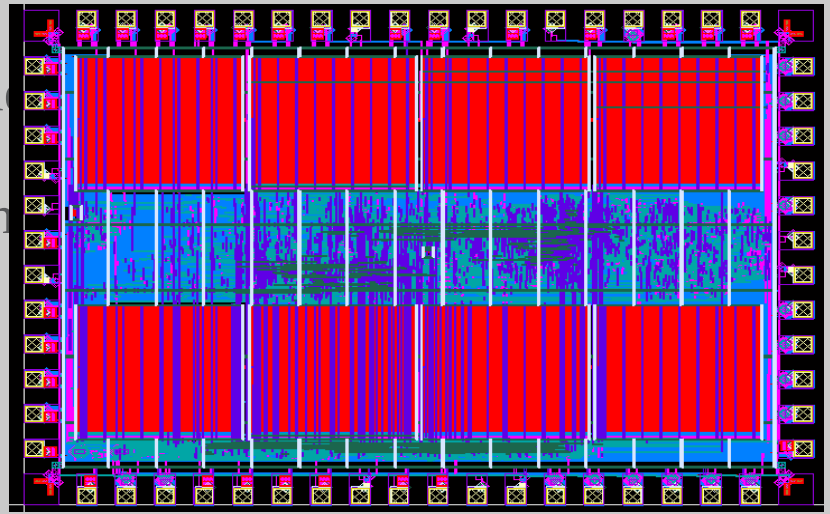
Coding Hardware

- Last week: overview of VHDL for modeling, simulation and designing large

```
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
--------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SRAD_Top is
    Port ( clk      : in std_logic;
           ExtRST   : in std_logic;
           NewFrm   : in std_logic;
           DataIn   : in std_logic_vector(7 downto 0);
           Addr_R   : out std_logic_vector (13 downto 0);
           Dsp_Addr                      : out std_logic_vector (13
downto 0);
           DataOut  : out std_logic_vector(7 downto 0);
           InAdMux  : out std_logic;
           DisAdMux : out std_logic;
           Dsp_WE   : out std_logic;
           VGA_Ena  : out std_logic;
           SRAD_Clk : out std_logic
                    );
end SRAD_Top;

architecture Structure of SRAD_Top is

COMPONENT Antilog
    Port ( D        : in std_logic_vector(7 downto 0);
```

re compon

ure as much



Register Example

```
// Simple SystemVerilog 4-bit register

module reg4 (input d0, d1, d2, d3, en, clk,
             output q0, q1, q2, q3);
  logic q0_tmp, q1_tmp, q2_tmp, q3_tmp;

  always_ff @(posedge clk) begin
    if (en) begin
      q0_tmp <= d0;
      q1_tmp <= d1;
      q2_tmp <= d2;
      q3_tmp <= d3;
    end
  end

  assign #2 q0 = q0_tmp;
  assign #2 q1 = q1_tmp;
  assign #2 q2 = q2_tmp;
  assign #2 q3 = q3_tmp;
endmodule
```

Register Example

```
// Simple SystemVerilog 4-bit register

module reg4 (input d0, d1, d2, d3, en, clk,
             output q0, q1, q2, q3);
  logic q0_tmp, q1_tmp, q2_tmp, q3_tmp;

  always_ff @(posedge clk) begin
    if (en) begin
      q0_tmp <= d0;
      q1_tmp <= d1;
      q2_tmp <= d2;
      q3_tmp <= d3;
    end
  end

  assign #2 q0 = q0_tmp;
  assign #2 q1 = q1_tmp;
  assign #2 q2 = q2_tmp;
  assign #2 q3 = q3_tmp;
endmodule
```

Will this synthesize?

Register Example

```
// Simple SystemVerilog 4-bit register

module reg4 (input d0, d1, d2, d3, en, clk,
             output q0, q1, q2, q3);
  logic q0_tmp, q1_tmp, q2_tmp, q3_tmp;

  always_ff @(posedge clk) begin
    if (en) begin
      q0_tmp <= d0;
      q1_tmp <= d1;
      q2_tmp <= d2;
      q3_tmp <= d3;
    end
  end

  assign #2 q0 = q0_tmp;
  assign #2 q1 = q1_tmp;
  assign #2 q2 = q2_tmp;
  assign #2 q3 = q3_tmp;
endmodule
```

Will this synthesize?

