Lecture 21:
Synthesis & Timing Analysis

Mark McDermott
Electrical and Computer Engineering
The University of Texas at Austin

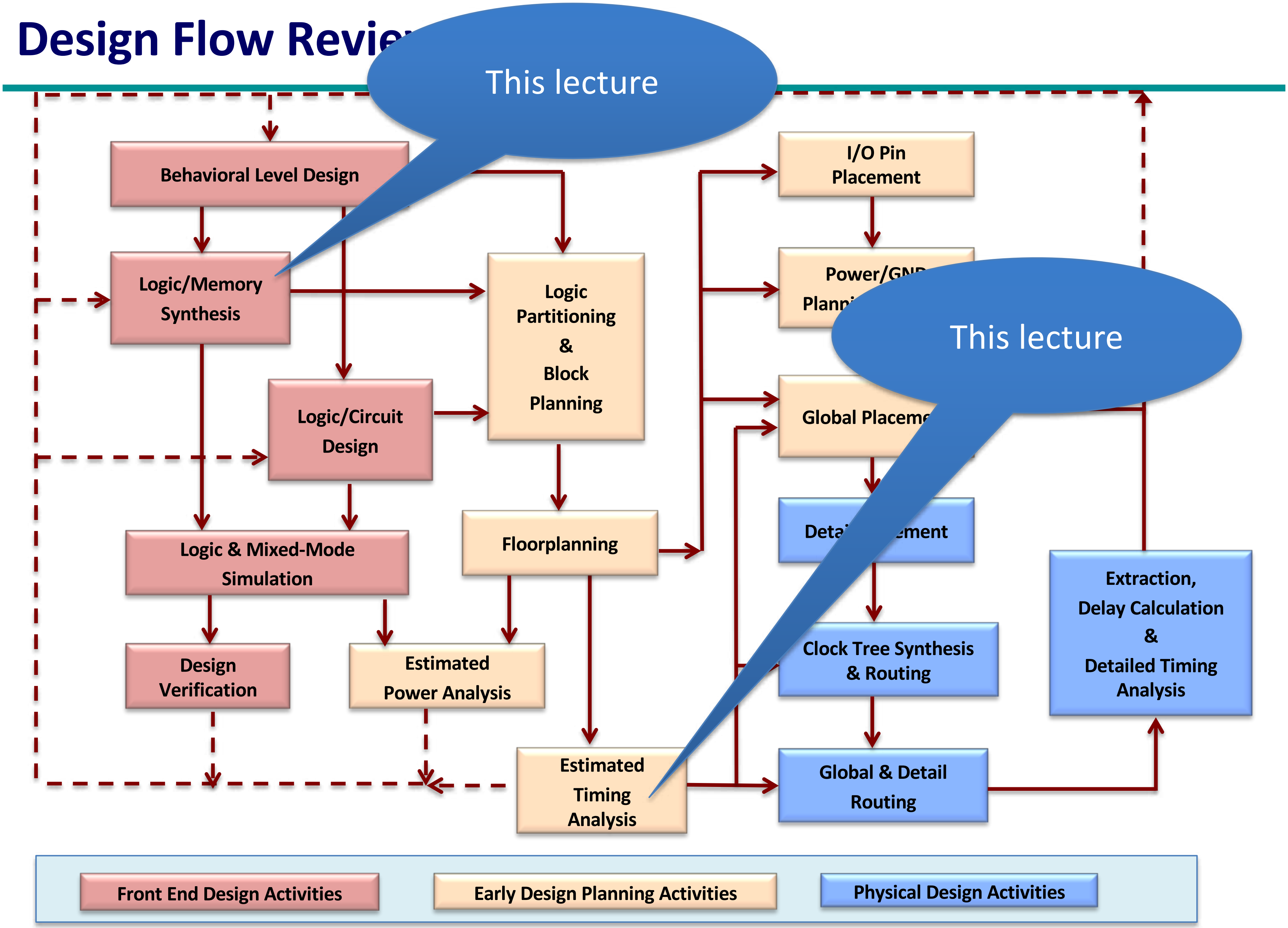
Agenda

- Overview of Logic Synthesis
- Overview of Static Timing Analysis

TLAs

- EDP - Early Design Planning
- EDP-TC - Timing Closure for EDP
- TA - Timing Analysis
- STA - Static Timing Analysis
- SSTA – Statistical Static Timing Analysis
- .lib – File containing cell timing and power information for STA
- DCL - Delay Calculator Language
- AT - Arrival Time
- RAT - Required Arrival Time
- PT – Pass-through
- LCB - Local Clock Buffer
- CL – Combinational Logic
- FF – Flip Flop

Design Flow Revie

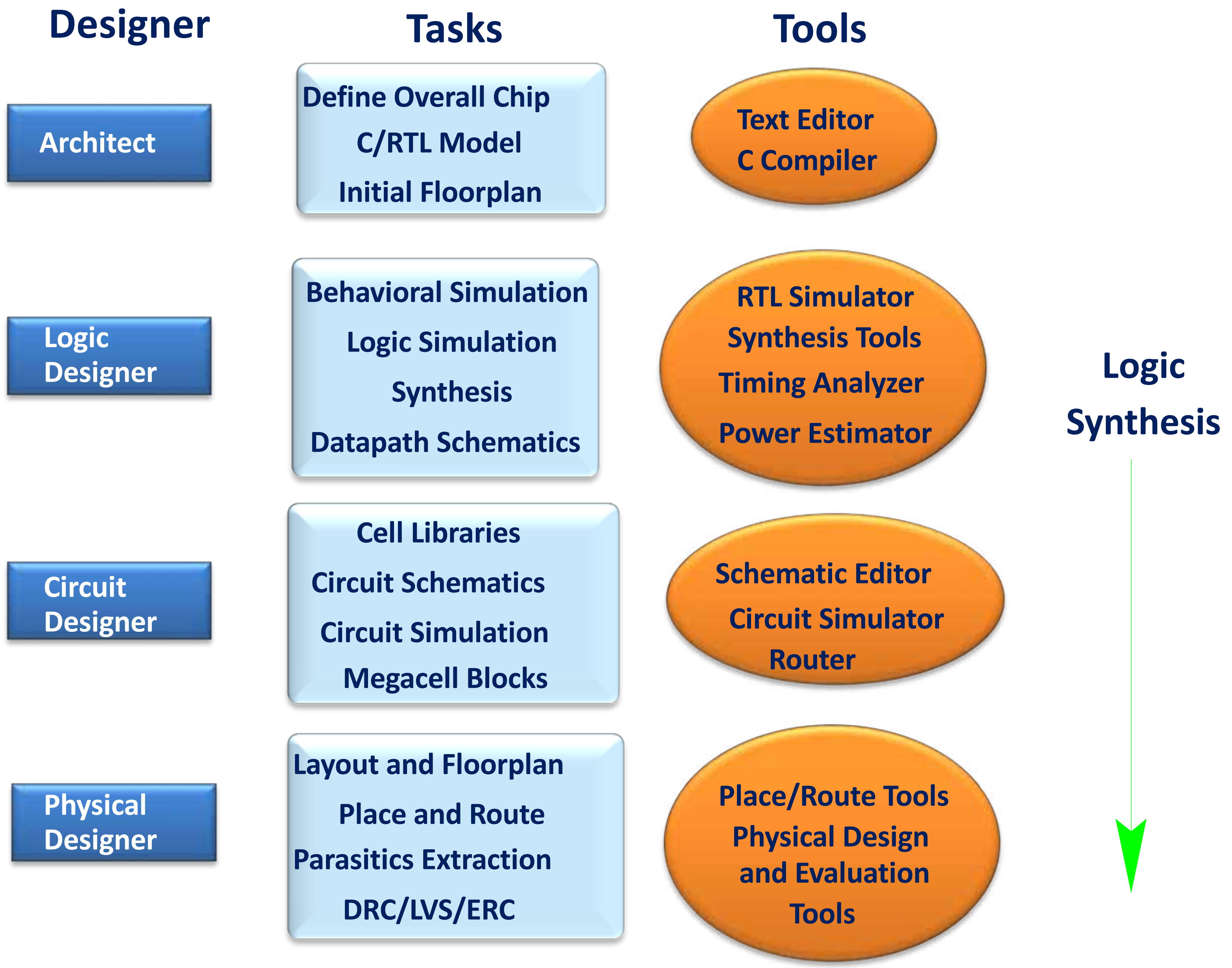

This lecture
This lecture
Behavioral Level Design
Logic/Memory Synthesis
Logic/Circuit Design
Logic & Mixed-Mode Simulation
Design Verification
Logic Partitioning & Block Planning
Floorplanning
Estimated Power Analysis
Estimated Timing Analysis
I/O Pin Placement
Power/GND Planni
Global Placeme
Detai ement
Clock Tree Synthesis & Routing
Global & Detail Routing
Extraction, Delay Calculation & Detailed Timing Analysis
Front End Design Activities
Early Design Planning Activities
Physical Design Activities

Logic Synthesis

Brief History of Logic Synthesis

- 1960s: first work on automatic test pattern generation used for Boolean reasoning
  - D-Algorithm
- 1978: Formal Equivalence checking introduced at IBM in production for designing mainframe computers
  - SAS tool based on the DBA algorithm
- 1979: IBM introduced logic synthesis for gate array based main frame designed
  - LSS, next generation was BooleDozer
- End 1986: Synopsys founded
  - first product “remapper” between standard cell libraries
  - later extended to full blown RTL synthesis
- 1990s other synthesis companies enter the marker
  - Ambit, Compass, Synplicity. Magma, Monterey, ...

