Section VI: Other Topics

Overview

- Formal verification
- Tapeout issues
- Fixed-timing
- Your call – based on questionnaire responses

Formal Verification

Motivation

- Provide a much faster alternative to simulation/regressions to demonstrate that two netlists are functionally equivalent
- Complete verification, not vector-based
- Verify that RTL matches gate-level
- Verify that gate level ecos back-ported to RTL are equivalent

Motivation (#2)

- Verify that a netlist with:
  - Scan loops restitched
  - Buffer chains inserted
    - Spare gates added
    - Gates resized and remapped

Is still functionally equivalent to the input netlist.

Theory and Concepts

- Compare points: a combinational logic endpoint during verification, such as primary outputs, sequential elements, block-box inputs
- Compare points are usually matched by object names in the netlist

Matching approaches

- Name-based:
  - Exact name
  - Compare point matching based on net names
  - Name filtering
- Non-name-based:
  - Topological equivalence
  - Signature and topological analysis

Compare rules

- Name translation rules created by the user to map object names between two designs
- Uses regular expression syntax

Logic Cone

- When the same compare point can be found in both netlists, the input logic cones are analyzed for functional equivalence
- Logic cones extend from the input to a design object to a primary input or another compare point

Setup

- Inputs are:
  - Two netlists
  - Simulation models of logic components
  - Headers with port directions of black-box components (analog, ram, IP without verifiable models)
  - Setting values on nets to turn off:
    - Gates on clocks
    - Scan enable (if chains reordered
  - Custom compare rules
  - Script to control execution

Execution

- Important to look for errors/warnings when reading simulation libraries to catch:
  - Missing models
  - Illegal models (e.g. behavioral RTL)
- Know roughly how long your comparison should take. When times grow exponentially, this may mean some compare points may not be matching, so some compare rules may be required. Consider setting timing limits on verification

Execution (#2)

- Useful to have the run save its session so that the post analysis data can be recalled, rather than regenerated
- Diagnosing errors:
  - With gui, find compare mismatched compare points
  - Generate schematics for the two cones
  - Have schematics annotated with failing patterns
  - Isolate subcone where input values differ and trace back to source

Hints and Kinks

- Provide as many models as possible (udp's)
- If design requires flattening to match compare points, more memory will be required
- Signatures analysis probably slowest method

Tapeout Issues

Tapeout Issues

- Mask Costs
  - .18u == $250k
  - .13u == $750k
  - .11u == ????
- Metal fill
  - Potential timing impact
  - Problems with tools getting required density

Tapeout Issues (#2)

- What IS the diesize anyway?
  - Seal ring
  - Scribe lines
- Staggered layer tapeouts
  - Preview release a good idea as pipe cleaner (tsmc eyeball BGA pattern)
  - Si Layers first, perhaps met1
  - Other layers later after full checks complete

Tapeout Issues (#3)

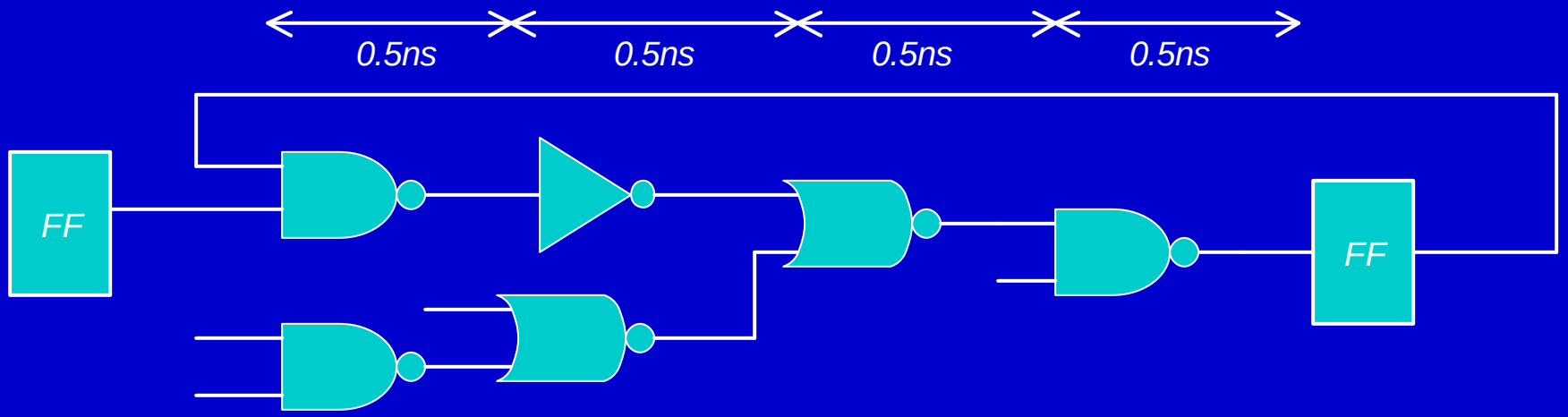
- Metal-only ECO tapeouts
  - Formal verification to check hand edits vs. newly synthesized gate level netlist from new RTL
  - Full LVS rerun , DRC rerun at least on modified layers
  - Layer XOR for unedited layers vs. original tapeout data
  - Timing “ok”, depending on change extent