Yes, will ignore '#2'

Simulation & ModelSim

- ModelSim used to simulate and verify your HDL code
- To run ModelSim, type following at the command prompt:

```
vsim &
```

- Will need to first setup environment

```
. /sw/etc/mentor/modelsim-se.sh
```

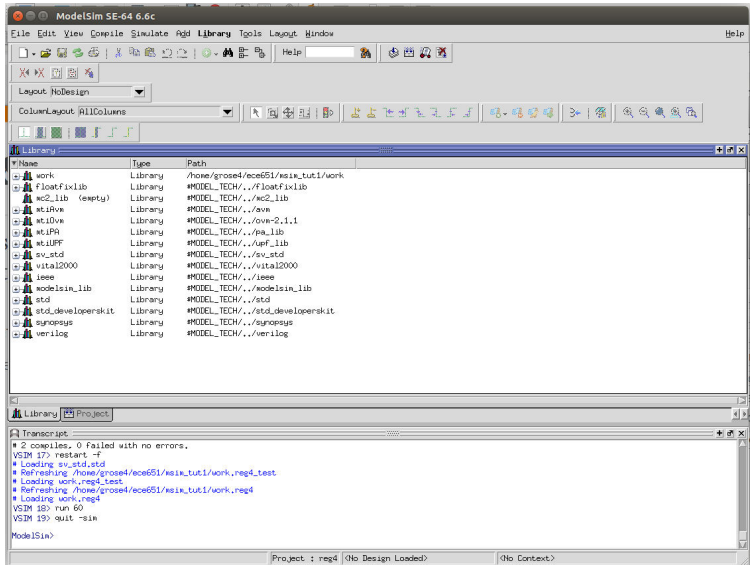
Simulation & ModelSim

- ModelSim used to simulate and verify your HDL code
- To run ModelSim, type following at the command prompt:

```
vsim &
```

- Will need to first setup environment

```
. /sw/etc/mentor/modelsim-se.sh
```



Simulation Test Bench

```
// Simple SystemVerilog 4-bit register Testbench

module reg4_test;
  bit di0, di1, di2, di3;
  bit ten, tclk;
  bit qo0, qo1, qo2, qo3;

  reg4 my_reg(.d0(di0), .d1(di1), .d2(di2), .d3(di3), .en(ten), .clk(tclk),
              .q0(qo0), .q1(qo1), .q2(qo2), .q3(qo3));

  initial begin
    ten <= 0; tclk <= 0;
    di0 <= 0; di1 <= 0; di2 <= 0; di3 <= 0; #5;
    ten <= 1; tclk <= 1; #5;
  end

  always begin
    tclk <= 0; #5; tclk <= 1; #5;
    di0  <= 0; di1  <= 1; di2  <= 1; di3  <= 0;
    tclk <= 0; #5; tclk <= 1; #5;
    di0  <= 1; di1  <= 0; di2  <= 1; di3  <= 1;
    tclk <= 0; #5; tclk <= 1; #5;
    di0  <= 1; di1  <= 0; di2  <= 0; di3  <= 0;
    tclk <= 0; #5; tclk <= 1; #5;
    di0  <= 0; di1  <= 1; di2  <= 0; di3  <= 1;
    tclk <= 0; #5; tclk <= 1; #5;
  end
endmodule
```

Format can be picky – read warnings

QuestaSim

- For the register example, you can copy the `reg4` SystemVerilog code and testbench code into two separate files
- Create a simulation directory and place both SystemVerilog and testbench files in that directory (example: ~/HDLSim)
- You can then follow steps outlined in QuestaSim tutorial to simulate your SystemVerilog code
- Similar approach can be taken for simulating Verilog netlist that results from synthesis in Design Compiler

Synopsys Design Synthesis

- Synthesis of HDL description of your design is done using Design Compiler and running the command **`'dc_shell -gui'`**
- Upon executing the **`'dc_shell -gui'`** command from the Linux command prompt you will get a new prompt for Design Compiler
- NOTE: do NOT run **`'dc_shell -gui'`** in the background with &
- Synthesis is completed by issuing several commands at the Design Compiler command prompt

RTL Synthsis Commands

- Set the location of the standard cell library so Design Compiler can find the standard cell definitions:

```
set search_path "search_path . /sw/cadence/FreePDK45-1.3/osu_soc/lib/files"
```

- For this class, we can use the FreePDK45 standard cell library developed by Okahoma State Univerisity:

```
set alib_library_analysis_path "/sw/cadence/FreePDK45-1.3/osu_soc/lib/files"
set link_library   [set target_library [concat  [list gscl45nm.db] [list dw_foundation.sldb]]]
set target_library "gscl45nm.db"
```

- The **`'set'`** command is also useful for defining several Design Compiler objects from library definitions to optimization parameters