Synthesis in the Design Flow

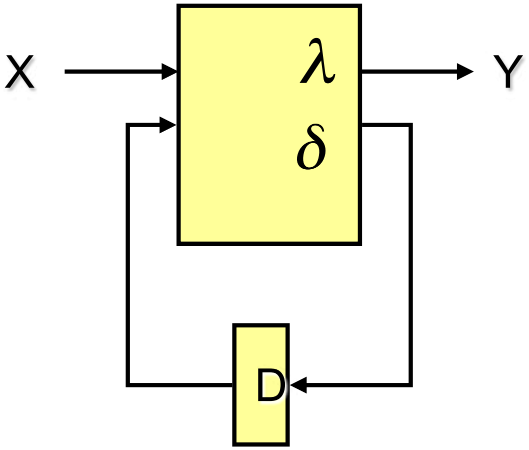


Designer	Tasks	Tools
Architect	Define Overall Chip C/RTL Model Initial Floorplan	Text Editor C Compiler
Logic Designer	Behavioral Simulation Logic Simulation Synthesis Datapath Schematics	RTL Simulator Synthesis Tools Timing Analyzer Power Estimator
Circuit Designer	Cell Libraries Circuit Schematics Circuit Simulation Megacell Blocks	Schematic Editor Circuit Simulator Router
Physical Designer	Layout and Floorplan Place and Route Parasitics Extraction DRC/LVS/ERC	Place/Route Tools Physical Design and Evaluation Tools

What is Logic Synthesis?

- Design described in a Hardware Description Language (HDL)
  - Verilog, VHDL
- Simulation to check for correct functionality
  - Simulation semantics of language
- Synthesis tool
  - Identifies logic and state elements
  - Technology-independent optimizations (state assignment, logic minimization)
  - Map logic elements to target technology (standard cell library)
  - Technology-dependent optimizations (multi-level optimization, gate strengths, etc.)

What is Logic Synthesis? (cont)



Given: Finite-State Machine F(X,Y,Z, λ, δ) where:

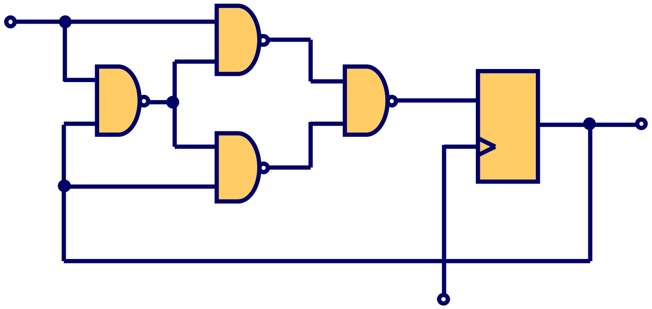
X: Input alphabet

Y: Output alphabet

Z: Set of internal states

λ : X x Z → Z (next state function)

δ : X x Z → Y (output function)



Target: Circuit C(G, W) where:

G: set of circuit components g∈{Boolean gates,
flip-flops, etc}

W: set of wires connecting G

Objective Function for Synthesis

- **Minimize area**
  - in terms of literal count, cell count, register count, etc.
- **Minimize power**
  - in terms of switching activity in individual gates, deactivated circuit blocks, etc.
- **Maximize performance**
  - in terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems
- **Any combination of the above**
  - combined with different weights
  - formulated as a constraint problem
    - "minimize area for a clock speed > 300MHz"
- **More global objectives**
  - feedback from layout
    - actual physical sizes, delays, placement and routing

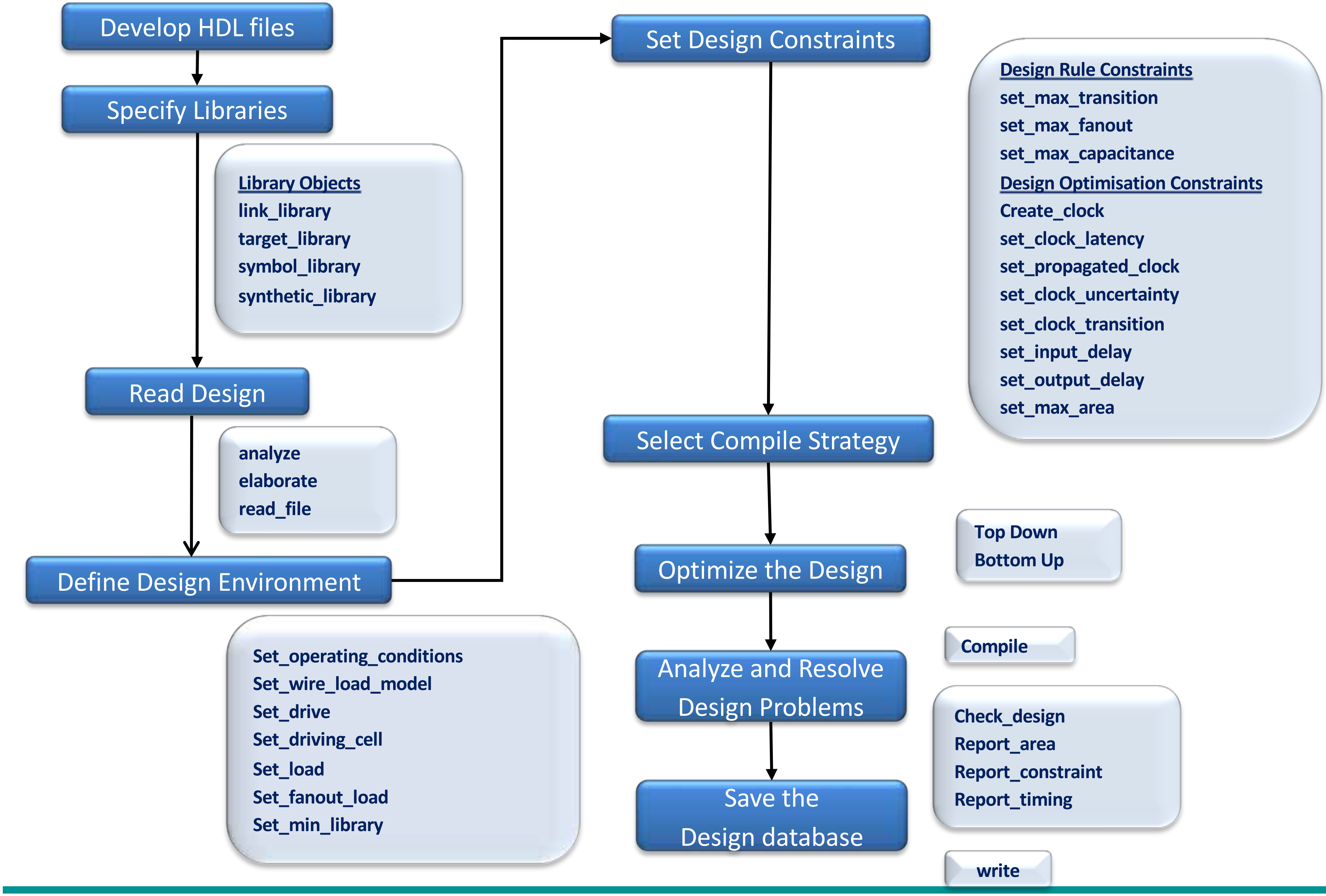
Constraints on Synthesis

- **Given implementation style:**
  - two-level implementation (PLA, CAMs)
  - multi-level logic
  - FPGAs

- **Given performance requirements**
  - minimal clock speed requirement
  - minimal latency, throughput

- **Given cell library**
  - set of cells in standard cell library
  - fan-out constraints (maximum number of gates connected to another gate)
  - cell generators

Synthesis Flow



Develop HDL files

Specify Libraries

Library Objects
link_library
target_library
symbol_library
synthetic_library

Read Design

analyze
elaborate
read_file

Define Design Environment

Set_operating_conditions
Set_wire_load_model
Set_drive
Set_driving_cell
Set_load
Set_fanout_load
Set_min_library

Set Design Constraints

Design Rule Constraints
set_max_transition
set_max_fanout
set_max_capacitance

Design Optimisation Constraints
Create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area

