Advances In Industrial Logic Synthesis

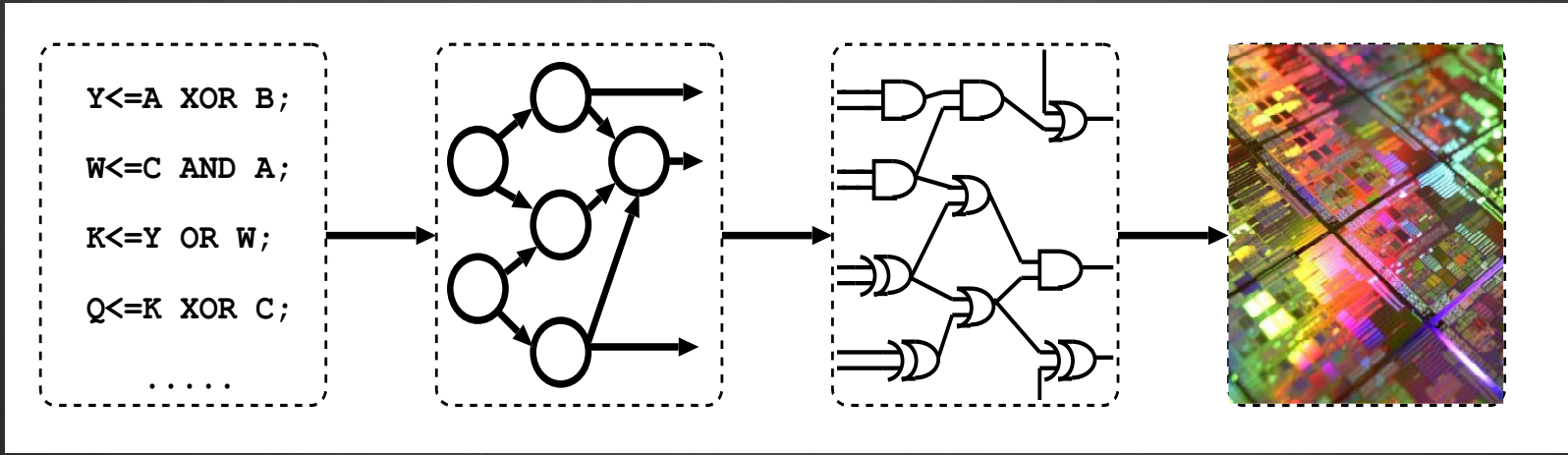
Luca Amarù[★], Patrick Vuillod[✪], Jiong Luo[★]

Design Group, Synopsys Inc., Sunnyvale, California, USA [★]
Design Group, Synopsys, Grenoble, FR [✪]

SYNOPSYS®

Silicon to Software™

Logic Synthesis



- Primary goals of logic synthesis:
- Reduce **delay**
- Reduce **area**
- Reduce **power**
- **With small runtime budget!**

- What is currently a "good deal" for the EDA industry?
- Compare to a complete design flow
- 1% runtime overhead for 10% total negative slack reduction (*at negligible area cost*)
- 1% runtime overhead for 1% area reduction (*at no timing cost*)

How to obtain these challenging goals?

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

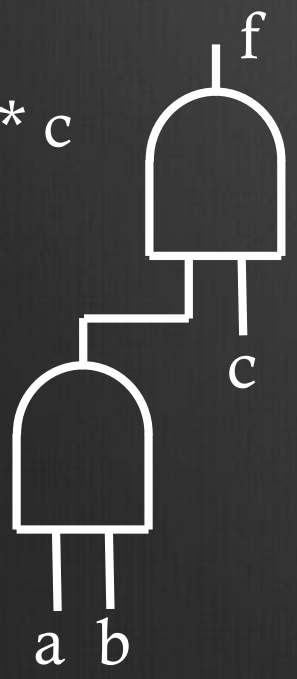
Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Exact Delay Synthesis: The Problem

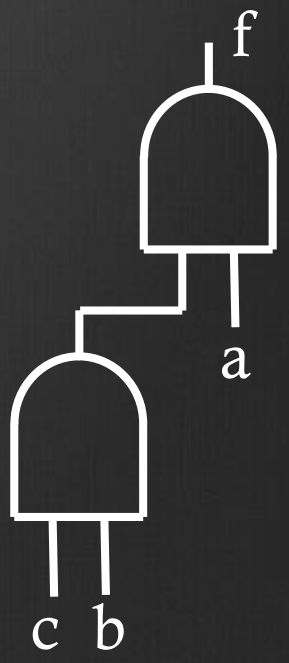
- Given:
  - a Boolean function
  - a set of arrival times at the inputs
  - a gate library with associated delay values
- The ***exact delay synthesis problem*** asks for a circuit implementation with the smallest arrival time at the output(s).

