To Box or Not to Box: Preserving Special Logic Blocks in Technology-Independent Logic Optimization

Siang-Yun Lee
Cadence Design Systems
Munich, Germany

Heinz Riener
Cadence Design Systems
Munich, Germany

Abstract

The modern logic synthesis process is usually separated into phases of technology-independent optimization, technology mapping, and post-mapping optimization. During technology mapping, besides simple cells composed of only one or two gates in the technology-independent representation, more complex cells such as half adders and full adders are often also available in the standard cell library. Being able to utilize such complex cells more helps improve the final PPA results. However, we observe that technology-independent optimization tends to over-optimize towards the cost metrics measurable with technology-independent representations (e.g., AIG size) by decomposing blocks of logic that could have been mapped into complex cells. Thus, in this paper, we study the possibilities of "boxing" blocks of logic before technology-independent optimization and preserving them during optimization. Our experiments show that transparent-boxing, i.e., preserving boxes while utilizing the information on their logic, achieves better results compared to black-boxing, i.e., treating boxes as black boxes and ignoring their logic completely.

Keywords: Logic synthesis, combinational circuits, Boolean optimization

1 Introduction

Due to scalability concerns, modern logic synthesis flows are typically separated into technology-independent optimization, which focuses on removing logical redundancies using a homogeneous network data structure like the *AND-Inverter Graphs* (AIGs) [6], and technology mapping [4], which converts the optimized *subject graphs* (e.g. AIGs) into mapped netlists using a technology-specific standard cell library. In technology mapping, the subject graph is *covered* by blocks of logic available in the library by structural and logical analysis. When multiple possibilities exist, choices are made to optimize towards cost metrics measurable using information provided in the standard cell library, such as cell area, power, and delay. In contrast, in technology-independent optimization, choices are made to minimize rather simple and approximated cost metrics, such as the number of AIG nodes and the AIG depth.

While the contents of different standard cell libraries differ, some special, commonly used cells are often available, such as half adders and full adders. A *half adder* (HA) is a two-input, two-output cell computing the following functions:

$$\mathrm{HA}_c(x, y) = x \wedge y$$
$$\mathrm{HA}_s(x, y) = x \oplus y.$$

A *full adder* (FA) is a three-input, two-output cell computing the following functions:

$$\mathrm{FA}_c(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z)$$
$$\mathrm{FA}_s(x, y, z) = x \oplus y \oplus z.$$

Empirical observation has been found that correctly utilizing adder cells, instead of composing the same functions using elementary logic gates, may improve the *power, performance, and area* (PPA) results of the final chip [3]. Thus, it is helpful to preserve these logic blocks during optimization. However, without intentionally blocking and protecting them, classic technology-independent optimization algorithms have a high chance of destroying them structurally, making it difficult to re-discover them during technology mapping.

Currently, this problem is tackled in some open-source as well as industrial tools by *black boxing* special logic blocks like adders. When an adder is black-boxed, AIG nodes representing the adder function are taken away, creating an empty hole in the network whose interfaces are connected to additional virtual PIs and POs to keep the graph acyclic and non-dangling. While the black-boxing strategy successfully protects the adders, we argue that a *transparent-boxing* strategy that does not neglect the logical information within boxes would be more advantageous. In a transparent-boxing strategy, boxed nodes are kept in the network and are utilized to, for example, compute don't care conditions for other nodes.

In this paper, we discuss the implementation, limitations, and advantages of different boxing techniques, including black boxing and transparent boxing. Our experiment on technology-independent evaluation shows that transparent boxing achieves the most reduction in the number of free ANDs as well as in the estimated mapped network size, compared to black boxing and no boxing.

2 Preliminaries

2.1 Logic Networks

Logic networks are technology-independent representations heavily used in logic synthesis. A *network* is a *directed acyclic graph* (DAG) where nodes model simple logic gates and edges

model wires potentially with integrated inverters (fanin negations). The *primary inputs* (PIs) and the *primary outputs* (POs) are the interfaces between a network and the outside environment. A representative example of logic networks is the *AND-Inverter Graph* (AIG) where each node is a two-input AND gate. In the figures in this paper, wires with inverters are drawn as dashed edges.

In a typical AIG data structure, a list of nodes is stored, where each node stores pointers to its two fanin nodes, together with fanin negation tags. PIs are represented by special nodes, identified by having identical fanins. POs are stored separately as a list of pointers to nodes with negation tags. During AIG construction and throughout optimization, trivial cases are checked and simplified such that the following conditions should never happen: (1) a node has a constant (0 or 1) fanin (2) a non-PI node's two fanins point to the same node (with or without the same fanin negation tag). Whenever these cases appear, the node should not be created and should be replaced with either a constant or its other fanin. Such replacement and simplification may propagate towards the transitive fanouts.