Select Compile Strategy

Top Down
Bottom Up

Optimize the Design

Compile

Analyze and Resolve Design Problems

Check_design
Report_area
Report_constraint
Report_timing

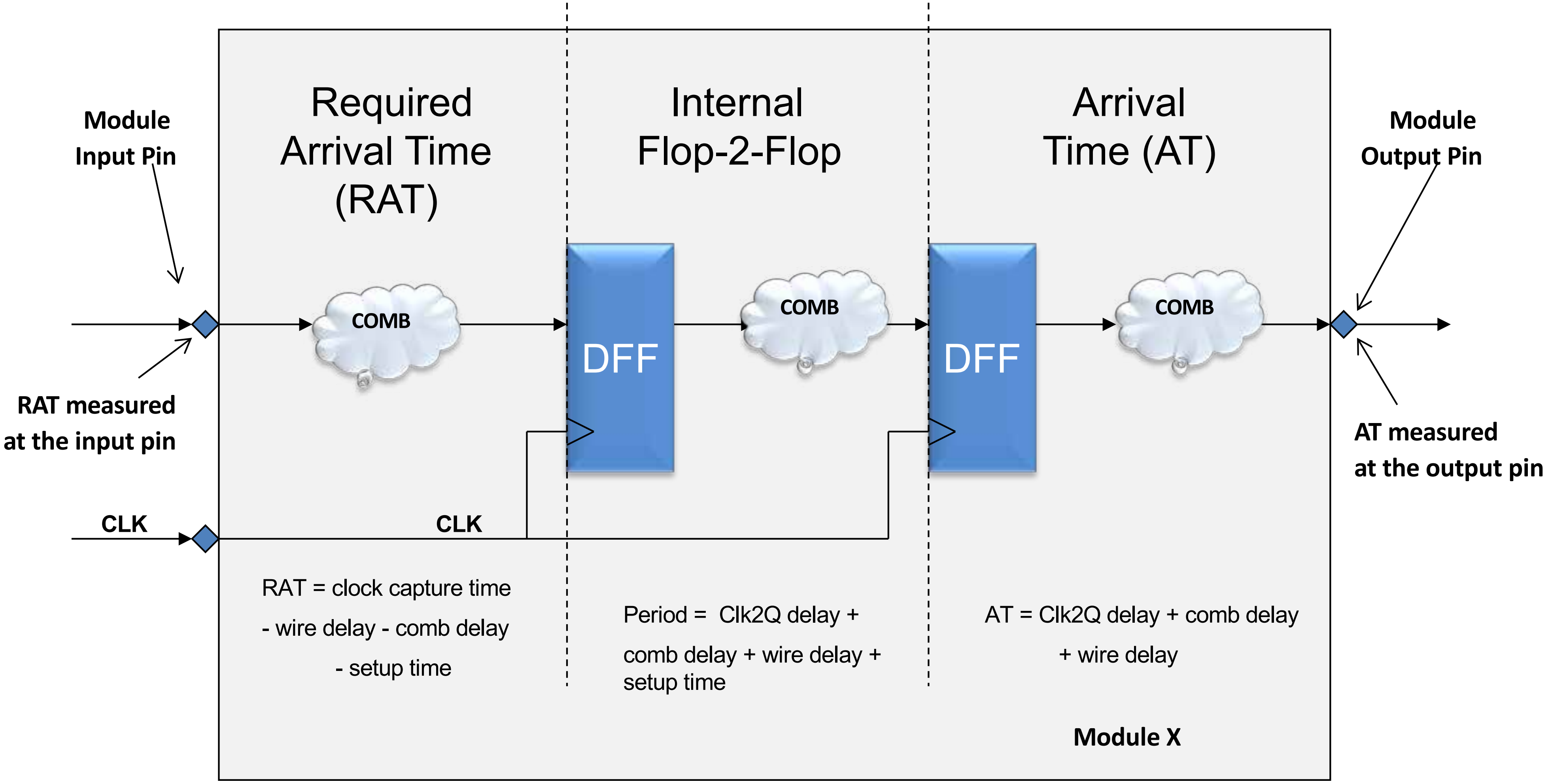
Save the Design database

write

Agenda

- Overview of Logic Synthesis
- **Overview of Static Timing Analysis**

Basics of Timing: AT, RAT, Cycle time



RAT = clock capture time

- wire delay - comb delay

- setup time

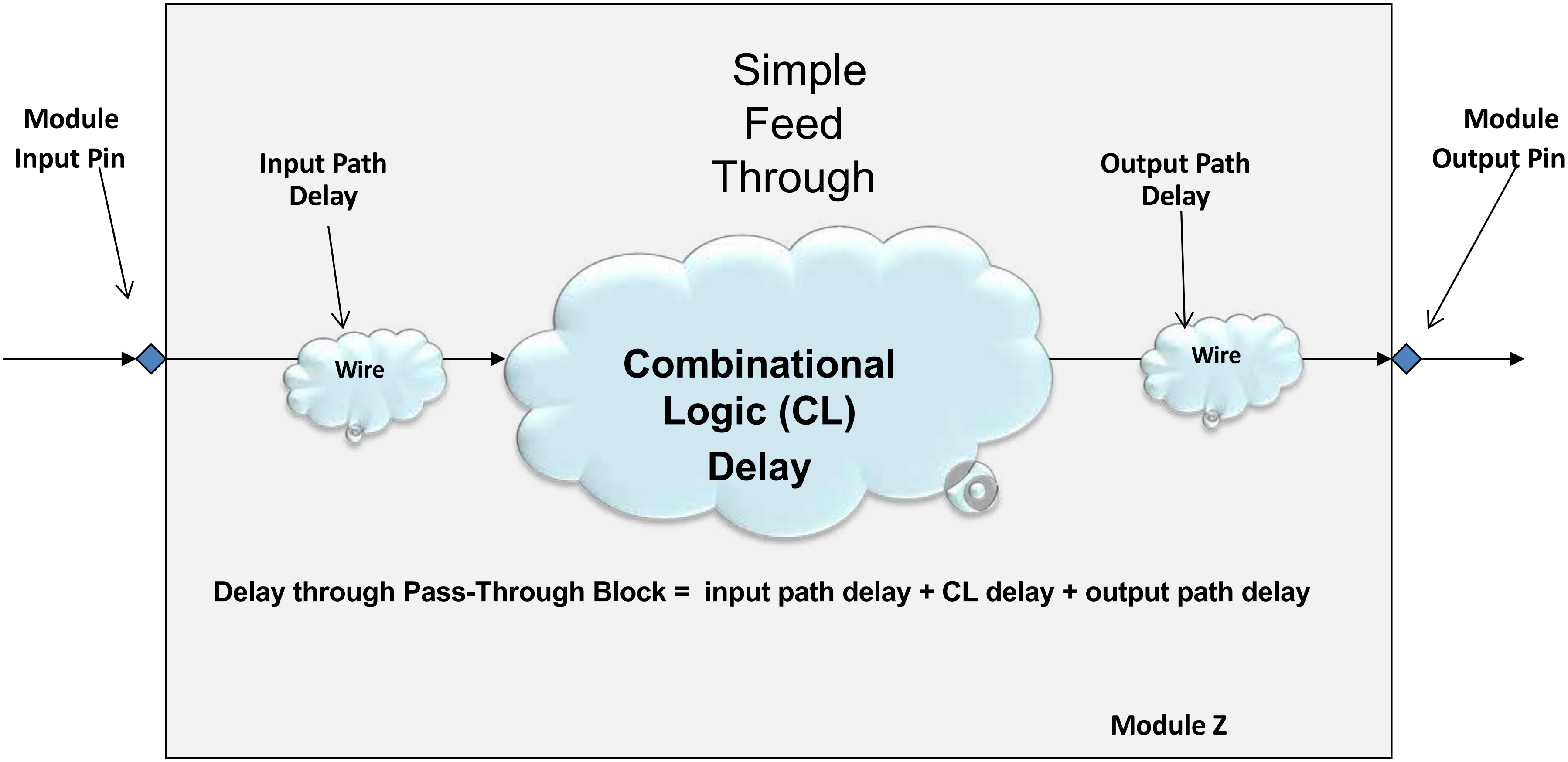
Period = Clk2Q delay +

comb delay + wire delay + setup time

AT = Clk2Q delay + comb delay

+ wire delay

Basics of Timing: Pin-2-Pin (Pass-through)



Delay through Pass-Through Block = input path delay + CL delay + output path delay

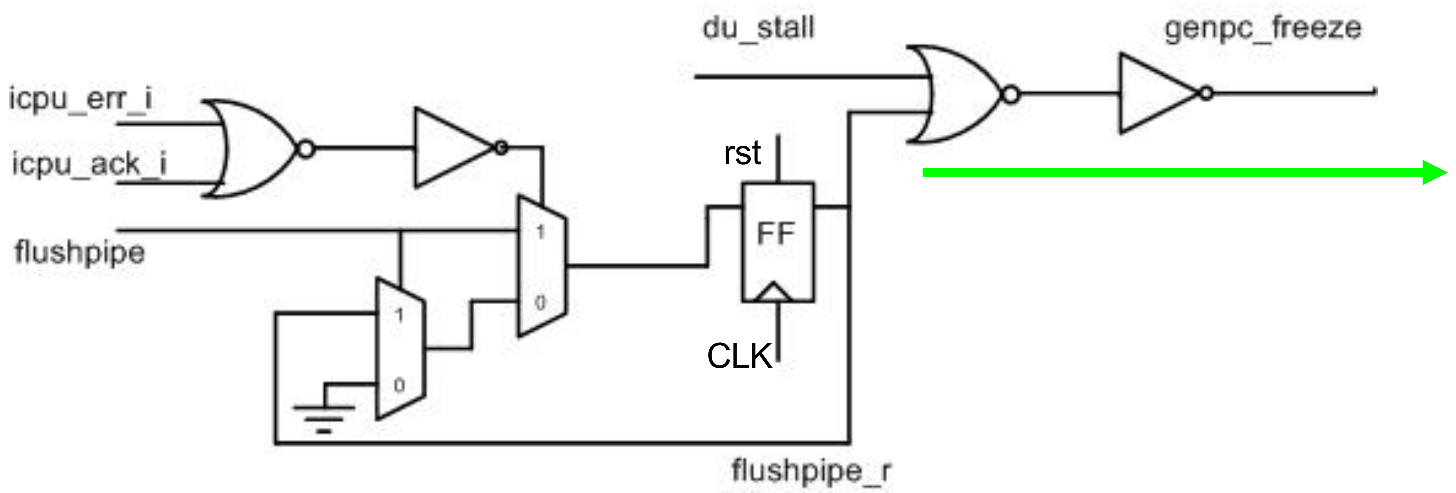
Example

We will walk through the below code to show how to calculate pass-throughs, RATs and ATs.

```
input           du_stall;
input           icpu_ack_i;
input           icpu_err_i;
input           flushpipe;
output          genpc_freeze;

reg             flushpipe_r;

assign genpc_freeze = du_stall | flushpipe_r;

always @ (posedge clk or posedge rst)
   if (rst)
      flushpipe_r <= 1’b0;
   else if (icpu_ack_i | icpu_err_i)
      flushpipe_r <= flushpipe;
   else if (!flushpipe)
      flushpipe_r <= 1’b0;
```