RTL Synthesis Commands

- Before synthsizing, you must load your HDL file:

```
analyze -library WORK -format -sverilog {/home/grose4/ece651/msim_tut1/reg4.sv}
```

- The **`-sverilog`** option tells Design Compiler that the file being read in is SystemVerilog (default is Verilog)
- Read lower level HDL files first with the file containing the top module read in last
- The HDL source code is elaborated by entering:

```
elaborate reg4
link
```

RTL Synthesis Commands

- With everything loaded and elaborated, you can synthesize your HDL file by issuing the following command:

```
compile_ultra -gate_clock -no_autoungroup
```

- The mapped/synthesized gate level netlist needs to be saved to a file that can be simulated and imported into the place & route tools at later design stage:

```
write -f verilog -h -o reg4_glnet.v
```

Tcl Files & Synthesis

- A Tcl file in Design Compiler is simply a list of the commands that are to issued at the Design Compiler prompt to complete each synthesis step

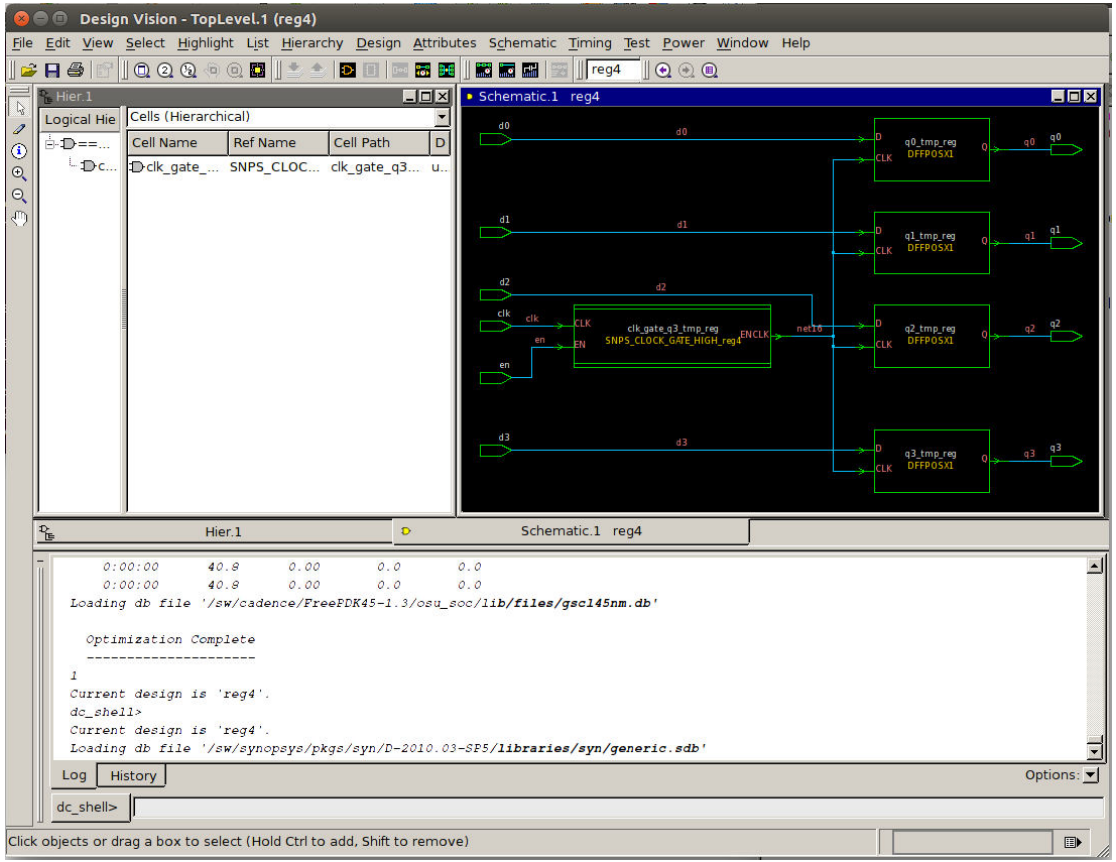
- You can create your own Tcl file by listing the commands already mentioned and saving the file as <mytclfile>.tcl

- To load and run a Tcl file from Design Compiler, in the GUI select **File -> Execute Script...**