f = a * b * c



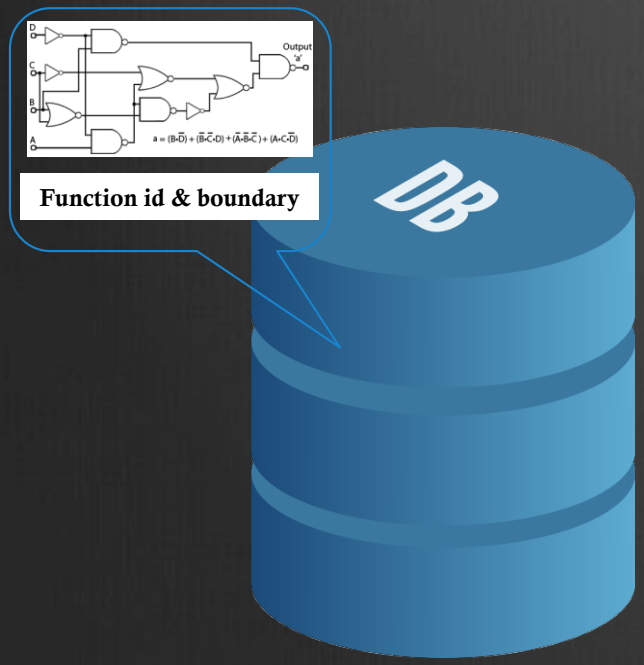
$t_a=4$
$t_b=1$
$t_c=0$

$d_{and}=2$
$d_{inv}=1$



Exact Delay Synthesis: The Problem

- Unfortunately, exact delay synthesis is intractable:
  - Existing design flows rely on heuristic methods.
  - However, heuristic techniques may miss optimization opportunities.
- Affordable solution: pre-compute & store exact delay circuits (database) for later re-use during synthesis.



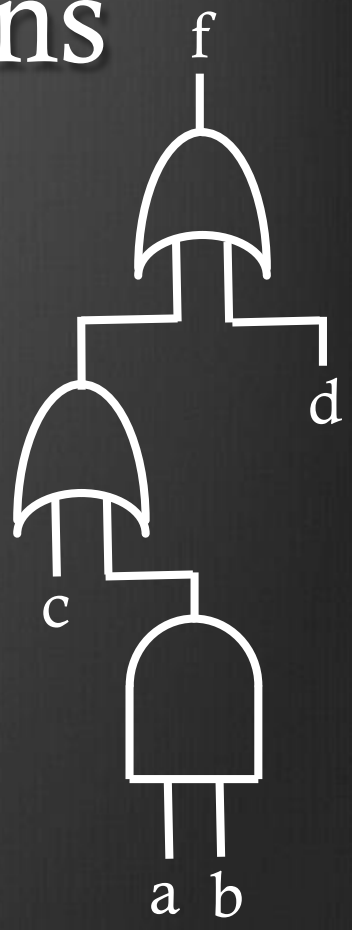
- A database for exact delay synthesis is a $k \times m$ matrix of logic circuits:
  - k is the number of functions
  - m is the number of arrival time patterns
- For n input variables, we have $k \leq 2^{2^n}$,
- m is unbounded as the number of possible arrival time patterns is infinite!