Example – Arrival Time (AT)

Computing Arrival Times



flushpipe_r is launched by a FF. Clock2Q delay is 134.7ps

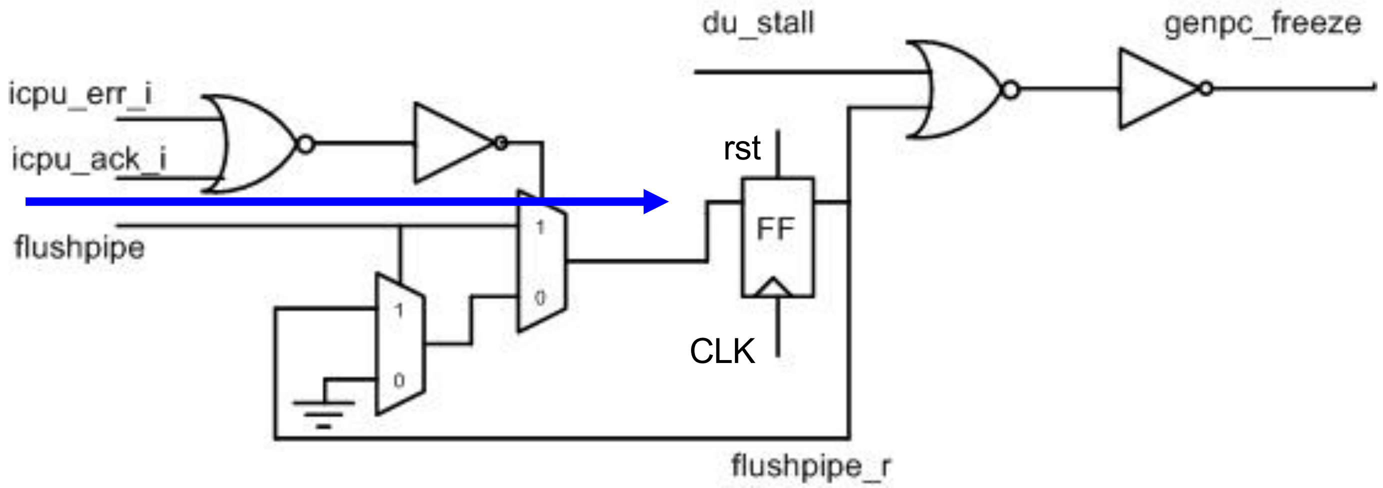
flushpipe_r goes through a NOR2 and INV for a delay of 72.28ps

Total Arrival Time is: Clock2Q + Logic Delay + wire delay = 134.7 + 72.28 + wire delay

Arrival Time for genpc_freeze is: ~208ps + wire delay

Example – Required Arrival Time (RAT)

Computing Required Arrival Times



RAT for icpu_err_i and icpu_ack_i includes delay through a NOR, INV, MUX, as well as the setup time to the FF

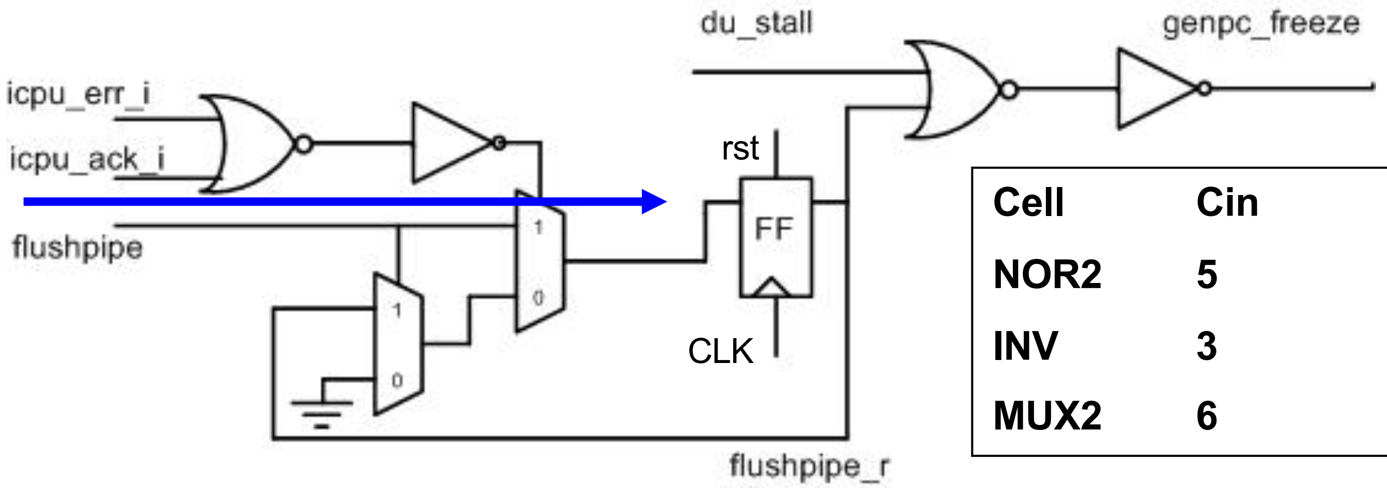
RAT for flushpipe includes 2 MUX delays and the setup time (use the worst case here, since flushpipe has 2 paths to the FF).

Since this path is receiving, assume the gates are minimum sizes.

Since we won't have the nice fanout of 3 working for us in this case, it's time for some Logical Effort fun! (Also applies to Arrival Time calculations.)

Example – Required Arrival Time (RAT)

Computing Required Arrival Times



Cell	Cin	G	P
NOR2	5	9	37
INV	3	9	21
MUX2	6	8	32

The g and p values for the NOR, MUX, and INV are listed in the table to the right.

To use, multiply ‘g’ by ‘h’, which is the Cout/Cin value, and add ‘p’

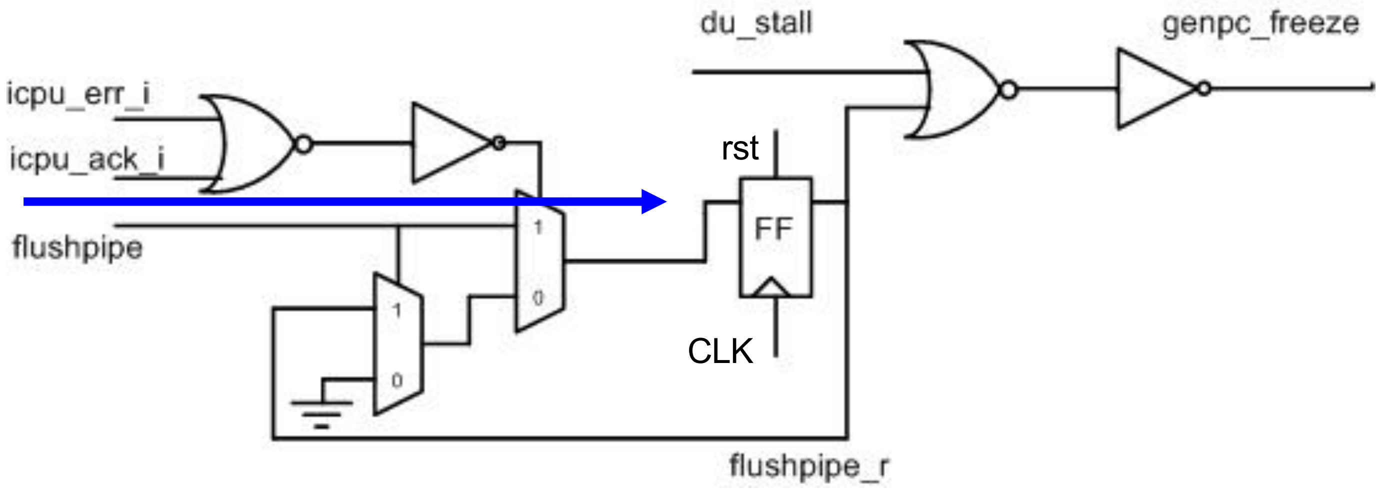
The NOR has h=3/5 (INV/NOR), so its formula is 9*3/5+37 = 42.4ps (note that it’s larger than the FO3 value in the spreadsheet -> this shows that logical effort is not quite accurate…)

The INV is driving a minimum MUX, so the ‘h’ is 6/3 (MUX/INV). Delay = 9*2+21 = 39ps

The MUX is driving a FF (assume cin=6), so the ‘h’ is 6/6. Delay = 8*1+32 = 40ps

Example – Required Arrival Time

Computing Required Arrival Times



We now have everything we need to compute the RATs for the three inputs. <u>Remember that for a RAT, you subtract the delay from the usable clock period!</u>

For icpu_err_i, the RAT is Clock Period - NOR - INV - MUX - Setup = 900 - 42.4 - 39 - 40 - 100 = 678.6ps