Tapeout Issues (#4)

- Possible to use synthesis to create ECO from spares, but tricky
- All Layer ECO tapeouts:
  - All verification must be rerun
  - Careful with resynthesis
- Data size: 64bit OS/File system, versions of tar or gnutar may be required

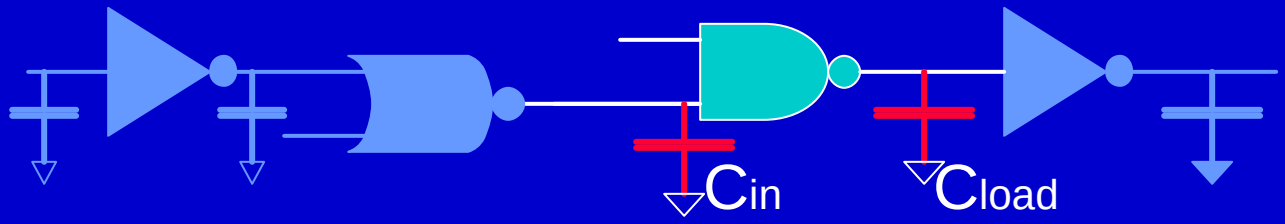
Fixed-Timing

Gain-Based Synthesis (Magma)



- Timing constraint sets up total path delay
- Delay initially divided evenly over the stages
- Stage delay budget is converted into a desired gain
  - Gain fixes ratio of parasitic load to input capacitance
- Gain kept fixed by implicit gate sizing during layout design
- Gain budget (= delay) of the stages adapted continually
  - This ‘gate trimming’ makes all paths critical

Load-Independent Delay Model



"Fast circuit design on a napkin"

Ivan Sutherland (1991):

