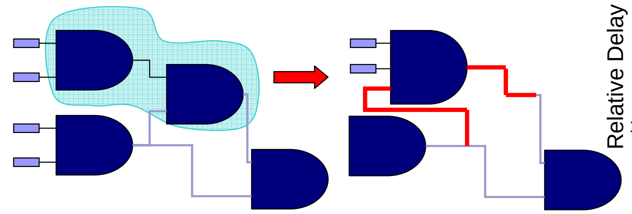
Node Mergers in the Presence of Don't Cares

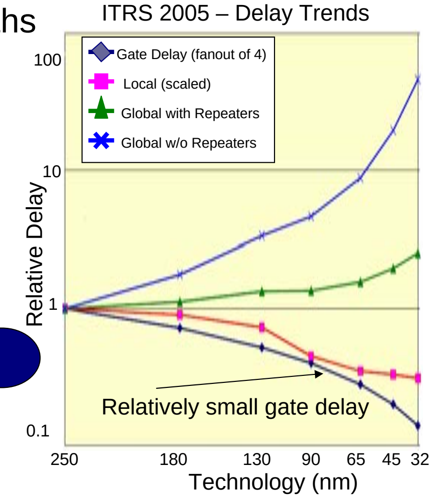
Stephen M. Plaza, Kai-hui Chang,
Igor L. Markov, and Valeria Bertacco

Univ. of Michigan, EECS

Motivation & Context

- Wire delays dominate critical paths (130nm,90,65,…)
- Tech. mapping, place-and-route are key to delay estimation





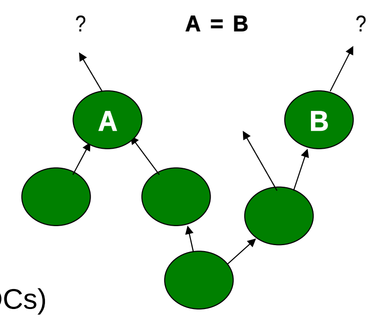
- Dynamic power, leakage

Impact on Design Techniques

- Physically-aware synthesis
  - Minimize impact on placement
  - Cannot assume simple unmapped netlists, e.g., AND/NOT/OR circuits
  - Avoid costly netlist conversions
- Aggressive optimization required
  - Find optimizations post-synthesis

Optimization with Node Mergers

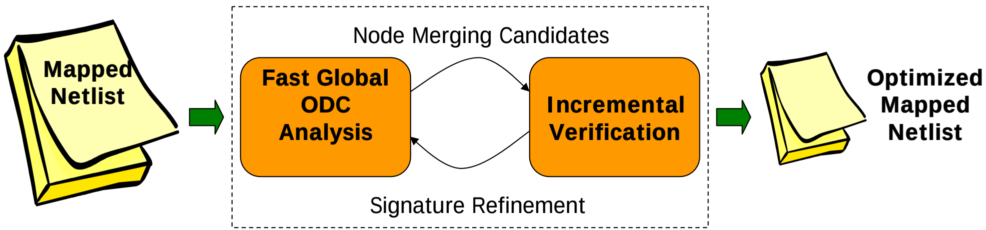
- Merge equivalent nodes
  - Area reduction
  - Eq. checking applications
  - Scalable w/SAT & simulation
  - Exploits satisfiable/controllable don't cares
- Consider downstream logic
  - Exploits observability don't-cares (ODCs)
  - Find more mergers



Node Mergers with Global Don’t Cares

- We implement an aggressive synthesis strategy
- Perform node mergers in the presence of satisfiable/observability don’t cares
  - Not restricted to local don’t cares [Zhu et al. DAC ’06]
  - Focus on post-synthesis optimizations

Our Global Node Merging Framework



Outline

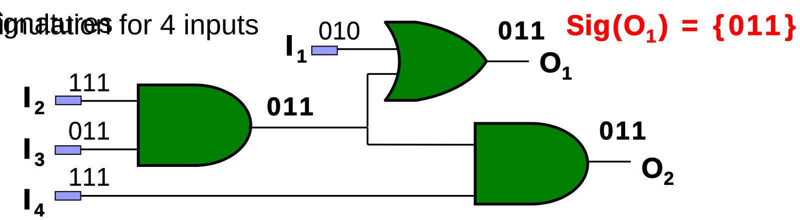
- Background
- Approximate global ODC analysis
- Incremental node merging verification
- Previous work
- Experiments and conclusions