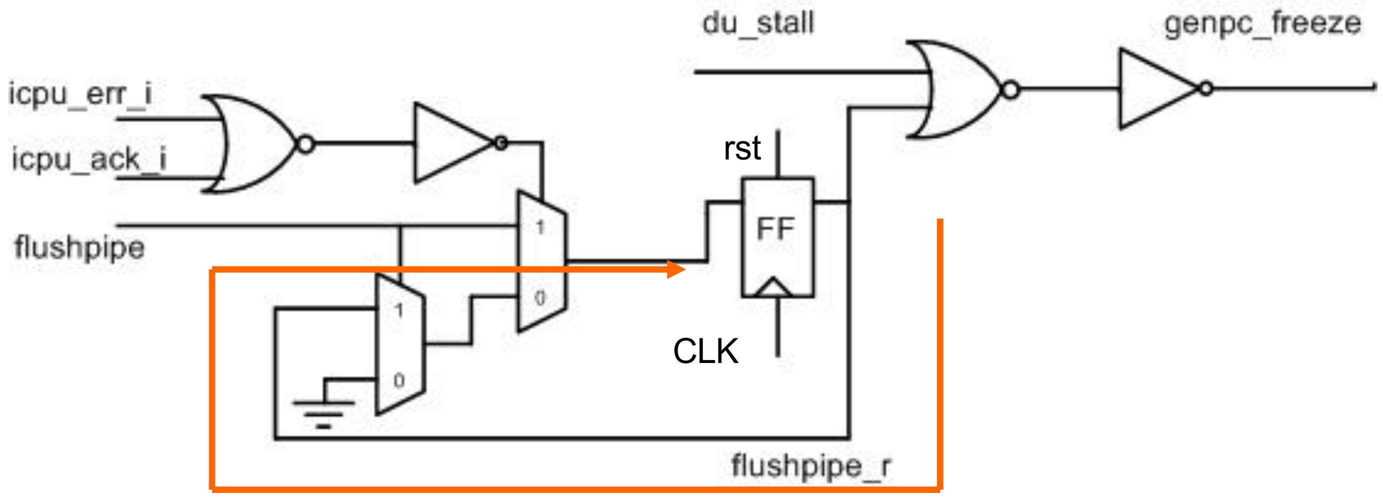
icpu_ack_i sees the same path, so it's RAT is also 678.6 ps

flushpipe sees Clock Period - MUX - MUX - Setup = 900 - 40 - 40 - 100 = 720ps

NOTE that again, these do not include any wire delay!!!

Example – Internal F2F Path

Computing Internal Flop-to-Flop Times



You need to verify that all internal paths meet timing as well.

In this case you would make sure that the C2Q + Mux Delay + Mux Delay + Setup time is less than the clock period (900ps)

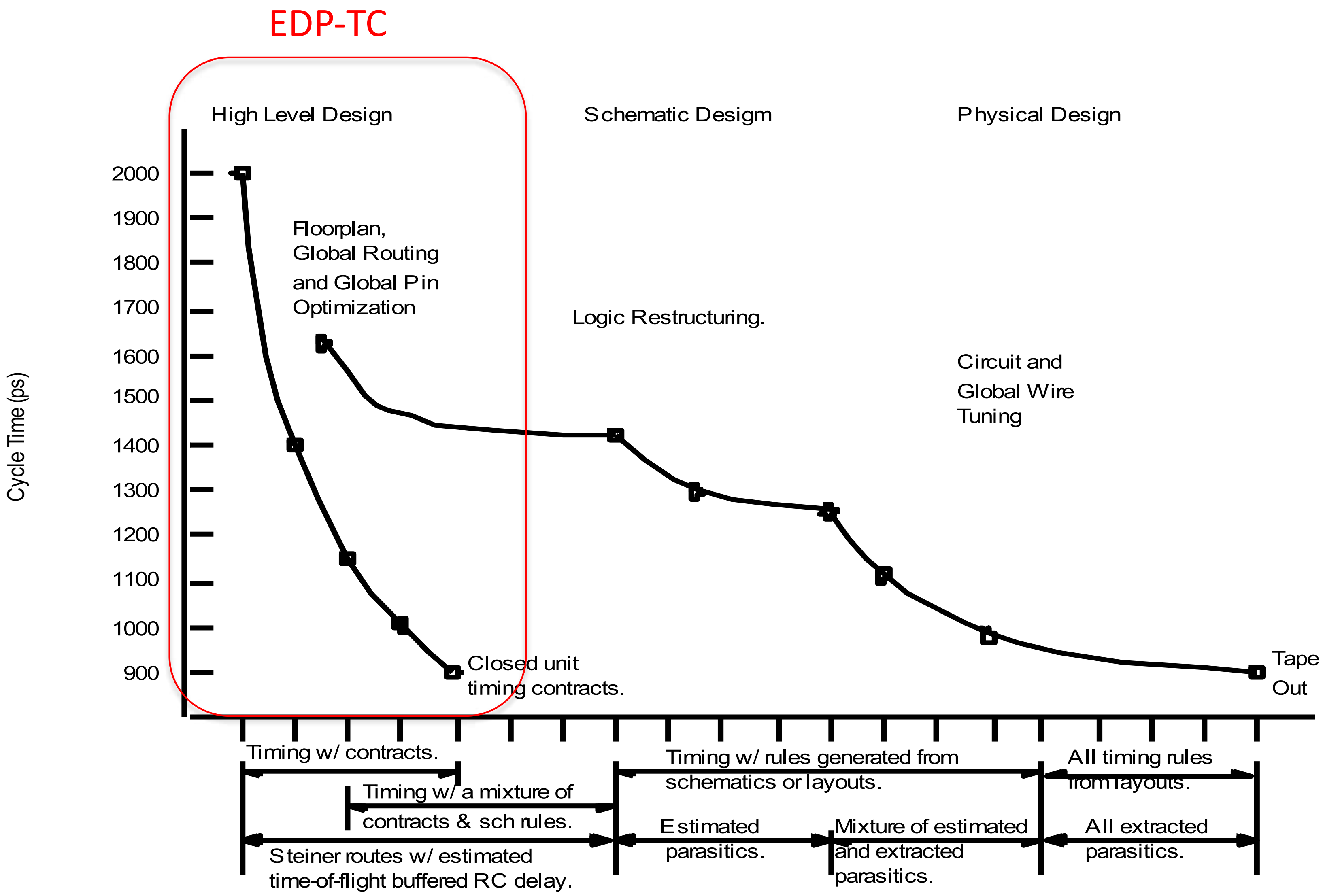
Delay = 134.7 + 40 + 40 + 100 = 314.7 ps < 900 ps -> In this case we meet timing

EDP-TC What Is It?

- The process to identify and close on chip area and timing objectives and constraints during the micro-architectural design phase.

- Rapid Design space exploration during micro-architectural phase
  - Drive changes to the micro-architecture to enable achieving area and timing goals.
  - Enabling Rapid Convergence on Area & Timing closure during design implementation phase.

Typical Timing Closure Progression



EDP-TC Goals & Objectives

- End result is a micro-architectural starting point that is known in advance to have an implementation that can meet the program goals for area, timing and power.
- Get architects and logic designers thinking about physical implementation required to meet the various timing objectives while still in the micro-architectural design phase
- Give designers a methodology & process for:
  - rapidly evaluating the micro-architectural and timing effects of chip physical design decisions (rapid design space exploration).
  - chip floor planning targeted at closing not just area but also all key timing requirements.

Nature of EDP-TC

- Simplified analysis compared to implementation phase
  - Using 1 PVT* late mode timing point
    - Assume monotonic switching per gate (no MIS)
    - Some pessimism built into uncertainty
  - Parasitic loads are estimated and based on placement
    - During implementation phase the goal will be to use extracted parasitics
  - Wires between blocks assume some max edge rate
    - i.e., virtual repeaters, time of flight wire delay calculations
  - All arrival and required times are absolute (class project)
    - All launch/capture pairs assumed synchronous
    - Analysis performed without LCBs

* Process/Voltage/Temperature

EDP-TC Starting Point Data Requirements

- Initial chip size, form factor and I/O requirements.
- Initial chip timing goals.
- Initial top level floorplan-able block list & functionality.
- Initial chip & top level floorplan-able block connectivity.
- For each floorplan-able block
  - initial sizes
  - initial form factors
  - initial pin positions
  - initial timing assertions
- These initial starting points normally evolve during the EDP-TC process.

EDP-TC Methodology How-To

- Methodology Overview
- Block Size Estimation (another lecture)
- Block Timing Assertions Generation
  - How do you get the numbers
- Delay Estimation

Methodology Overview (Big Picture)

- Determine chip I/O definition from architectural specification
  - I/O placement (next levels of packaging & system considerations)
- Determine initial cut at top level floorplan-able blocks from architectural and/or functional descriptions and specifications.
- Generate first pass top level netlist specifying interconnection of top level floorplan-able blocks and chip I/O's
- Estimate initial top level floorplan-able block sizes
  - Analyze the block's component parts
    - Use prior implementations of similar functions as a starting point
    - Perform first pass logic realization on some sub-blocks
- Estimate chip size
  - Floorplan-able block area + wiring uplift (~25-30%)

Methodology Overview (Big Picture - cont)

- Produce chip floorplans
  - determine initial form factors
    - block attributes (memory cell)
    - connectivity (bus widths)
    - Wire-ability
- Iterate on floor-plan to close area & timing constraints
  - Given initial floor-plan, estimate timing of top level critical timing paths based on top level connectivity, block placement, and pin placements
  - Modify block form factor, placement, pin placement and architectural/functional description if required to improve timing and or area.
    - Changes to architectural specifications will yield updates to the number of blocks, their sizes and /or form factors, and the netlist (connectivity) of the top level blocks.
- Done when you have an architectural specification and a floor-plan that achieves area and timing goals.