Example Tcl File

```
set search_path    "search_path . /sw/cadence/FreePDK45-1.3/osu_soc/lib/files"
set link_library   [set target_library [concat  [list gscl45nm.db] [list
dw_foundation.sldb]]]
set target_library "gscl45nm.db"

set mw_logic1_net  "VDD"
set mw_logic0_net  "GND"
analyze -library WORK -format sverilog {/home/grose4/ece651/msim_tut1/reg4.sv}
elaborate reg4
link
check_design

create_clock clk -name ideal_clock1 -period 10
define_design_lib WORK -path "./work"
compile_ultra -gate_clock -no_autoungroup
write -f verilog -h -o reg4_glnet.v
write_sdf reg4.sdf
write_sdc reg4.sdc
```

Synthesis of reg4



THE UNIVERSITY OF TENNESSEE KNOXVILLE

Synthesis of reg4 Verilog Netlist Output

```
module SNPS_CLOCK_GATE_HIGH_reg4 ( CLK, EN, ENCLK );
  input CLK, EN;
  output ENCLK;
  wire   net4, net6, net7, net10, n1;
  assign net4 = CLK;
  assign ENCLK = net6;
  assign net7 = EN;

  LATCH latch ( .CLK(n1), .D(net7), .Q(net10) );
  AND2X1 main_gate ( .A(net10), .B(net4), .Y(net6) );
  INVX1 U2 ( .A(net4), .Y(n1) );
endmodule


module reg4 ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );
  input d0, d1, d2, d3, en, clk;
  output q0, q1, q2, q3;
  wire   net16;

  SNPS_CLOCK_GATE_HIGH_reg4 clk_gate_q3_tmp_reg ( .CLK(clk), .EN(en), .ENCLK(
        net16) );
  DFFPOSX1 q3_tmp_reg ( .D(d3), .CLK(net16), .Q(q3) );
  DFFPOSX1 q0_tmp_reg ( .D(d0), .CLK(net16), .Q(q0) );
  DFFPOSX1 q1_tmp_reg ( .D(d1), .CLK(net16), .Q(q1) );
  DFFPOSX1 q2_tmp_reg ( .D(d2), .CLK(net16), .Q(q2) );
endmodule
```

Optimization Options for Synthesis

- Another look at RTL to physical layout (GDSII*) design flows
- Metrics used for optimization
  - Speed
  - Area
  - Power
  - Others?
- ASIC/SoC design choices that allow optimizations

*GDSII = Graphic Data System II (physical layout format used by Cadence)

Timing Driven Synthesis

- Traditionally, timing has been one of the major drivers in ASIC/SoC design
- The goal is to minimize the delay on the *critical path* of the design so that the frequency is maximized
- CAD tools must include accurate and robust models for estimating the delay through the circuit(s)
- During synthesis, several choices can be made to reduce delay:
  - Optimize number of logic levels
  - Logic family used (if in standard cell library)
  - Cell sizing (usually have multiple sized std. cells)

Power as a Design Metric

- Power determined by four major factors:
  - Capacitance being driven (C)
  - Voltage (V_{DD})
  - Frequency (f)
  - Activity factor (α)

$$P = \alpha \cdot C \cdot V_{DD}^2 \cdot f$$

- Low-power design techniques focus on these factors for controlling the power consumption of a design

Low-Power Design Techniques

- Dynamic Voltage Scaling (DVS)
  -- lower V_{DD} during runtime for quadratic savings

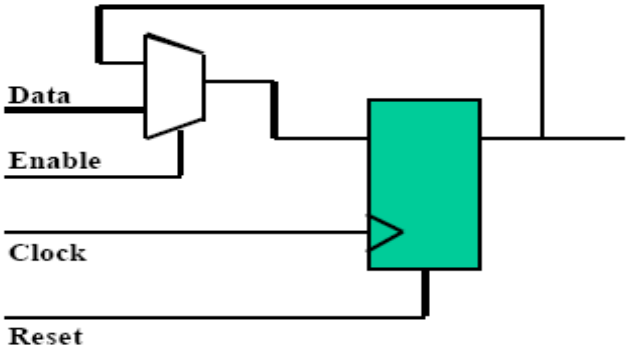
- Frequency scaling

- Sleep mode transistors

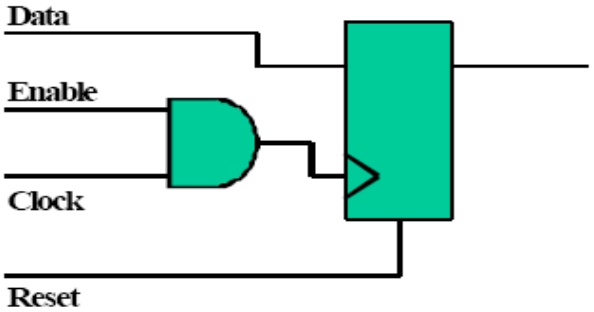
- Note: These are all techniques for reducing dynamic power; as technology scales, static power is becoming more of a concern