Moreover, a technique called *structural hashing* is often adopted to avoid redundancies. A hash table is used to quickly look up a node using its fanins (negation tag considered) and to ensure that there is not a second node with exactly the same fanins, i.e., a node is unique to its fanins. During network modification (e.g., node substitution), some nodes' fanins may change and they should be re-hashed, potentially becoming identical to another existing node and are thus merged. Such changes may propagate towards the transitive fanouts.

2.2 Simulation-Guided Resubstitution

Boolean resubstitution is a technology-independent logic optimization algorithm that attempts to *resynthesize* the *maximum fanout-free cone* of a *root* node using existing *divisor* nodes having common support as the root. In cut-based resubstitution, divisors are collected within a smaller window supported by a cut of size 8-10 (typically no larger than 16), and the resynthesis is performed with (complete) truth tables exhaustively simulated from the cut. In contrast, in simulation-guided resubstitution [5], partial truth tables computed from the primary inputs using a non-exhaustive set of simulation patterns are used to approximate nodes' global functions. One of the advantages of simulation-guided resubstitution is that global satisfiability don't cares are considered implicitly with global simulation without needing extra effort to compute them.

2.3 Black-box and Transparent-box Testing

In the field of software testing, the term *black-box testing* refers to a testing strategy that examines the functional correctness of software according to its input-output relationship without knowledge of its internal workings. In contrast, *transparent-box testing* (or *white-box testing*, *clear-box testing*, *glass-box testing*) tests the internal structures of software.

In the context of logic synthesis, we use the term *boxing* to mention the extraction of blocks of logic in a technology-independent representation that can be mapped into (complex, potentially multi-output) cells. We borrow the term *black-boxing* to mention the strategy where the boxed logic is taken away with a "hole" of some dangling wires remaining in the network. In contrast, we use the term *transparent-boxing* for the strategy where the logic function and the internal implementation of the boxes are known and can be utilized during optimization.

2.4 Adder Identification

In [7], algorithms to reverse engineer arithmetic circuits, i.e., identifying adders in an AIG, are presented. To identify elementary adders (HA and FA), the proposed method first enumerates all 3-cuts in the AIG, computes their truth tables, hashes the cuts, and finally finds pairs of identical cuts with different roots whose truth tables are NPN-equivalent to the FA function.

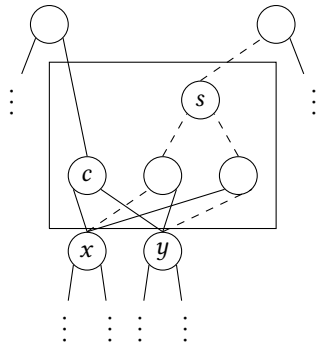
3 Boxing Technology-Independent Logic Networks

In order to preserve specific logic blocks in the network during logic optimization, we propose to create boxes to protect them. A *box* in a network is a connected subset of nodes. Two boxes in the same network shall not overlap. A box can be identified by a set of *box inputs*, which are nodes outside of the box having a fanout inside the box, and a set of *box outputs*, which are nodes inside of the box with at least one fanout outside of the box. Optionally, a label referring to the logic function(s) computed by the box outputs in terms of the box inputs may be associated with the box. For example, Figure 1 shows a half adder box in a network. In this section, we discuss how black and transparent boxes can be implemented in an AIG-like data structure.

3.1 Black-Boxing

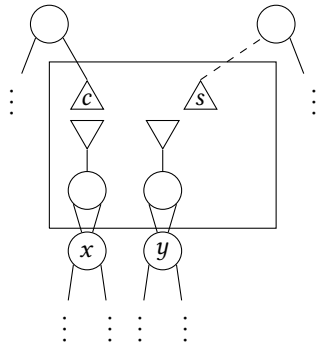
In a black-boxing strategy, nodes within the box are taken away from the network, and the box's inputs and outputs become dangling. To keep the network a valid DAG, additional virtual PIs are added and used to replace the box outputs, and additional virtual POs are added and connected to the box inputs. Practically, we insert additional buffers between the box inputs and the added virtual POs to easily identify these POs as box inputs, in case the box inputs are also connected to a real PO of the network. Figure 2 illustrates such a black-boxing strategy.

Classic logic optimization algorithms can be applied directly on a black-boxed network without incurring algorithmic or correctness errors (e.g., the optimized network becomes cyclic or functionally non-equivalent to the original



Box inputs = $\{x, y\}$
Box outputs = $\{c, \neg s\}$

Figure 1. A boxed half adder in a network.



