Inverter Redistribution through Self-Dual and Self-Anti-Dual Function Transformation

Jingren Wang[1] Guangyu Hu[2] Shiju Lin[1] Hongce Zhang[1]

[1]MICS, HKUST(GZ), Guangzhou, China

[2]IIP-MICS, HKUST, Hong Kong, China

May 30, 2025

Outline

1. Motivation
2. Background
3. Proposed Method
4. Experiments
5. Conclusion

The Problem: Responsibility on workflow

Imaging a scenario that Alice work on pure logic synthesis and Bob works on mapping. One day Bob complained to Alice: "Why you have obviously redundant inverters and why you don't have a distribution guideline for them?" Both of them blame each other for "Never asked."

- This was originally a workflow guideline problem rather than an optimization technique.

Key insight:
Give inverter distribution guideline *before* mapping can standardize the workflow and clear the responsibility.

Actual drawback on inverter control due to different heuristic optimization methods.

- Inverter number might silently increase without monitoring.
- Optimizations are independent of each other. They make arbitrary different decisions on inverters in different scenarios.

Key insight:
An "early stopping" is needed also for inverters in such scenario.

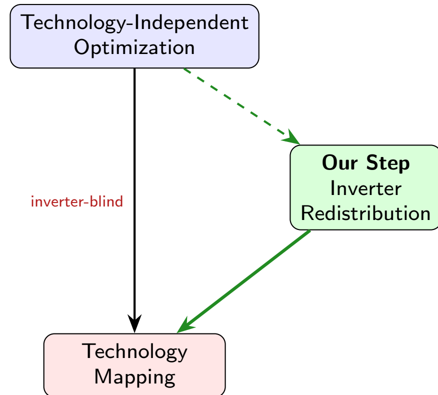
```
(base) → Downloads abc -c "source /Users/jingrenwang/Downloads/Github/abc/
abc.rc; read_truth -xf ./2026_IWLS_Contest_Benchmarks_260121/ex270.truth;ds
d; strash; &get; &st; &dc4; &put; share; strash; &get; &st; &dc4; &put; sha
re; compress2rs; compress2rs; compress2rs; compress2rs; print_stats;"
ABC command line: "source /Users/jingrenwang/Downloads/Github/abc/abc.rc; r
ead_truth -xf ./2026_IWLS_Contest_Benchmarks_260121/ex270.truth;dsd; strash
; &get; &st; &dc4; &put; share; strash; &get; &st; &dc4; &put; share; compr
ess2rs; compress2rs; compress2rs; compress2rs; print_stats;".

Loading resource file "/Users/jingrenwang/Downloads/Github/abc/abc.rc".
ex                              : i/o =     8/   16  lat =    0  and =     417
NegEdges = 428 lev = 10
(base) → Downloads abc -c "source /Users/jingrenwang/Downloads/Github/abc/
abc.rc; read_truth -xf ./2026_IWLS_Contest_Benchmarks_260121/ex270.truth;ds
d; strash; &get; &st; &dc4; &put; share; strash; &get; &st; &dc4; &put; sha
re; compress2rs; compress2rs; compress2rs; compress2rs; compress2rs; print_
stats"
ABC command line: "source /Users/jingrenwang/Downloads/Github/abc/abc.rc; r
ead_truth -xf ./2026_IWLS_Contest_Benchmarks_260121/ex270.truth;dsd; strash
; &get; &st; &dc4; &put; share; strash; &get; &st; &dc4; &put; share; compr
ess2rs; compress2rs; compress2rs; compress2rs; compress2rs; print_stats".

Loading resource file "/Users/jingrenwang/Downloads/Github/abc/abc.rc".
ex                              : i/o =     8/   16  lat =    0  and =     417
NegEdges = 430 lev = 10
(base) → Downloads
```

The Problem: Inverter Blindness in AIG Optimization

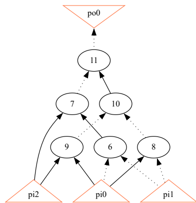
Given facts:

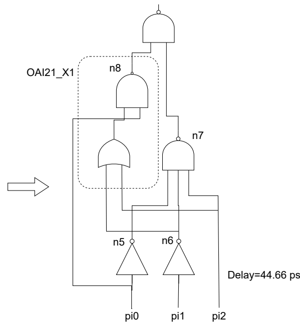
- AIG-based synthesis treats inverters as complemented edge attributes
- Cost functions (level, node count) ignore inverters entirely
- Inverter distribution evolves **randomly** during optimization
- At technology mapping → inverters suddenly matter for delay & area

Key insight:
Can we reshape inverter distribution *before* mapping to standardize the working flow, remove redundancies and improve results?