Clock Gating

- Dynamic power control through synthesis typically due to clock gating
- Usually this means shutting off the clock to flip flop(s)
- Example to the right:
  - Conceptually the same
  - Implementation 1 clocks the flip flop every cycle
  - Implementation 2 only clocks when enabled
    -- *the lower power design*



Implementation 1



Implementation 2

Synthesized Clock Gating

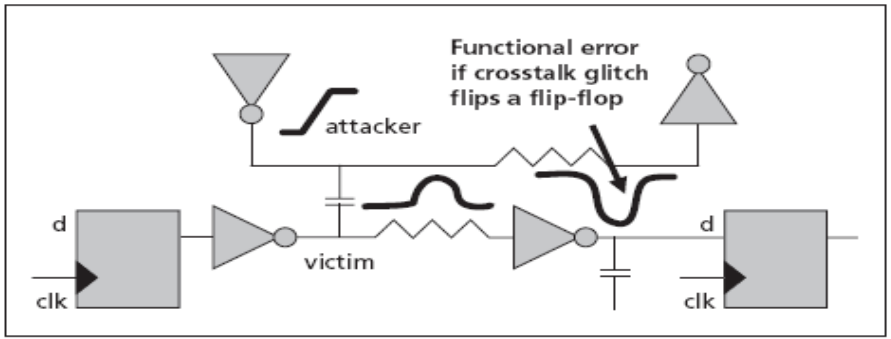
- To implement clock gating for power control during synthesis, tools analyze design at elaboration stage
- Most likely, gating structures are not applied to every register – cost in power of gating would exceed savings on the flip flops
- Synthesis tool tries to find gating enable signals within the design that can control the clock for a register bank

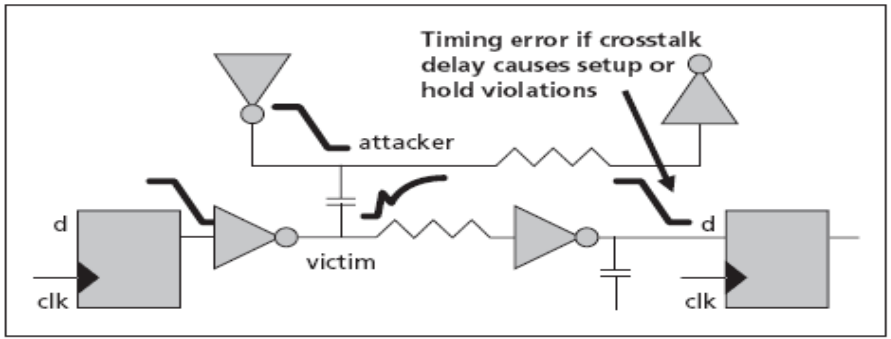
Signal Integrity

- A good design flow also must take into account signal integrity
- Interconnect plays a dominant role in silicon performance in nanometer designs -- coupling capacitance beginning to dominate
- Most signal integrity optimizations take place during routing, but things can be done up front:
  - Give critical global signals special treatment (stricter restrictions on signal skews)
  - Carefully select of intellectual property (IP) blocks

SI Closure Criteria

- Traditionally, signal integrity effects were analyzed and repaired manually or just ignored
  -- this approach no longer works
- Signal integrity failures due to: reduced feature size, smaller interconnect pitch, & lower V_{DD}