Delay = (g * h) + p

Delay of the gate + its load

Fixed part, parasitic delay

Logical effort depends on function of gate

Electrical effort proportional to output load

Cload / Cin

Cf. 'Logical Effort' by Sutherland, Sproull, Harris

Morgan Kaufmann publishers, ISBN 1-55860-557-6

Magma View of Logical-Physical Boundary

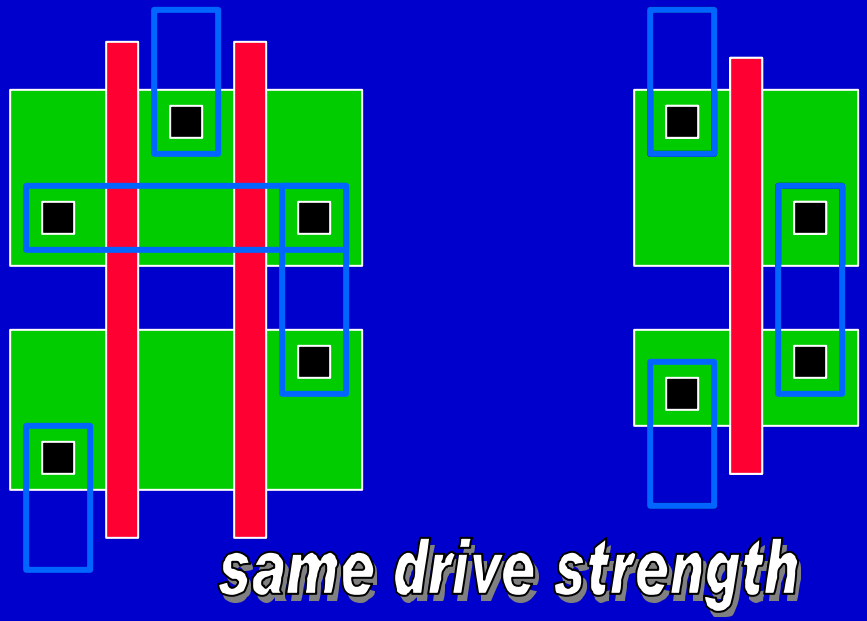
Conventional synthesis

- Cell Area fixed
- Delay is a gamble
- Worst case delay determines timing (max)
- Iterate to make ends meet.
- Only critical paths are addressed. Therefore gates will be too big:
  - waste of area
  - waste of power
  - Cause unnecessary aggressors

Gain based synthesis

- Delay fixed
- Cell Area unknown
  - Sum of areas determines chip size
- No iterations required
  - benefit: speed
- All cells are sized, so each gate has exactly the right drive strength:
  - Not too little
  - Not too much (waste of area)
  - Avoids SI problems

Logical effort: g

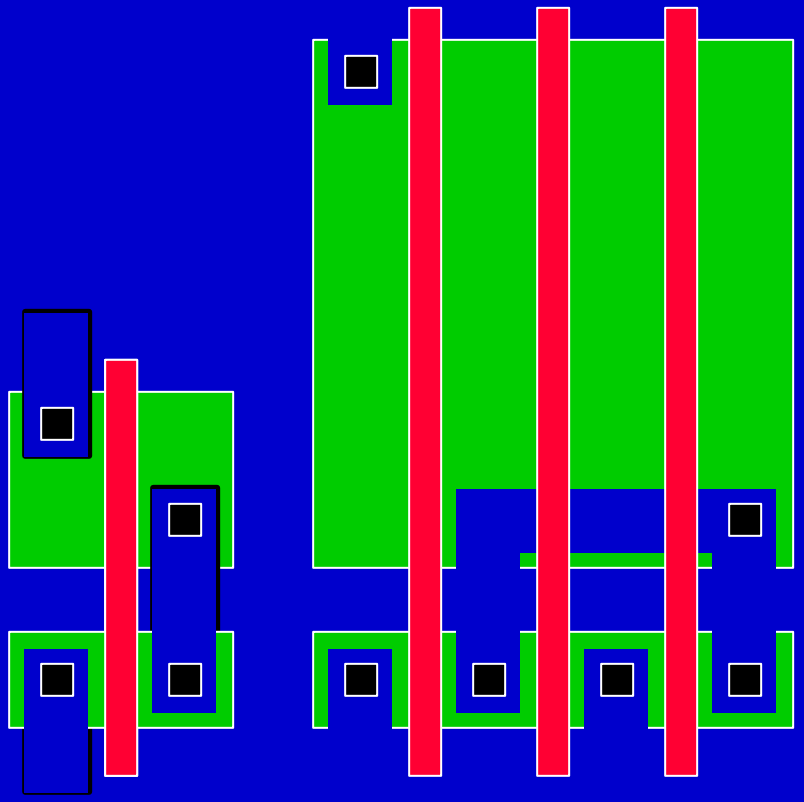


Nand 2: Cin = 4/3 Inverter: Cin = 1

- To keep same output drive strength, the 2 n-transistors in series must double their size.
- As a result, the input capacitance of the nand is larger.
- For the same output drive strength, an inverter needs less input capacitance.
- More complex gates have less gain