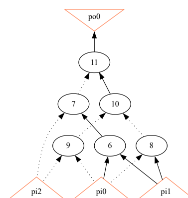


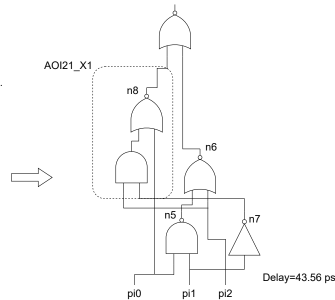
Minimal Reproducible Example





(a)





(b)

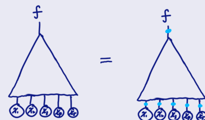
- Both sides map the **same function** from AIG to standard cells; dashed edges are complemented
- Left: baseline; Right: our method identifies a **self-dual subfunction**, complements all cut inputs + root fanout
- Result: fewer complemented edges on the critical path → better delay

Self-Dual and Self-Anti-Dual Functions

Self-Dual

$$f(x_1, \ldots, x_n) = \bar{f}(\bar{x}_1, \ldots, \bar{x}_n)$$

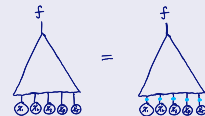
Complementing *all* inputs and the output preserves the function.



Self-Anti-Dual

$$f(x_1, \ldots, x_n) = f(\bar{x}_1, \ldots, \bar{x}_n)$$

Complementing all inputs alone preserves the function.



Self-Dual and Self-Anti-Dual Functions

Self-Dual Example: 3-input Majority

$$\mathrm{maj}(a, b, c) = ab + bc + ac$$

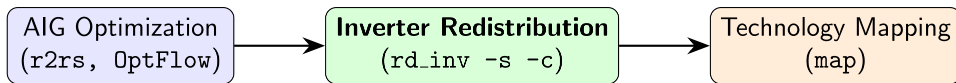
- Complement all inputs:
  $\mathrm{maj}(\bar{a}, \bar{b}, \bar{c}) = \bar{a}\bar{b} + \bar{b}\bar{c} + \bar{a}\bar{c}$
- Negate output (De Morgan):
  $\overline{\bar{a}\bar{b} + \bar{b}\bar{c} + \bar{a}\bar{c}}$
  $= (a + b)(b + c)(a + c) = ab + bc + ac$ ✓

Self-Anti-Dual Example: 2-input XOR

$$a \oplus b = a\bar{b} + \bar{a}b$$

- Complement all inputs:
  $\bar{a} \oplus \bar{b} = \bar{a}b + a\bar{b}$
- Equals original:
  $\bar{a} \oplus \bar{b} = a \oplus b$ ✓

Method Overview

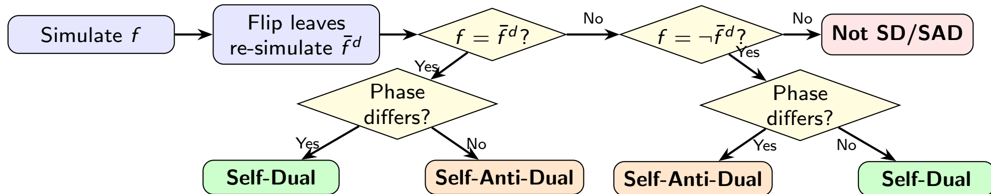


Three-step process per node:

1. **Detect** — simulate truth tables to identify self-dual / self-anti-dual subfunctions on tree cuts
2. **Evaluate** — apply policy to decide if redistribution is beneficial
3. **Transform** — complement cut leaves (and root fanout for self-dual)

Key property: The AND nodes in the original AIG are never modified.

Detection Algorithm



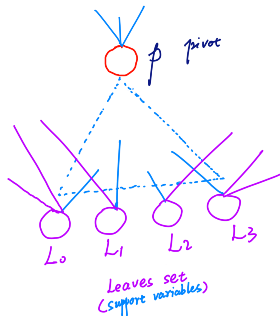
- $f = \bar{f}^d$ **(truthEq)**: phase differs → Self-Dual; same phase → Self-Anti-Dual
- $f = \neg\bar{f}^d$ **(isComp)**: phase matches → Self-Dual; phase differs → Self-Anti-Dual
- Masking vector `uMask` handles cuts with fewer than 5 leaves

Self-Anti-Dual transformation:

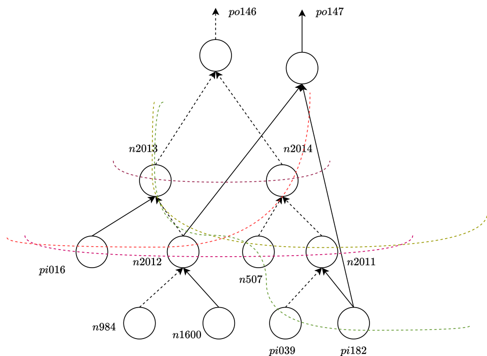
- Complement edges at **cut leaves** (only those feeding internal nodes)

Self-Dual transformation:

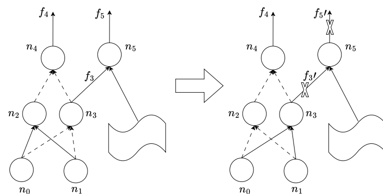
- Same as self-anti-dual, **plus** complement **fanout edges** of the root node



Tree Cuts vs. DAG Cuts



5 tree cuts (dashed) of node *po*146



DAG cut leads to equivalence failure

Tree cuts are safe and independent. DAG cuts are prohibited.