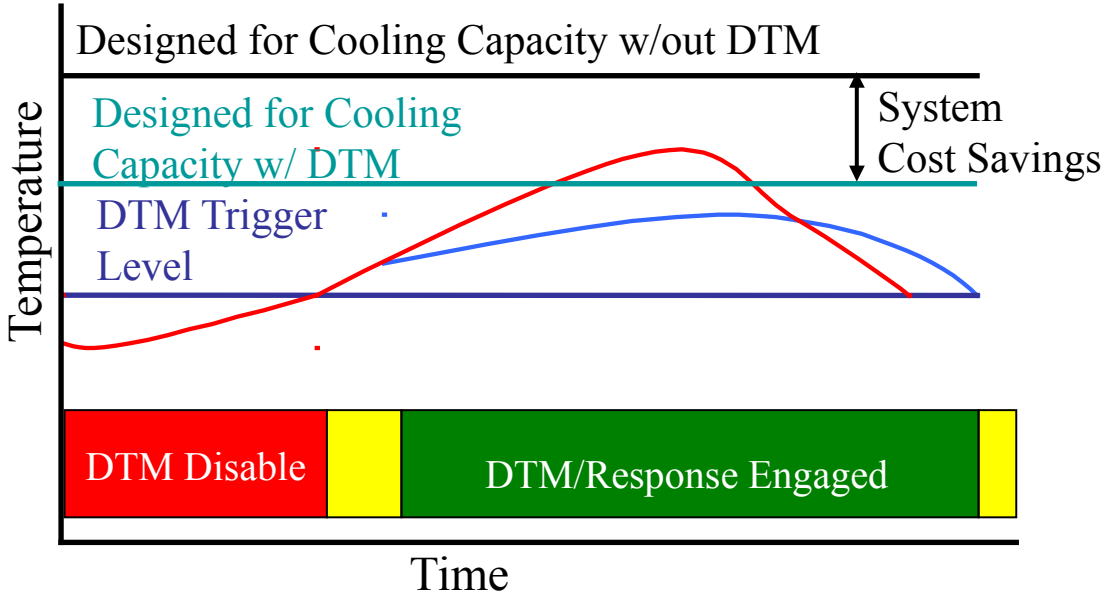
Source: Managing Signal Integrity in Nanometer Digital Designs, Cadence Technical Paper, 2004.

Thermal-Aware Design

- Temperature is more of a concern as technology continues to scale well below 100nm
- As temperature is related to power density, low-power techniques can be reduce temperature
  - DVS = Dynamic Voltage Scaling
  - Frequency Scaling
  - Use of sleep mode
- A thermal-aware design *responds* to temperature:
  - Actively monitoring “hot spots” with sensors
  - Monitoring the activity factor

DTM: Dynamic Thermal Management

- Dynamic Thermal Management (DTM) can reduce packaging cost and improve portability



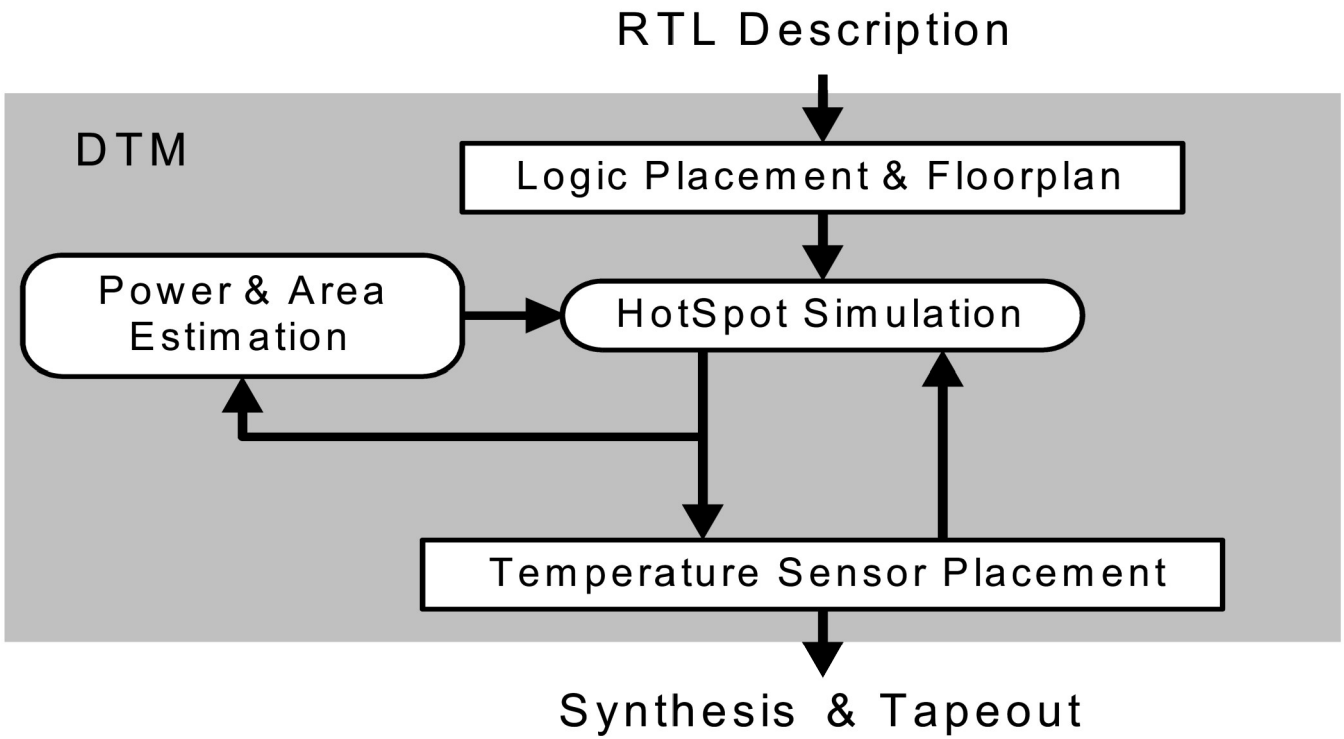
Source of Fig.: D. Brooks et al., "Dyn. thermal management for high-perf. microproc.," HPCA-7, 2001

