Multi-level Logic Benchmarks: An Exactness Study

Luca Amaru*,
Mathias Soeken†, Winston Haaswijk†, Eleonora Testa†, Patrick Vuillod*,
Jiong Luo*, Pierre-Emmanuel Gaillardon‡, Giovanni De Micheli†

*Synopsys Inc., Design Group, Sunnyvale, California, USA
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
‡LNIS, University of Utah, Salt Lake City, Utah, USA
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions
The EDA community heavily relies on benchmarks to evaluate the performance of academic and commercial tools.

Logic optimization and synthesis are core EDA tasks where benchmarking is essential.
The first set of combinational benchmarks was reported in ISCAS’85.

Sequential circuits were added in ISCAS’89.

In 1991, all benchmarks were collected and distributed under the maintenance of the MCNC.
After MCNC, many other benchmark suites were proposed for some classes of EDA tasks:

- High level synthesis
- FPGA
- Physical design
- Testing
- ...

In 2005, a new set of benchmarks for logic synthesis was presented at IWLS.

In 2015, a new set of purely combinational benchmarks was presented at IWLS.

Missing feature: Can we compare against exact synthesis results for multi-level benchmarks?
Exact-size Benchmarks

- Exact size benchmarks
- TCAD paper 2007
- LEKO and LEKU
  - LEKO: known optimum
  - LEKU known upper-bound
- Exactness idea: replicate a small circuit, in a special way, with a known optimum solution
  - The composite circuit is also optimal
- LEKU = LEKO collapsing + gate decomposition
Exact-size Benchmarks

LEKO Example:

$C_5$ circuit

Composite circuit:

LEKO $G_{25}$
Motivation For This Work

- If we want to measure the ultimate performance of logic synthesis heuristics, we need exact results for multi–level logic benchmarks.

- Exact results and benchmarks for size do exist: LEKO & LEKU.

- In this work, we show how to build exact depth benchmarks.
  - Measure the ultimate performance of delay–oriented synthesis algorithms and tools.

- We provide a non–replicative method to build exact depth, multi–level circuits, with:
  - Non–trivial functionality
  - Non–monotone
  - No disjoint support decomposition
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions
Exact Benchmarks

Problem: we want to measure the efficiency of heuristic delay optimization techniques, for large circuits with known optimum results

- Thousands of I/O
- Tens of thousands of gates
- Potentially small delay (but large delay as starting point)

Solution: build synthetic benchmarks that are provably optimal

- Avoid trivial cases
- Avoid monotone circuits
- Avoid disjoint support decomposable circuits
- Add extra complexity
Why Synthetic Exact Benchmarks?

**Question:** why synthetic exact benchmarks?

Generating optimum circuits for arbitrary functions is an intractable problem.

Relates to the MCSP, thought to be at least NP-hard, still unproven.

Relax constraints to make the problem tractable:

- Impose complexity constraints on the function but not on the actual functionality.

The logic synthesis version of Heisenberg’s uncertainty principle:
- The more we want to know the functionality a priori, the harder it becomes to know the exact circuit.
Balanced-tree Exact Circuits

- How to build exact-depth multi-level circuits?
- Let’s start from a simple concept and move from there
- Balanced-tree circuits with distinct inputs as leaves are depth optimal by construction
- Intuitively true, easy to prove

\[ f = abc + abd \]
Are Balanced–tree Circuits Depth Optimal?

Proof by contradiction:

- A balanced tree is a $n$ level implementation for a $2^n$ variables function.
- Let’s assume it is possible to implement the same function in $n-1$ levels

The function realized cannot depend on all $2^n$ variables: hence the contradiction.
Simple algorithm:

- Generate a binary tree with $n$ levels, $2^n$ leaves, $2^n - 1$ nodes
- Populate the nodes (randomly) with AND/OR binary operators
- Populate the leaves with distinct primary inputs
- The root of the tree is the primary output

Problem: the functions implemented are trivial

- Only monotone circuits
- Specialized synthesis algorithms can easily identify such structures
Breaking the monotone property:

**Algorithm 1** Generation of depth-optimal multi-level circuits with disjoint support.

**INPUT:** Complexity measure $n$

**OUTPUT:** Depth-optimal circuit with $2^n$ inputs.

create empty balanced binary tree with $n$ levels;

for each node $n$ do

assign $n$ a random binary operator;

end for

enforce the presence of at least one binate operator;
Disjoint Support Decomposition

- Addressed problem: the functions implemented are not monotone
- Remaining problem: the functions implemented are still easy to synthesize
- Disjoint support decomposition is natively applicable here
Breaking the DSD Property

We need portions of logic with possibly joint support
Breaking the DSD Property

- Idea: merge depth optimal tree circuits with shared support!