Node and Cut Selection

Which nodes to process:

1. DAG nodes and tree nodes on the **primary graph** (no choices)
2. **Choice candidates** in the choices network

- Choice representatives are **skipped** — the primary structure is already near-optimal

Which cuts to use:

- Only **tree cuts** — guaranteed inside MFFC (Maximum Fanout Free Cone)
- DAG cuts are excluded due to equivalence risk(Oppertunity?)

Redistribution Policy

Critical Path

- Redistribute only if #complemented edges > #regular edges on critical path
- $\Rightarrow$ **Net reduction** of inverters on critical paths
- Targets delay improvement

Non-Critical Path

- Redistribute if inverter count $\geq$ half of all edges
- Allows **zero-gain** redistribution
- Creates favorable conditions for mapping

Zero-Gain Redistribution

- Tree-structured XORs (self-anti-dual) with inverter ratio $= 0.5$
- No local improvement, but **enables inverter absorption** during mapping, e.g., XNOR2 (44.4 ps) over XOR2 (53.5 ps)
- Cumulative delay improvement: **3–4%** on arithmetic circuits

Experimental Setup

- **Platform:** Apple M4 Pro, 24 GB, macOS Tahoe 26.2
- **Framework:** ABC — entirely on AIG representations
- **Verification:** CEC (combinational equivalence checking) after redistribution
- **Benchmark:** EPFL combinational suite (random control + arithmetic)

Synthesis flows compared:

OptFlow `st; &get; &if -g; &dch; &put`

Baseline (`r2rs`)^8 (OptFlow)^8 `map; topo; stime`

Ours (`r2rs`)^8 (OptFlow)^8 `rd_inv -s -c` `map; topo; stime`

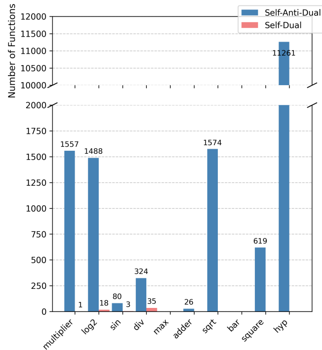
Key: Only a *single* round of `rd_inv` needed — AND nodes unchanged, level slack preserved.

Results: Arithmetic Logic

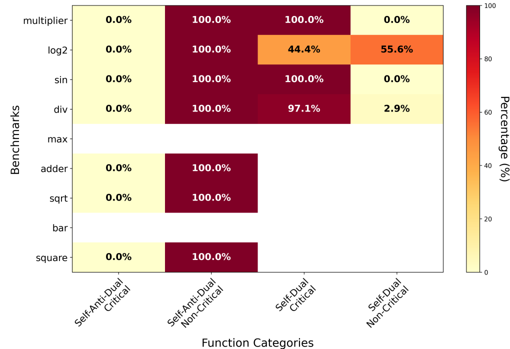
Benchmark	Delay Diff	% Improvement	Gate Diff	INV_X1 Diff
sqrt	−5289.82 ps	**3.86%**	+429	−429
square	−17.69 ps	**1.60%**	−29	−15
hyp[16]	−10437.94 ps	**1.46%**	−361	−1959
voter	−13.93 ps	**1.09%**	−22	−10
multiplier	−26.84 ps	**0.77%**	−175	−20
log2	−73.87 ps	**0.85%**	−101	+17
div	−61.47 ps	**0.10%**	−86	+8
sin	−1.55 ps	**0.04%**	−4	+34

- Random control logic: **no change** — inverter distribution unaffected
- **No degradation on any benchmark**
- Arithmetic circuits are **highly sensitive** — structured XOR chains and self-dual subfunctions
- *sqrt*: 3.86% delay reduction with only +29 area cost

Analysis: Self-Dual vs. Self-Anti-Dual Distribution



SD/SAD function count



Critical vs. non-critical

- **Self-anti-dual** (tree XORs) vastly outnumbers self-dual — dominant contributor
- Self-dual: found exclusively on critical paths → direct delay reduction
- Self-anti-dual: found exclusively on **non-critical paths** → enables inverter absorption

Conclusion

Summary

- Proposed a **delay-oriented inverter redistribution** preprocessing step using self-dual and self-anti-dual function properties
- Operates on AIG tree cuts — **preserves original structures of AND nodes**
- Equivalence guaranteed by tree-cut isolation and compositional closure

Key Results

- **No degradation** on any benchmark
- Arithmetic benchmarks: up to **3.86% delay reduction** (*sqrt*)
- Average delay reduction: **0.49%**
- Zero-gain redistribution on non-critical paths enables inverter absorption during mapping

Thank You

Jingren Wang

jwang929@connect.hkust-gz.edu.cn