Block Timing Assertions Generation

- **Block Timing Assertions - What Are They?**
- **Usage of Block Timing Assertions in EDP-TC.**
- **Clock Cycle Adjusts (CCA) in slack calculations.**
- **Estimating delays for initial floorplans.**
- **How Timing Contracts (block assertions) are used in the implementation phase of the design.**

Block Timing Assertions --- What Are They?

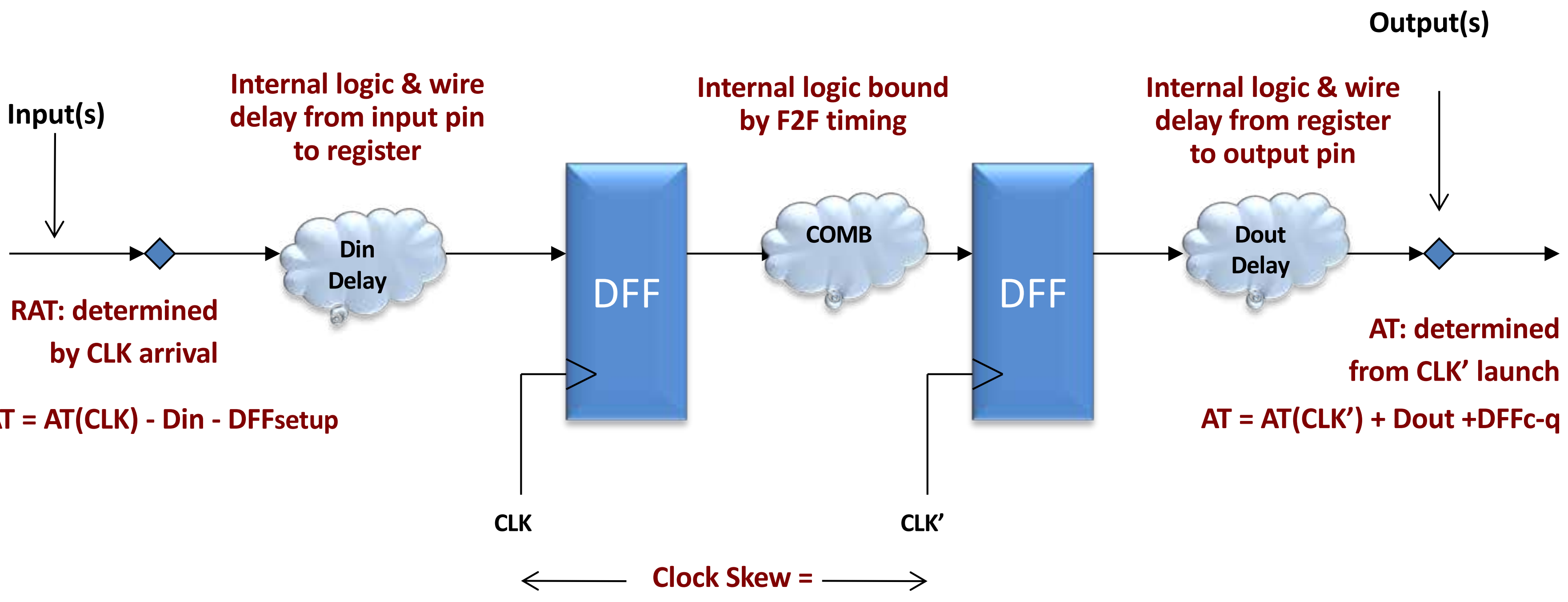
- Basic Block Timing Model
  - Depicts timing information about paths in a particular block
    - 3 types of paths modeled in a block
      - capture: block input to register
      - launch: register to block output
      - purely combinatorial: delay from block input to output

- Basic Block Assertions
  - Input Pin Required Arrival Times (RAT)
    - For each input pin on a block
      - latest time a signal can arrive at that pin and still get successfully captured in the register inside the block fed by that pin.
        - Calculated by: RAT = {AT(clock @ register)} - {Internal logic & wire delay between pin and register} - {register setup requirement}
      - combinatorial: RAT = Need to analyze entire path from register launch to register capture, along with combinatorial delay for the portion of the path inside this block.

Block Timing Assertions --- What Are They? (con't)

- **Basic Block Assertions (con't).**
  - Output Pin Arrival Times: (AT)
    - For each input pin on a block
      - latest time that a signal launched from a register inside the block that feeds the pin arrives at the pin.
        - Calculated by: AT = {AT(clock@register)} + {Internal logic & wire delay between register and pin} + {register launch delay}
      - combinatorial: AT = same problem as combinatorial RAT described on preceding page.
  - Block assertions determined by block alone except for purely combinatorial paths
    - Preferable to eliminate if possible both wire feed-throughs & purely combinatorial paths from all top level blocks.
      - Want assertions & block timing properties to be floor-plan independent to enable rapid iteration.

Path Types Modelled in a Block



Internal logic & wire delay from input pin to register

Internal logic bound by F2F timing

Internal logic & wire delay from register to output pin

RAT: determined by CLK arrival

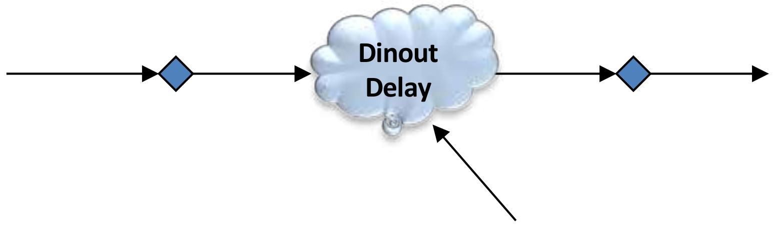
RAT = AT(CLK) - Din - DFFsetup

AT: determined from CLK' launch

AT = AT(CLK') + Dout +DFFc-q

Clock Skew = CLK - CLK'

RAT: determined by capture register block & global wire delay & Dinout



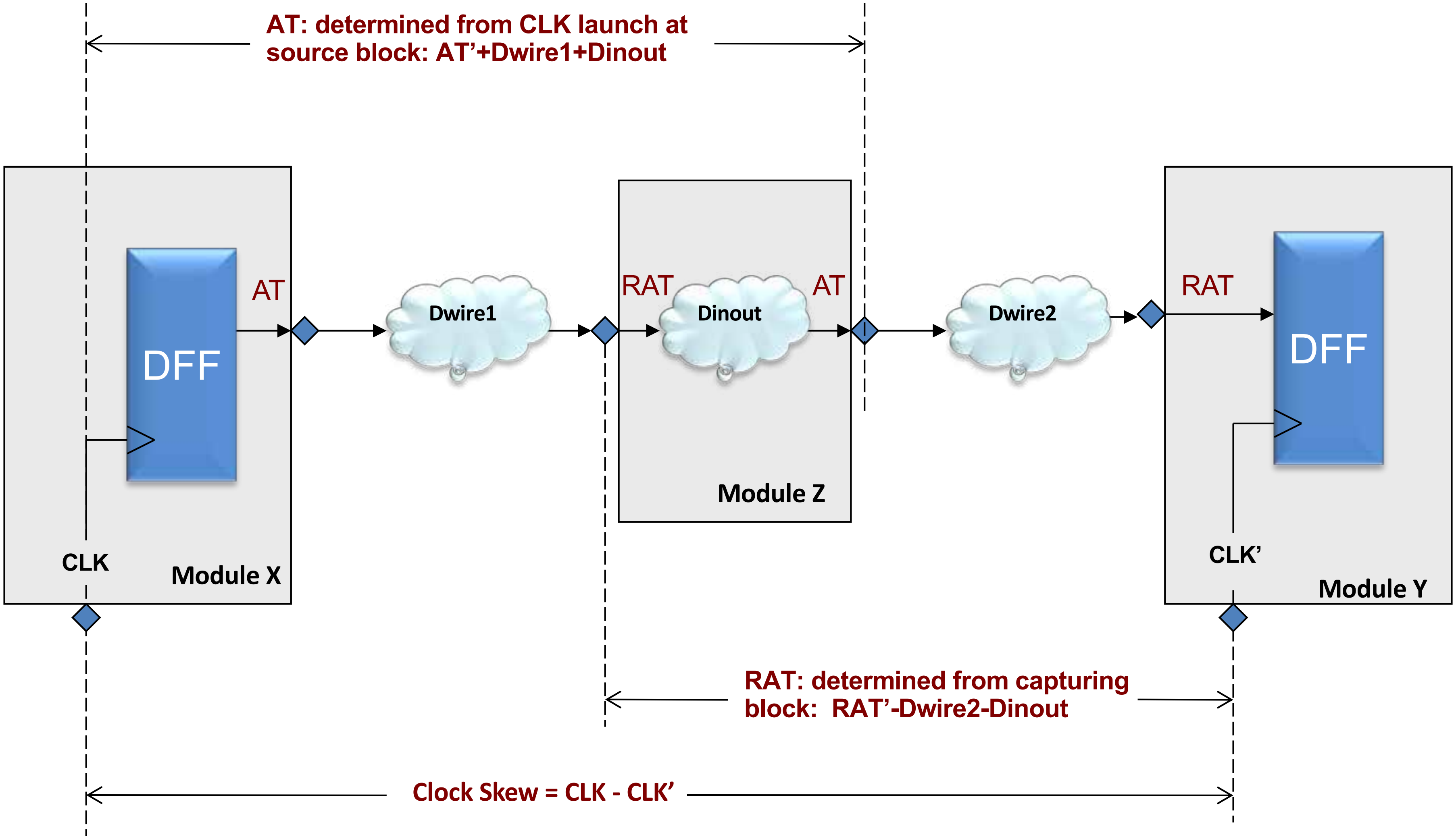
AT: determined by launching register block & global wire delay & Dinout