- Each tree is generated by the previous algorithm
  - Functions are non-monotone

- The two trees can be combined with a top binary operator
  - XOR/XNOR are preferable
Benchmark Properties

The circuit obtained has:

- $2^n$ inputs
- $n + 1$ levels
- $2^n - 1$ nodes

The function implemented is:

- Non-monotone
- Not directly DSD decomposable
This is true if the cardinality of the functional support is $2^n$.

More about this in a moment.
Adding More Complexity

- Can we make the circuit more complex?
- Swap inputs of one tree
- Add primary outputs as internal nodes

BAD NEWS: we may decrease the cardinality of the functional support
Functional Support

Simple example:

\[ ab + c + d \]

Functional support
cardinality = 3

\[ a + c + d \]

Functional support
cardinality = 4

\[ ab' + c \wedge d \]

Functional support
cardinality = 4

\[ a + b + c + d \]
Verifying the Functional Support

Solution: Check the functional support of the composite circuit
- BDD techniques
- SAT techniques

```
abc 01> read example.blif; strash; print_supp -s
Total func supps =  4.
Total struct supps =  4.
Sat runs SAT =  0.
Sat runs UNSAT =  0.
Simulation =  0.00 sec
Traversal =  0.00 sec
Fraiging =  0.00 sec
SAT =  0.00 sec
TOTAL =  0.00 sec
abc 03>
```
Final construction algorithm:

**Algorithm 2** Generation of depth-optimal multi-level circuits with joint support.

**INPUT:** Complexity measure $n$

**OUTPUT:** Depth-optimal circuit with $2^n$ inputs.

- $A = \text{Algorithm 1}(n)$;
- $B = \text{Algorithm 1}(n)$;
- share primary inputs of $A$ and $B$;
- create node $n$ that joins roots of $A$ and $B$;
- assign node $n$ to a random binary operator;
- set $n$ as primary output of the composite circuit;
- verify the functional support of the composite circuit;

Hard to synthesize circuits (non-monotone, non-DSD)

At most 1 level far from the optimum
Exact Benchmark Example

- Example circuit with complexity measure = 4
- 16 inputs
- Result strashed into AIG
- The depth optimality is not guaranteed after strashing
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions
Experiments: Initial Circuits

Starting implementation: BDD—collapsing of exact circuits

\[ 2^n \text{ inputs} \]
\[ n+1 \text{ levels} \]
\[ 2^{n+1} - 1 \text{ nodes} \]

\[ 2^n \text{ inputs} \]
\[ 2^n \text{ levels} \]
\[ 2^n < \# \text{ nodes} < 2^{2^{(n-1)}} \]
Experiments: Sample Testing Flow

Mapping onto:
AND (unit cost) OR (unit cost)
XOR (unit cost) INV (free)
Experimental Results

- Use of different delay synthesis techniques
- Starting point: collapsed BDD
- DSD, AIG, MIG, AIG–strashing of the exact circuit

Exponential gap between known optimal results and heuristics
Experimental Results

- Same setup, reporting size values:
- Exponential gap also for size
- Limit for scalability: BDD collapsing
- 1024 inputs already has a 0.6M nodes BDD
- After that, collapsing becomes difficult and BDD-size is out of scale
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions
Conclusions and Future Research

- We presented a method to generate exact-depth multilevel benchmarks
  - Non-trivial functionality: non-monotone, non-DSD
  - Benchmark generation is efficient
    - Circuits with less than $2^{10}$ inputs can be generated in matter of seconds
    - Circuits with more than $2^{10}$ inputs can be generated in matter of minutes
    - Bottleneck is functional support check

- We proved an exponential gap between known optimal results and heuristics
  - More research is still needed in logic synthesis!

- Future research will consider:
  - More control on the implemented logic function in the exact benchmark
    - Possibility to embed specific functions, or function classes, frequently appearing in practical designs
  - More scalable collapsing
    - Hybrid BDD/SAT based collapsing
Questions?

Thank you for your attention!