Thermal Driven Floorplan

- In the ASIC/SoC design flow, floorplanning can be leveraged to minimize potential hot spots
- Logic blocks (or sections of blocks) deemed to be hot are not placed near one another
- Thermally driven floorplanning must utilize robust model for determining hot spots
- HotSpot (developed at UVA) is one tool for modeling on-chip temperature

Ref.: K. Skadron et al., “HotSpot: Techniques for Modeling Thermal Effects,” THERMINIC, 2002

DTM in the Design Flow

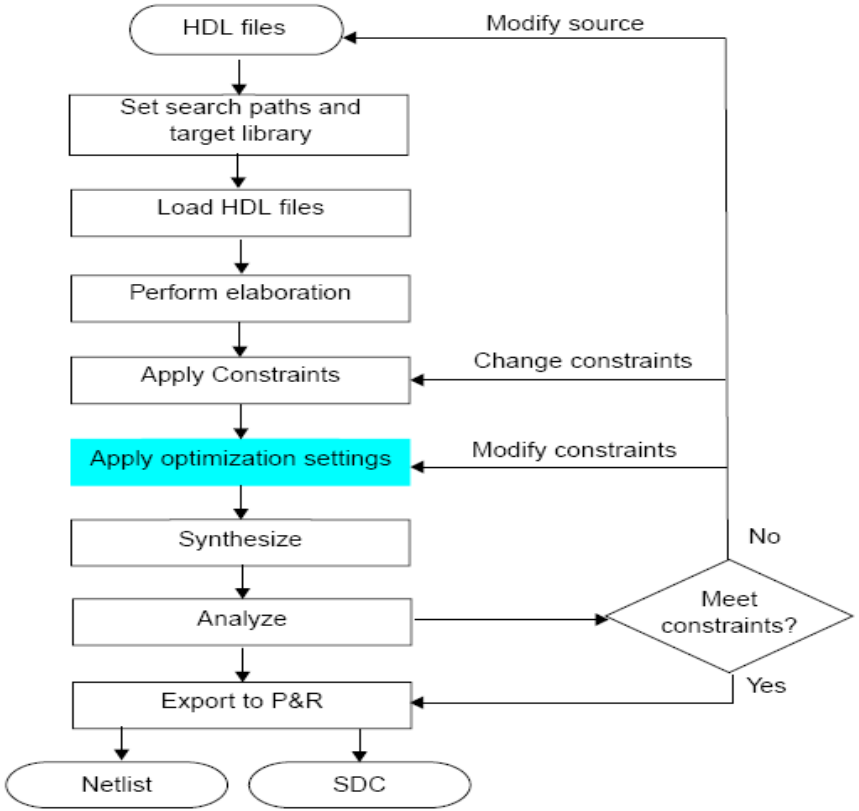


Levers for Design Optimization Choices

- So far we've discussed the issues and even some high level solutions
- Question now: what do we have control over to optimize a design?
- Some things can be done during synthesis, others must occur at later stages in the design flow
- We'll focus on synthesis today

Synthesis Design Flow

- Design flow for Design Compiler
- This particular flow shows each step on your (or your .tcl) perform
- Note the highlighted step: Apply optimization settings



Design Compiler: preserve

- Design Compiler will perform optimizations that can result in logic changes by default
- If you *do not* want some instances in your RTL description to change, you can use **`'set_dont_touch'`**:

```
dc_shell> set_dont_touch object
```

- *object* may be a hierarchical instance name, a primitive, or a module or submodule name

Design Compiler: Boundary Optimization

- Design Compiler performs boundary optimization for all hierarchical instances. Examples:
  - Constant propagation across hierarchies
  - Rewiring equivalent signals across hierarchy
- Essentially, boundary optimizations will be across module boundaries
- Boundary optimization can be controlled using:

```
dc_shell> compile -boundary_optimization

dc_shell> set_boundary_optimization subdesign
```