Delay for wire and combinatorial logic

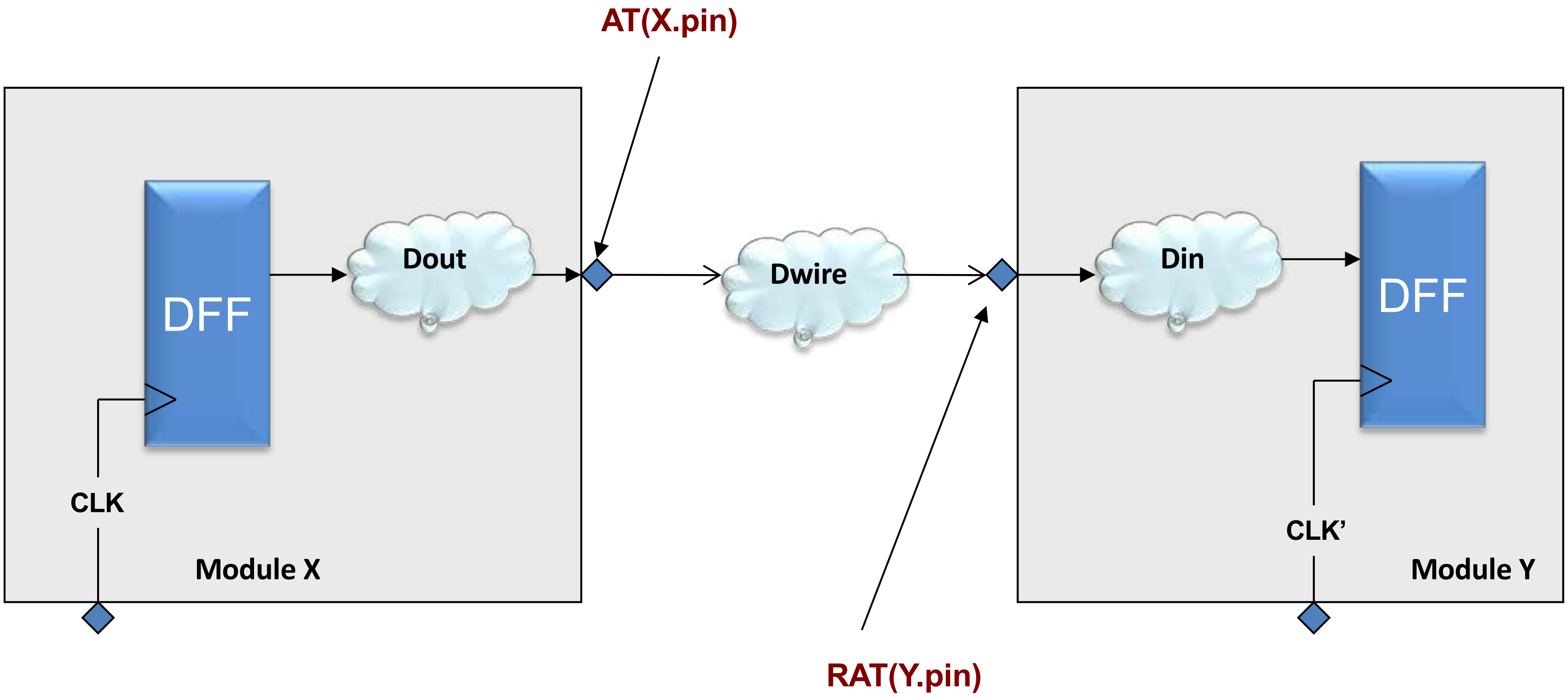
Usage of Block Timing Assertions in EDP-TC

- Every pin of every block and the chip top level block has both an AT and a RAT.
  - Connectivity determines which are combined to determine the slack (timing goodness) of a path.
- Calculate the slack for a path sourced from one block and sunk in another.
  - Avoid purely combinatorial paths and feed-throughs when possible
    - Avoid these at the full chip level
  - Slack calculation must consider phase of launching and capturing clocks in a path
    - all events derived from one cycle of the master clock (ignore multicycle paths for now)
    - no zero cycle setup paths exist
    - A cycle adjustment is made to this calculation when the leading edge of the master clock corresponds to the capture event of the path and the trailing edge corresponds to the launching event.
- When all paths have slack >= 0 the block assertions constitute the Timing Contracts for each block.

Assertion Generation for Combinatorial Paths


AT: determined from CLK launch at source block: AT'+Dwire1+Dinout
DFF
AT
Dwire1
RAT
Dinout
AT
Dwire2
RAT
DFF
Module Z
CLK
Module X
CLK'
Module Y
RAT: determined from capturing block: RAT'-Dwire2-Dinout
Clock Skew = CLK - CLK'

Usage of Block Timing Assertions

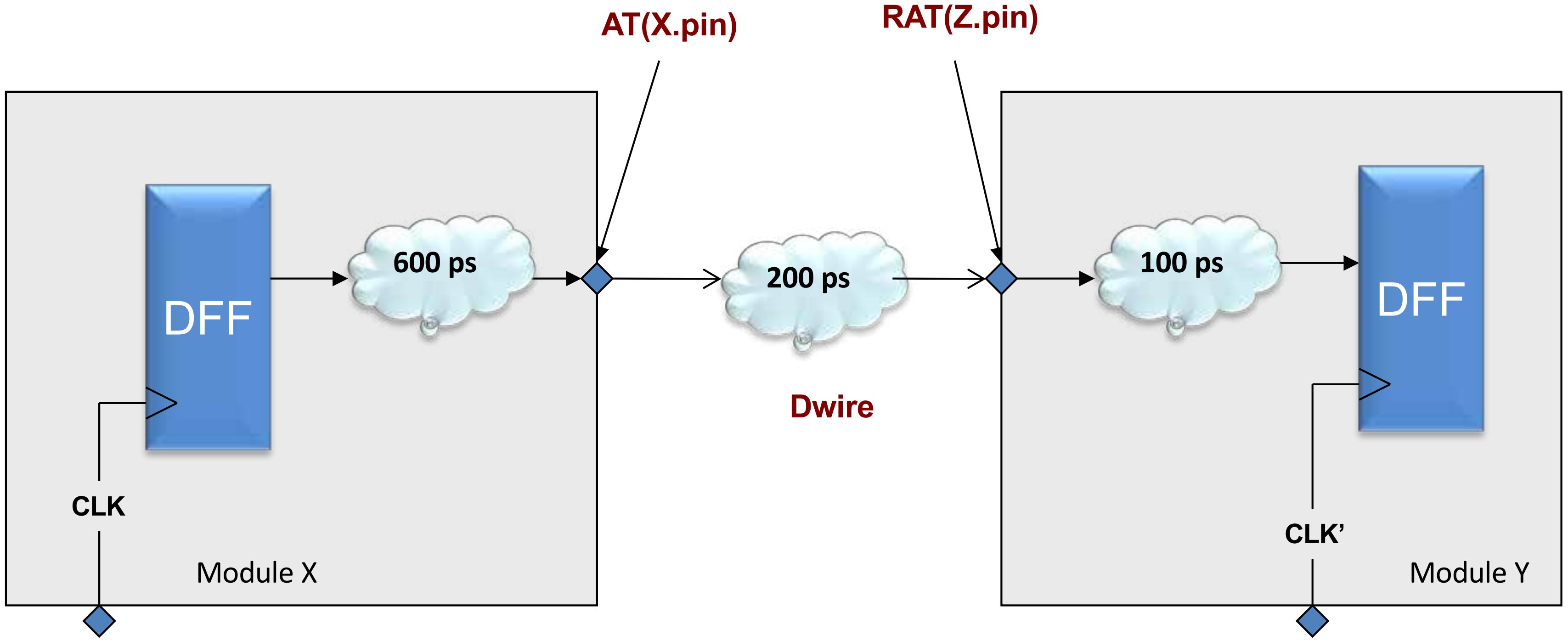


Slack(path of X.CLK->Y.pin) = RAT(Y.pin) - { AT(X.pin) + Dwire } + Adjust

How Timing Contracts are Used

- Implementation phase starts at the end of EDP-TC.
- Given that EDP-TC closed chip timing at 0 slack, the Block Assertions are the Timing Contracts.
- Each block during design is timed stand alone against these contracts, or budgets. Affects synthesis (auto or manual).
  - The RATs are now the assumed arrival times at the blocks inputs.
  - The ATs are now the assumed required times at the blocks outputs.
- The contracts (assertions) are typically periodically updated from full chip timing runs to reflect actual design changes.
  - It's important to continue to have a complete & consistent set of contracts that, if achieved by each block, yields a chip which meets the timing objective.

e.g., Contracts applied to block level timing



Module X Level Timing: RAT(X.pin) = 600

Module Y Level Timing: AT(Y.pin) = T-100

Wire Delay Estimation

- Wire delay calculation & analysis overview.
- Elmore Delay
- Wire Delay Estimation Summary
  - Time of Flight
  - Elmore Delay
  - Lumped RC product
  - RC Ladders