Logical effort: g

- 3-input nor

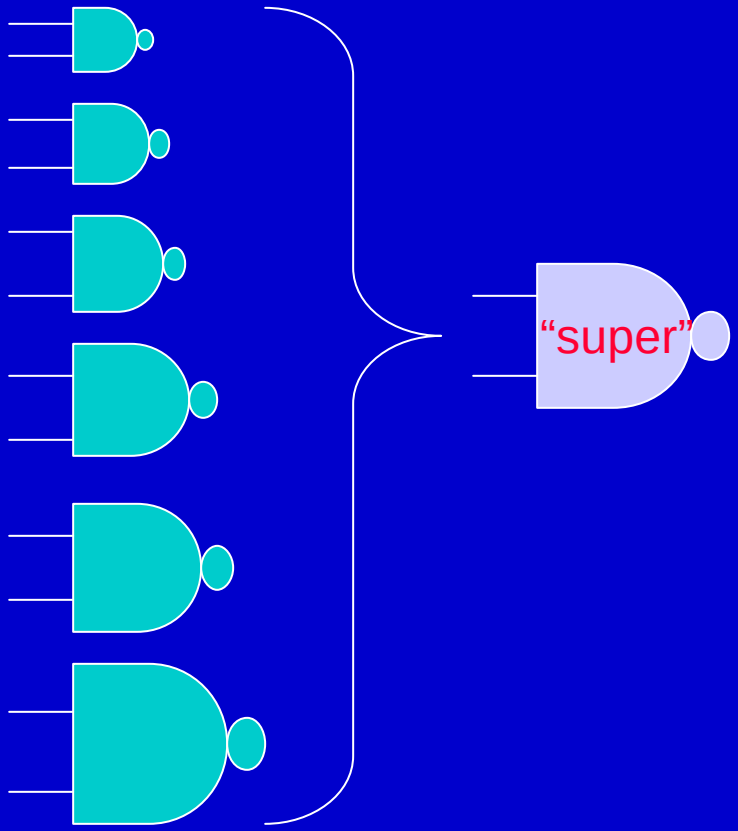


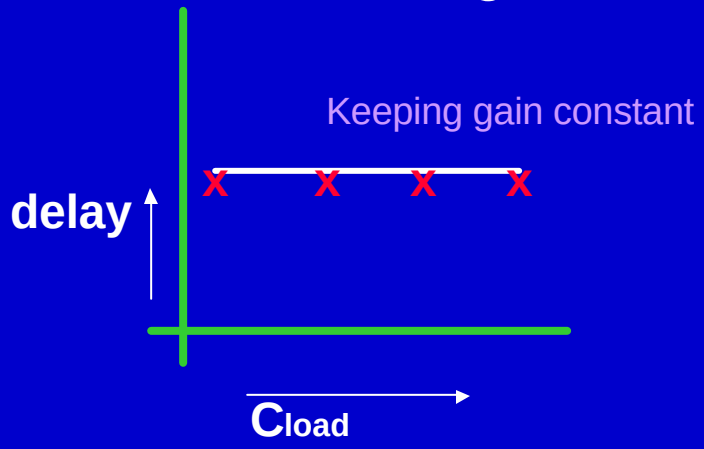
- Assuming that in static CMOS gates the mobility of the p-transistor is half of the n-mobility:

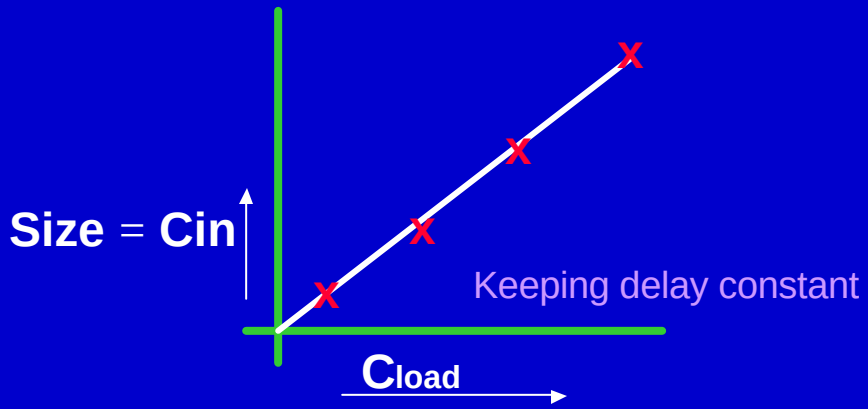
Gate	1	2	3	n
Inverter	1	-	-	-
Nand	-	4/3	5/3	(n+2)/3
Nor	-	5/4	7/3	(2n+1)/3