Signatures and Bit Simulation

- **Signature**: partial truth table associated with each node in a circuit
- Stimulate inputs with random simulation vectors
- Generate signatures through bit-parallel simulation

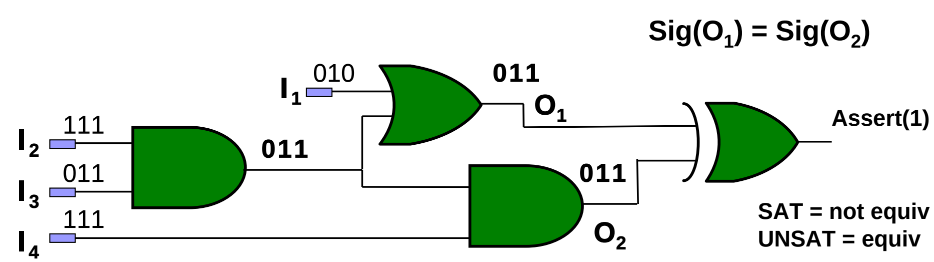
Generate signatures for 4 inputs

3 random simulations



Finding Node Equivalence with Simulation

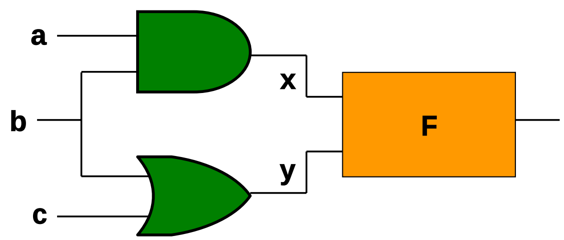
- Identify potential equivalence with signatures
- Verify with SAT—refine simulation if not equivalent
- Applications in verification, And-Inverter Graphs (AIGs) [Kuehlmann et al. ’02, Mishchenko et al. ’06]



Satisfiable Don't Cares

- Input patterns that cannot happen
- Handled implicitly by simulation

No simulation vector for a,b,c generates x = 1, y = 0

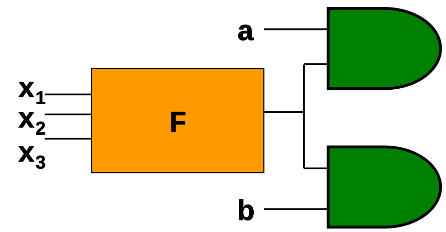


- $F(x(a,b,c),y(a,b,c)) \equiv F(x,y) - SDC(x,y)$

Finding Observability Don't Cares

- Internal value does not affect outputs (limited observability)
- Not accounted for by traditional simulation

F is a don't-care when a=0, b=0



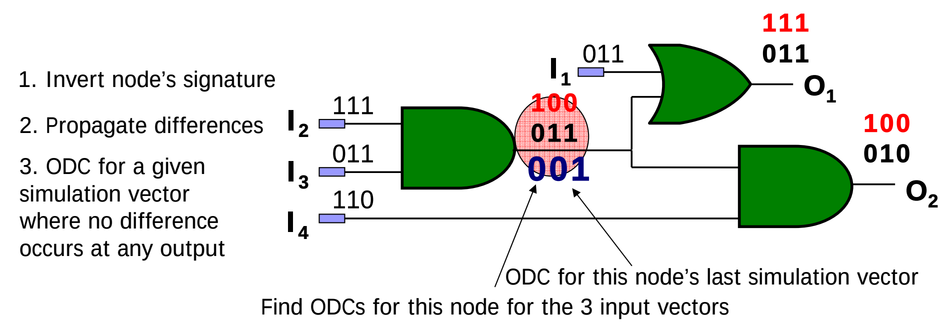
- **ODC-signature: $ODC(F(a=0,b=0,x_1,x_2,x_3)) = 1$**

Outline

- Background
- Approximate global ODC analysis
- Incremental node merging verification
- Previous work
- Experiments and conclusions

Deriving Global ODCs

- Compute ODC signature for each node
- Naïve algorithm: O(n) for one node
  $O(n^2)$ for circuit