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Equioptimizable Patterns

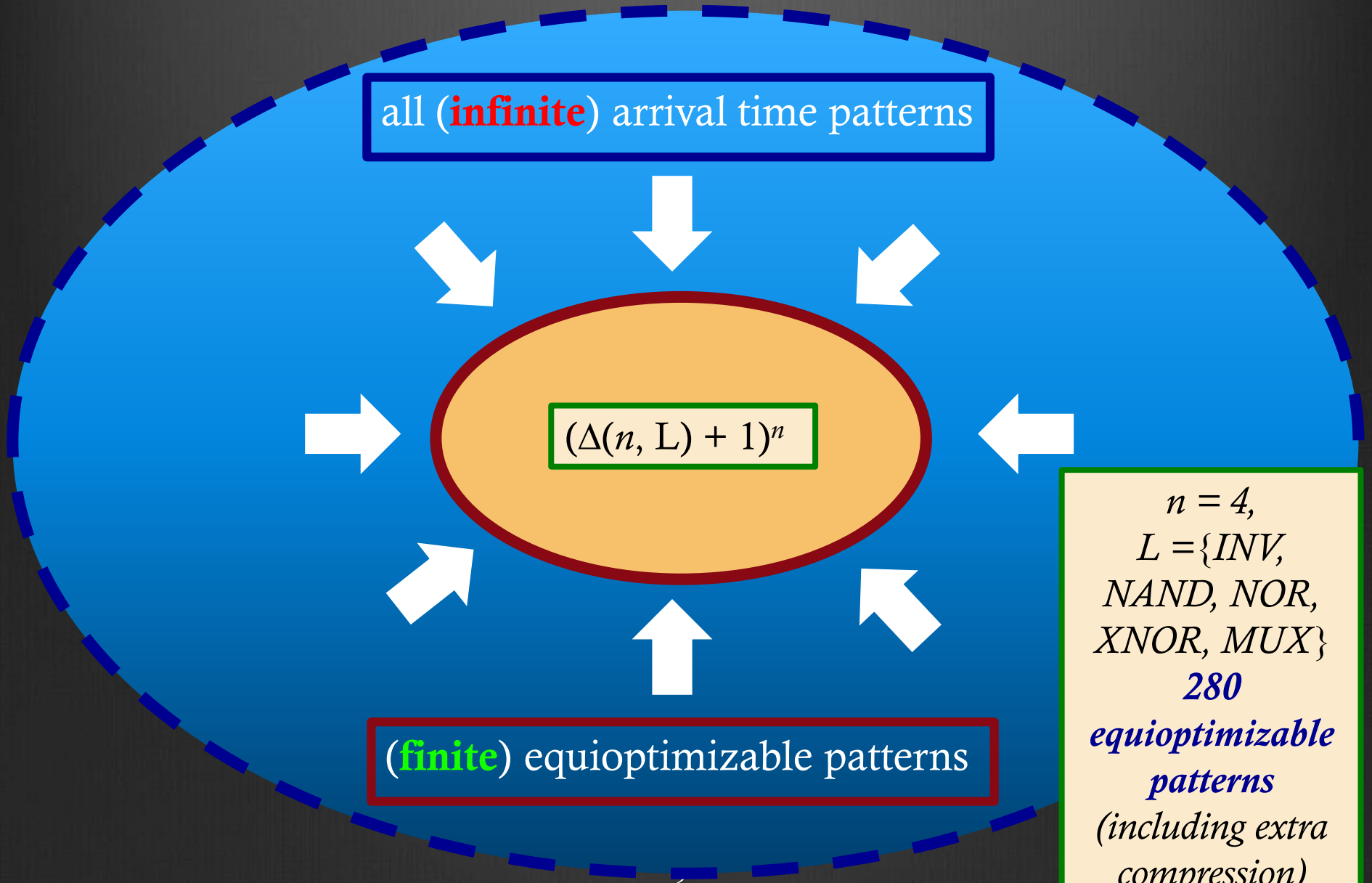
- Address the infinite number of possible arrival time patterns.
- Intuition. Consider:
  - a 4-variables Boolean function $f = ab + c + d$
  - a simple - *unit delay* - gate library L = {AND, OR, INV}
  - an input arrival pattern $T = [0,0,0,10]$
- Would the exact delay implementation for f change with:
  - $T = [0,0,0,100]$
  - $T = [0,0,0,10^6]$
- After a threshold, say $\Delta(n, L)$, the exact delay circuit remains the same:
  - Loose bound - $\Delta(n, L) = n * d_{MUX2:1}(L)$



Equioptimizable Pattern

Compress an arrival pattern s.t.: $t_i = max(t_i, (max(T) - \Delta(n, L)))$

Equioptimizable Patterns



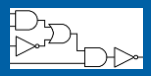
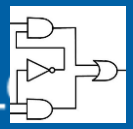
n = 4,
L ={*INV, NAND, NOR, XNOR, MUX*}
280 equioptimizable patterns
(including extra compression)

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Building Exact Delay Databases

- What is an exact delay database?

$$\mathrm{edd}(n, L) = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & \text{  } & \cdots & C_{2M} \\ \text{  } & \vdots & \ddots & \vdots \\ & C_{22} & \cdots & C_{NM} \end{bmatrix}$$