Gain-Based Synthesis: “Supercells”

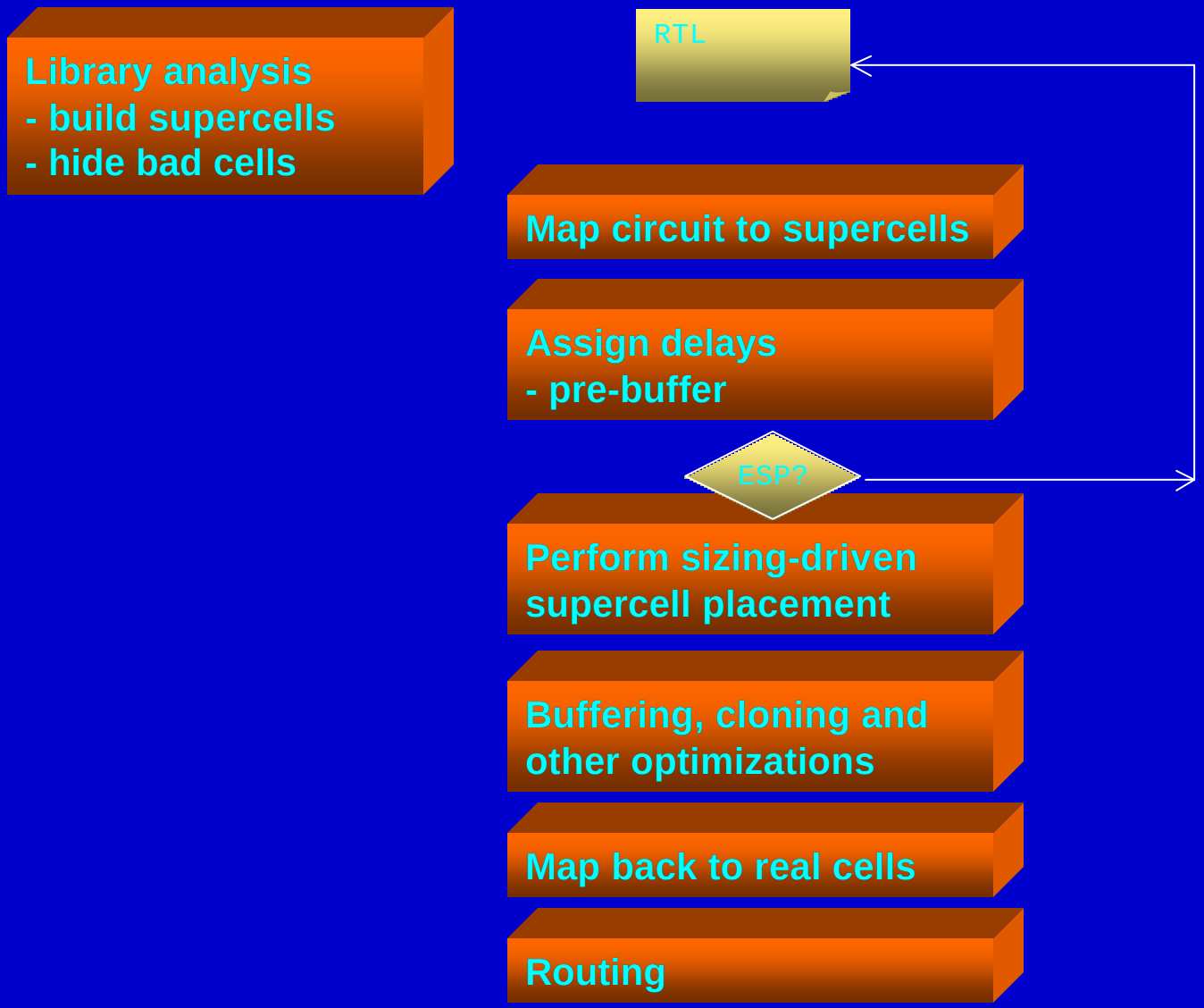
- Gain constant → delay constant over some range







Magma “Fixed Timing”



Other Topics: Your Call !