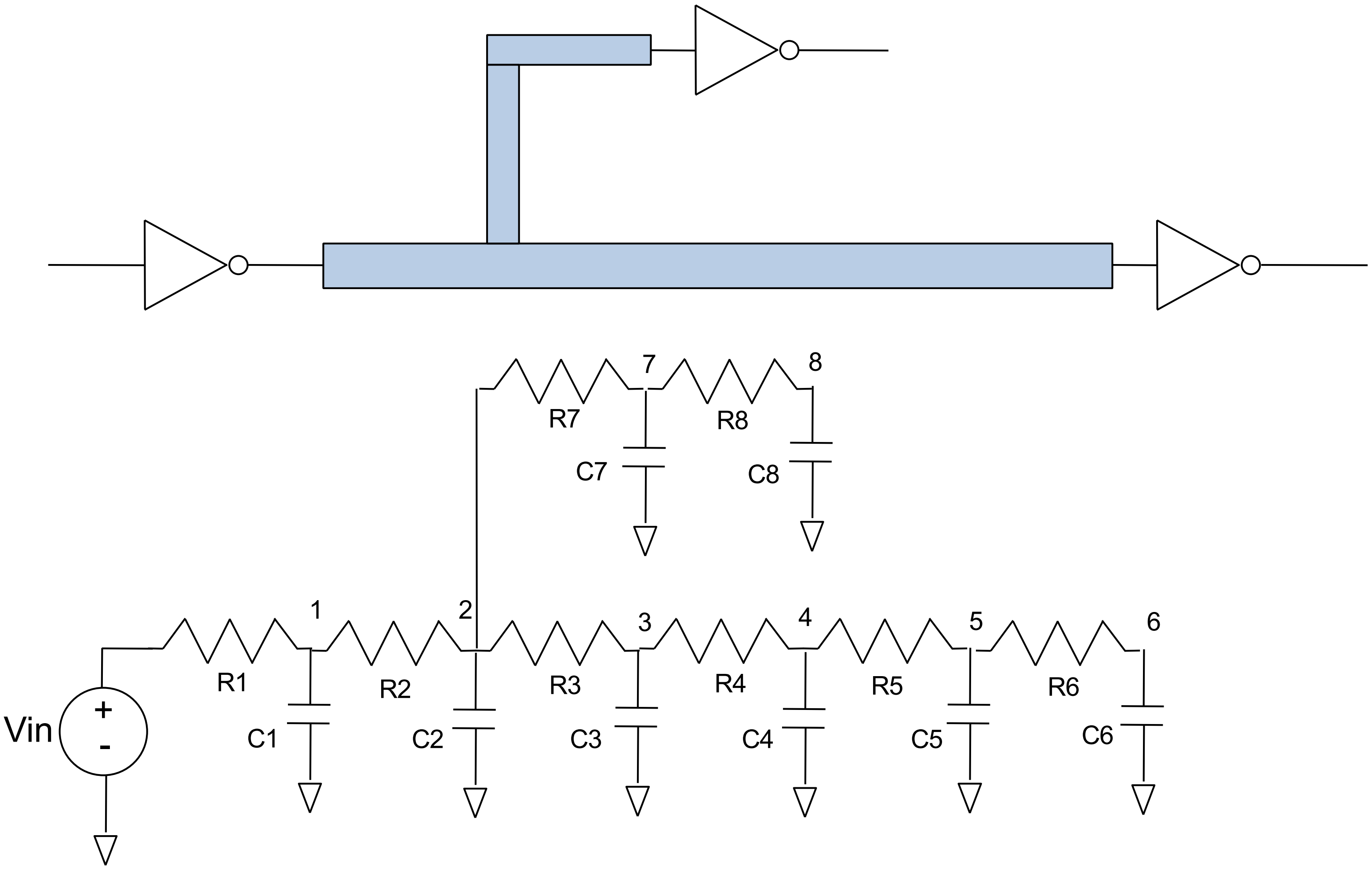
Analyzing On-Chip Interconnect

- Simplified interconnect analysis.
  - Time of Flight (EDP-TC)
    - Simplest approach for EDP-TC.
      - Given in picoseconds per millimetre
      - Assume optimal signal regeneration (buffering satisfies max allowable slew)
    - routing parasitic expressed as some delay per unit distance
    - determined for the process technology with spice simulations
    - assume certain levels of interconnect (parallel plate and fringing fields), coupling, and buffering
  - Lumped RC product
    - Overly conservative for long wires.
  - RC Ladders.
    - Limiting Case, R * C * (Length^2 / 2).
  - Elmore Delay Model.
    - Typically much less conservative from RC Ladders.
    - Effective estimates for Multi-Drop Nets.
- Save more complex analysis for implementation phase
  - shielding, inductance, 3D fields, etc.
  - poles/residues, AWE, 3D field solvers, etc

Elmore RC Delay Calculation Model

- More realistic RC delay than lumped RC for long nets.
- Able to handle multi-drop nets.
- The formula can be written from inspection of the RC tree.
- Calculable in linear time.
- Provable upper bound on RC delay.
  - Can still significantly overestimate RC delay in some cases.

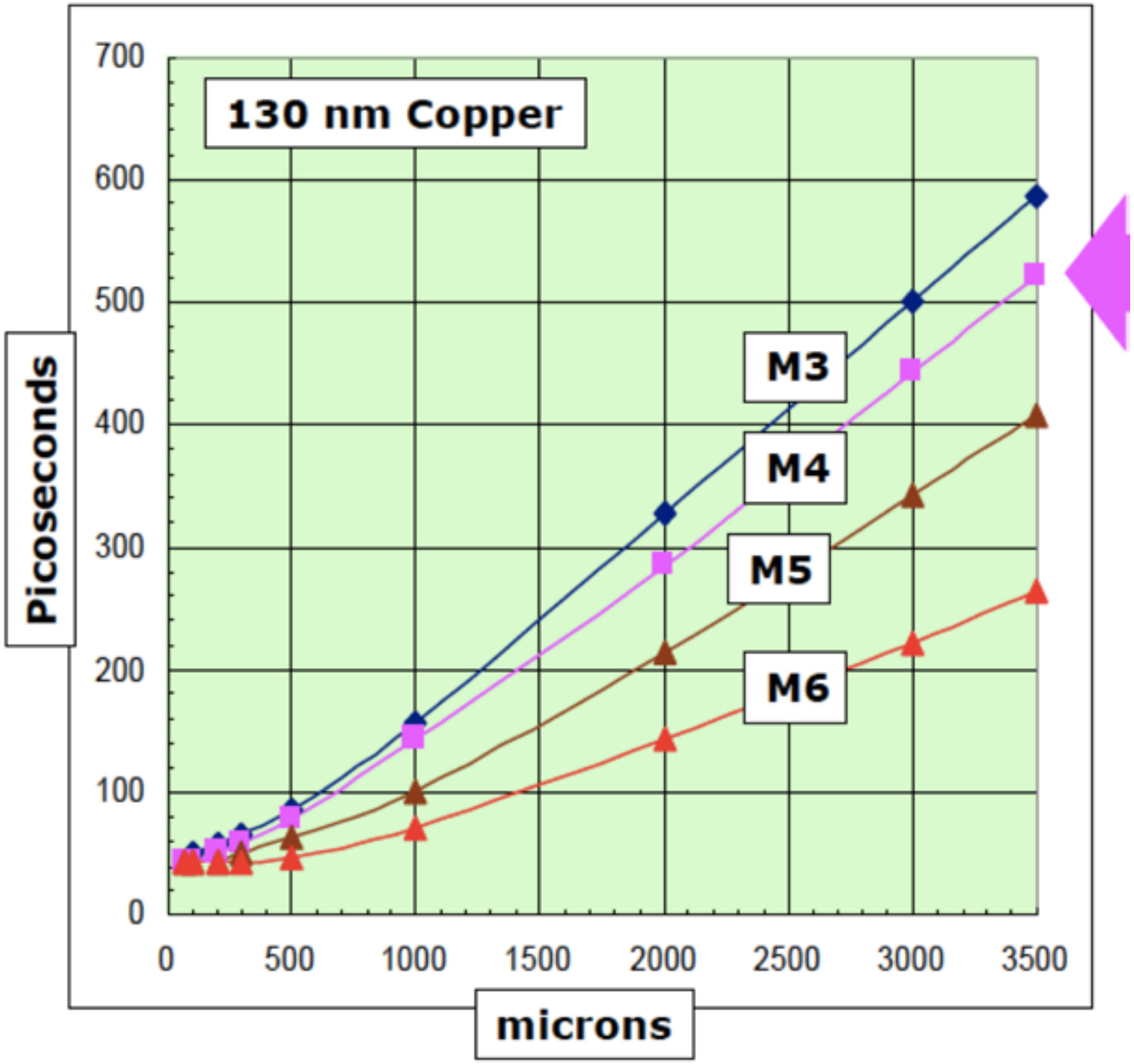
Elmore RC Delay Calculation Model (cont.)



$$Td6 = R1C1 + (R1+R2)(C2+C7+C8) + \left(\sum_{n=1}^{3} Rn\right)(C3) + \left(\sum_{n=1}^{4} Rn\right)(C4) + \left(\sum_{n=1}^{5} Rn\right)(C5) + \left(\sum_{n=1}^{6} Rn\right)(C6)$$

Wire Delay Estimation Summary

- **Time of flight is simplest and probably best for initial floor-plan timing.**
  - Use delay per wire length that considers best estimate for technology, routing layers, coupling, etc. as measured in early circuit analysis (spice)



Wire Delay Estimation Summary (cont.)

- Use Elmore Delay on selected nets as more estimated routing information becomes available
  - Especially if the use of wide wires or upper level metal for low impedance wiring is required to close timing

Summary: EDP-TC End Products

- What comes out of the EDP-TC process
  - Floorplan that will meet basic timing constraints
  - Block size & shape (discussed in another lecture)
  - Pin positions on all blocks
  - Pass through’s on all blocks
  - Timing contracts (assertions & constraints)

References

- http://www.eda.org/
- http://www.asic-world.com/verilog/veritut.html