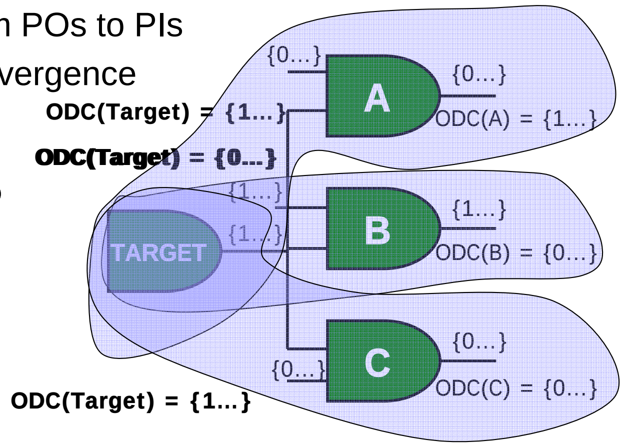


Fast Approximate ODC Analysis

- Linear traversal from POs to PIs
- Exact without reconvergence

Algorithm

1. Examine each of target's FO
2. Union ODC(FO) with local ODC for each FO
3. Intersect ODCs for each FO

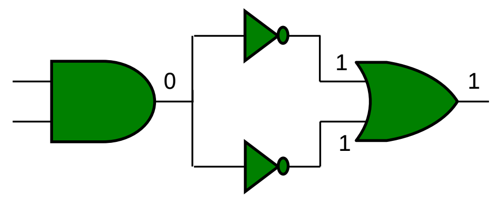


- Less scalable per node computation [Zhu et al. DAC '06]

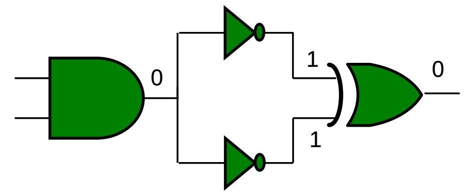
False Positive and Negatives

- Incorrect simulation due to reconvergence
- Happens infrequently
- Verified with SAT

False positive = adding false ODCs



False negative = removing actual ODCs



Identify Merger Candidates

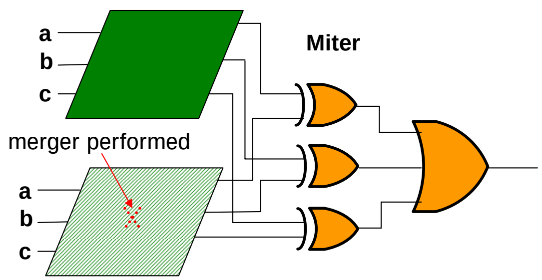
- Find candidate for later verification
- Use ODCs and signatures of each node
- G is a candidate to replace F i.f.f.
  - {Sig(F) – ODC(F)} ≤ Sig(G) ≤ {Sig(F) + ODC(F)}
  - i.e., node G is bounded by function interval of F

Outline

- Background
- Approximate global ODC analysis
- Incremental node merging verification
- Previous work
- Experiments and conclusions

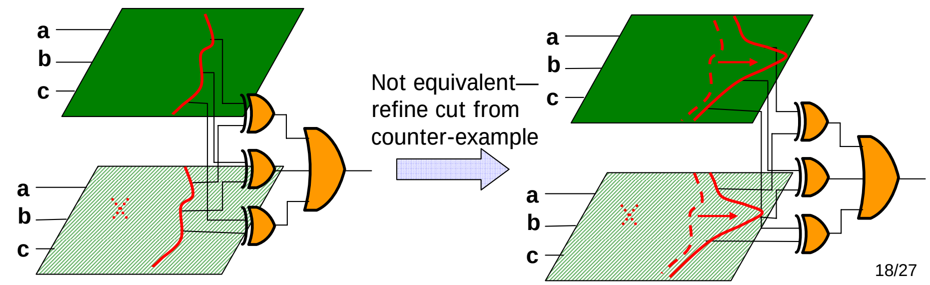
Proving Node Mergers up to ODCs

- Verify mergers indicated by simulation
- Use counter-examples to refine simulation [Zhu et al. DAC '06, Mishchenko et al. '06]
- Naïve approach
  - Merge node in netlist
  - Perform equivalence check over primary outputs



Dominator Algorithm

- Not all downstream logic is necessary to validate a merger
- Our approach:
  - Choose a set of **dominating** nodes from the merger site that form a cut through the circuit
  - Place miters along the cut
  - Run SAT and refine cut as necessary