- Entries C_{ij}: exact delay circuits (n, L)
  - Computed with explicit enumeration ($n < 6$) or implicit enumeration ($n \geq 6$)
- Rows: P classes (n)
  - e.g., n=3 has 80 P classes. P classes are known and independent of tech.
- Columns: equioptimizable patterns (n, L)
  - Compress eq. patterns into unique integers in a dense range (interval)
  - From equioptimizable T to: $\Sigma_i\, t_i\, (\Delta(n, L) + 1)^{(i-1)}$
  - Which maps : $[0,\ (\Delta(n, L) + 1)^n - 1]$

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Exact Delay Rewriting

How to use the exact delay database of size *n*?

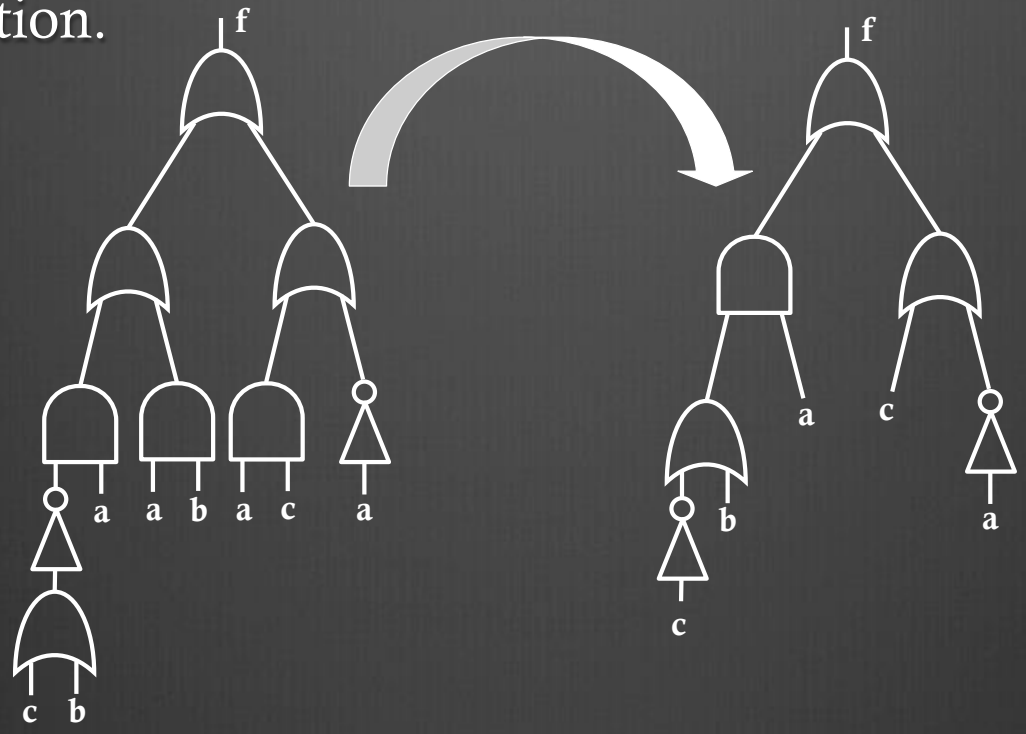
1. For each node of the network, compute all *n*-cuts:
2. For each cut:
   a. Constant time lookup of the exact circuit realization from the database
   b. Evaluate the delay gain (eventually attenuated by the area cost) for the replacement move
3. Evaluate best delay gain:
   a. If best delay gain > threshold: commit the best replacement
4. Iterate 1 as long as there is timing gain or maximum # passes is reached

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Boolean (Re)Synthesis

- Boolean resynthesis aims at improving an existing logic network implementation.



- Boolean resynthesis is capable of stronger optimization than algebraic techniques, but has higher computational complexity.

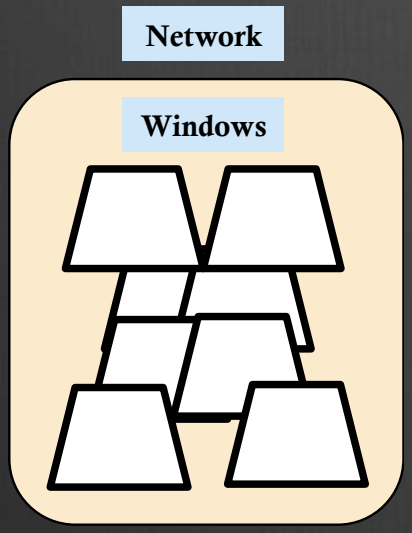
- We revisit fundamental data structures and algorithms for Boolean resynthesis, with focus on resubstitution, MSPF and refactoring.