Worst Negative Slack

- Design Compiler (and many other tools) uses *Worst Negative Slack* (WNS) to achieve timing requirements
- Slack = Design Delay – Predicted Delay
  - *Design Frequency* is essentially the target which is usually higher than *Market Frequency*
  - *Predicted Frequency* is the frequency of the current design determined by low-level simulation
- Negative slack occurs when the design does not meet the timing requirements
- Worst Negative Slack refers to the critical path, the path with the most delay

Retiming the Design

- *Retiming*: technique for improving performance of sequential circuits by repositioning registers
  -- reduces cycle time or area with no I/O latency change

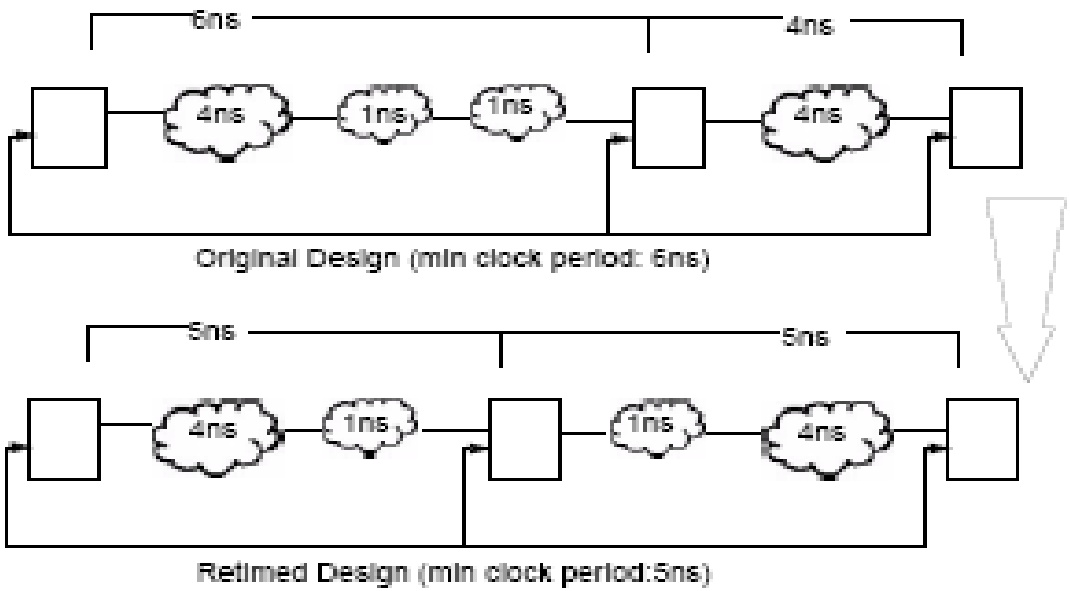
- Pipelining is a subset of retiming

- Retiming redistributes sequential elements at appropriate locations to meet requiremensts

- Retiming does not change combinational logic

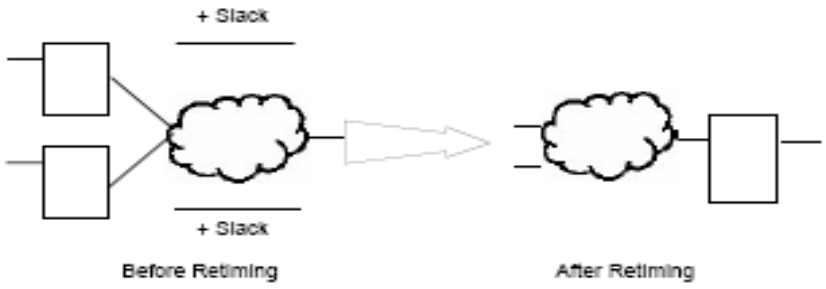
Retiming for Timing

- Improving clock period or timing slack common
- Design Compiler distributes the registers within the design to provide minimum cycle time



Retiming for Area

- When retiming for area, Design Compiler moves registers to minimize register count without worsening the critical path in the design



More on Retiming

- Typically, Design Compiler retimes blocks marked with option ‘retime’

```
dc_shell> compile_ultra -retime
```

- Design for Test (DFT) and low-power features can also be incorporated into retiming techniques

Summary

- Synthesis can optimize for performance, power, area, and even signal integrity
- Clock gating is a useful tool during synthesis for reducing dynamic power
- Retiming can be a powerful tool for minimizing both area and delay
- Design Compiler provides many options for such optimizations
  --suggest reading more about them