Finding Dominators

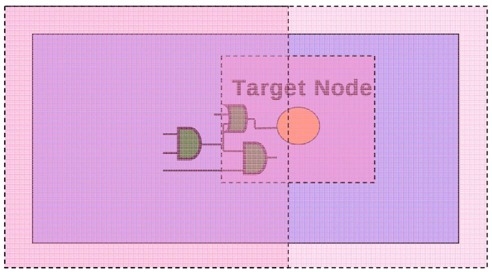
- When merging node G onto F
  - Simulate a subset of the differences between Sig(G) and Sig(F)
  - Find downstream nodes of F where differences disappear
- Similar to finding the D-Frontier in the ATPG domain
- Simulate counter-examples from SAT to extend the cut
- Stopping conditions:
  - The solver returns UNSAT—can merge
  - The solver returns SAT and the simulated differences reach a primary output—can't merge

Outline

- Background
- Approximate global ODC analysis
- Incremental node merging verification
- Previous research
- Experiments and conclusions

Exploiting Don't-Cares

- Previous: primarily local analysis
- Global SDCs through simulation [Goldberg et al. '01, Kuehlmann et al. '02, Mishchenko et al. '06]
- Small windows to exploit local SDCs and ODCs [Mishchenko et al. '05]
- Simulation+SAT to exploit global SDCs and local ODCs [Zhu et al. DAC '06]
  - Local ODCs approximated by considering <6 levels of logic
- Ours: Fast approximate simulation and incremental verification to exploit global SDCs and ODCs



Outline

- Background
- Approximate global ODC analysis
- Incremental node merging verification
- Previous work
- Experiments and conclusions

Experimental Setup

- IWLS '05 OpenCore benchmarks
- Synthesis tool used
  - Local rewriting (Berkeley's ABC package)
  - Simple mapping of 2-input gates
- Combinational sections of circuits considered

Pre/Post-Synthesis Optimization

	Before Synthesis			After Local Synthesis		
Circuit	**#gates**	**#mergers**	**%gate reduction**	**#gates**	**#mergers**	**%area reduction**
i2c	1898	245	13.4%	1055	30	3.2%
pci_spoci	2149	446	23.1%	1058	97	9.2%
systemcdes	4419	812	18.9%	2655	111	4.7%
spi	6440	1091	17.3%	3342	23	1.3%
tv80	14130	2464	18.2%	8279	606	7.1%
systemcaes	17488	3532	21.0%	10093	518	3.8%
ac97_ctrl	24856	3124	12.6%	13178	185	2.0%
usb_funct	28432	4141	15.0%	15514	186	1.4%
aes_core	30875	5729	19.0%	21957	2144	9.2%
average			**17.6%**			**4.7%**

Local vs. Global Simulation (Runtime Comparison)

Circuit (unoptimized)	OA Gear implementation of [Zhu et. al]—levels of downstream logic considered					Our global algorithm (OA Gear)
	2	4	8	16	32	
i2c	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s
pci_spoci	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s
systemcdes	0.3s	0.3s	0.3s	0.5s	0.6s	0.3s
spi	0.4s	0.5s	0.5s	1.8s	11.2s	0.4s
tv80	2.2s	2.3s	2.6s	8.2s	363.0s	2.2s
systemcaes	2.3s	2.4s	2.6s	11.9s	1300.0s	2.3s
ac97_ctrl	1.0s	1.0s	1.0s	1.0s	1.0s	1.0s
usb_funct	2.2s	2.3s	2.4s	2.8s	3.3s	2.2s
aes_core	3.0s	3.1s	3.4s	6.3s	7.9s	3.0s

Global vs. Local Merger Candidates

- Each node can have multiple merger candidates
- More candidates= more flexibility/choices
  - Physical optimizations
  - Timing optimizations

Local merging (5 levels) vs. global merging	
Circuit	%extra global mergers
i2c	24.7%
pci_spoci	11.7%
systemcdes	0.5%
spi	62.6%
tv80	75.8%
systemcaes	98.3%
ac97_ctrl	72.7%
usb_funct	96.9%
aes_core	89.2%
average	**59.2%**

Conclusions

- Optimization before and after aggressive local synthesis
- Fast simulation and SAT = scalable global analysis
- Global analysis = more merger candidates