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Theory of Boolean Filtering (for Resub)

- In principle, *k*-resubstitution can produce optimal logic networks
  - Unfortunately, *k*-resubstitution is (very) computationally expensive
- Example: simple 1-resubstitution using the AND-2 gates:
  - *N* internal nodes in the window, BDD/truth tables for each node.
  - Try to express each one node as AND of two other nodes in the window.
  - $O(N^2)$ equivalence checks for each node, $O(N^3)$ for the whole window.



Complexity gets even worse with higher order resub!

- Many equivalence checks are still spent in verifying null candidates
  - Resub that cannot possibly lead to a valid solution.

Theory of Boolean Filtering

- We aim at spending runtime only on valid resub candidates
- We make use of *canalizing functions* to implement Boolean filtering rules, for example:
  - $f = a * b$ is valid if and only if $f * a = f$
  - $f = a + b$ is valid if and only if $f + a = f$
  - $f = s' * a + s * b$ is valid if and only if $f * s \neq 0$ and $f' * s \neq 0$
  - AOI, XOR, XOAI, MUX-XOR and complex-resub rules can be derived from the general theory of canalizing functions
- With Boolean filtering: better utilization of computing power
  - (Many) more optimization opportunities found at the same runtime cost
  - (Much) less runtime is spent to find the same optimizations as before

```
abc 01> r voter.blif;st;ps
top                    : i/o = 1001/    1  lat =    0  and =  13758  lev = 70
abc 03> resub -K 16;time;ps
elapse: 1.41 seconds, total: 1.41 seconds
top                    : i/o = 1001/    1  lat =    0  and =  11611  lev = 68

xxx> rl voter.blif;ps
top          pi=1001(1001) po=1(1) nodes=13758 terms=13759 lits(sop)=27516
xxx> ttresub -c 16;time;ps
elapse: 1.2 cpu, 55804 Kb memory; total: 1.2 cpu, 55804 Kb memory (K=1024)
top          pi=1001(1001) po=1(1) nodes=10560 terms=11094 lits(sop)=21120
```

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Next Challenges

- With the ideas presented so far, we got (*and exceeded*) the:
  - 1% runtime overhead for 10% total negative slack reduction (*at no area cost*)
    - Exact Delay Synthesis
  - 1% runtime overhead for 1% area reduction (*at no timing cost*)
    - Boolean Resynthesis

- Where to get the next timing, area and power goals (focusing on logic opt.)?
  - *Stronger collapsing*
    - Symmetric normal forms, bdd/sop hybrid forms, etc.
  - *More exact synthesis*
    - Multi-output exact synthesis
  - *Revisit don't cares*
    - Trade runtime for memory
  - *Technology-dependent technology-independent synthesis*
    - Run logic optimization on tech. accurate assumptions and costs

- **More ideas?**
  - This workshop is a great opportunity to discuss with top experts in the field!!

Outline

- Exact Delay Synthesis
  - Theory of Equioptimizable Patterns
  - Building Exact Delay Databases
  - Exact Delay Rewriting
- Boolean (Re)Synthesis
  - Theory of Boolean Filtering
  - *Enhanced Resubstitution*
  - *Generalized Refactoring*
- Next Challenges
- Conclusions

Conclusions (*Take-home Messages*)

- **Exact solutions become more and more available**
  - We are at a point in technology where suboptimal design has a tremendous cost
  - Computing capabilities increased a lot since heuristic techniques have been developed
  - For small/medium instances of the problem, obtaining exact solutions is today runtime affordable
- **Time to revisit Boolean techniques!**
  - We have seen:
    - theoretical developments
    - practical improvements
  - making Boolean techniques runtime affordable (w.r.t. algebraic counterparts)
  - QoR is much better
- **Never stop exploring logic and its optimization!**
  - Physical design becomes increasingly important but logic remains the "backbone of EDA"
  - Logic optimization is still far from being considered a solved problem
    - the increase in the sizes of problem instances cause existing methods to break down, as to be expected of an intractable problem
  - It is critical to keep looking for new solutions

Questions?

Thank you for your attention!