Box inputs = $\{x, y\}$
Box outputs = $\{c, \neg s\}$

Figure 2. Data structure implementation of a black box in a network.

network). This is because (1) the box outputs are replaced with independent PIs, so no assumptions on their values in relation to other nodes in the network would be made, and (2) the box inputs are connected to additional POs, so their functions would be kept exactly the same under all circumstances.

3.2 Transparent-Boxing

A transparent-boxing strategy keeps track of, in addition to the interfacing box inputs and outputs, the logic function computed by the box, and even the internal implementation of the box as well. There are several ways of storing the box function. For example, storing truth tables, using binary decision diagrams (BDDs), or linking to separated sub-networks. To minimize the impact on the network data structure and to facilitate adaptation of existing algorithms to boxed networks, we propose to simply keep box nodes in the network but mark them as *don't touch*es.

A don't touch is an additional flag stored for each node in the network. Network modification methods, such as node substitution, are modified to avoid changing nodes marked as don't touch. Special attention should be paid to handling trivial cases, structural hashing, and their propagation towards transitive fanouts. Specifically, (1) constants and repeated fanins can be allowed at box inputs; decisions on whether to simplify the box can be made in a post-optimization process. (2) If a node in a box is (or becomes) structurally identical to a node outside of the box, both of them should be kept.

The advantage of this transparent-boxing strategy is that the network data structure remains the same as its non-boxed counterpart, so most non-modifying computations, such as simulation, cut enumeration, and structural analysis, can be done in exactly the same way. An illustration of a white-boxed data structure would look the same as in Figure 1, except that nodes in the box are tagged as don't touches.

4 Logic Optimization with Boxes

Using a black-boxing strategy, special logic blocks are easily preserved during technology-independent logic optimization. This strategy has the advantage of being straightforward and guarantees to work with any optimization algorithm without adaptation. However, the logical relations between the box inputs and outputs are completely neglected during optimization, which may lead to oversight of potential optimization opportunities. Moreover, by connecting box inputs and outputs to virtual POs and PIs, the overall structure (or the "shape") of the network is changed. For example, a box may be originally "deep" in the network, i.e., having a large logic depth from PIs, but the depth of the box outputs will become very small after black-boxing because they are connected to virtual PIs. As another example, due to the "holes" and disconnections caused by black-boxing, some cuts may disappear. These structural changes may have an impact on the quality of the results of some optimization algorithms.

To avoid the disadvantages of black boxing, we propose to keep the structural and logical information with transparent boxing. Using the proposed transparent-boxing strategy, most optimization algorithms should perform exactly the same as without boxing when optimizing non-boxed nodes. However, some optimization algorithms may benefit more from the information provided by the transparent boxes because they leverage don't-care computations in larger windows beyond the MFFC of the node being optimized. One prominent example of such algorithms is simulation-guided resubstitution [5], which relies on whole-circuit simulation to approximate nodes' global functions, where global satisfiability don't cares are implicitly considered. Another example is the new rewriting framework capable of using don't cares [10].

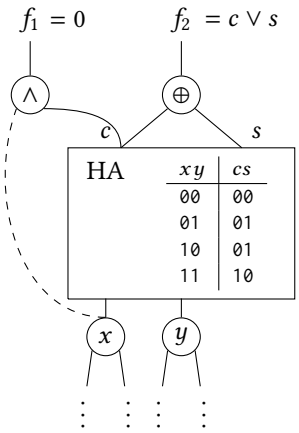


Figure 3. Example of optimization with boxes.

Figure 3 shows two examples of optimizations that are only discoverable with transparent boxing. First, the output function f_1 depends on a box output and a box input, which are related. With a black box in between, optimization algorithms cannot recognize that f_1 always evaluates to constant 0. In contrast, with a transparent box, an algorithm capable of computing don't cares would realize the satisfiability don't care condition $x = 0, c = 1$ and use this fact to substitute the AND gate with constant 0. Second, the output function f_2 depends on two outputs of the same box, which are also related. Again, with a black box model, the two box outputs are unrelated virtual PIs and cannot be optimized. However, with a transparent box, the don't care condition $c = 1, s = 1$ can be computed and used to optimize the XOR gate to an OR gate, which may have a lower cost.

5 Experimental Results

The boxing techniques discussed in this paper are implemented in the C++ logic synthesis library `mockturtle`[1] [8, 9] for experimental evaluation.

Although the proposed boxing techniques as well as the implementation are generic to what functions the boxes compute, in our experiments, we consider only the most commonly met type of complex cells: adders, including half adders and full adders.

5.1 Technology-Independent Evaluation

We use the EPFL benchmark suite [1] for the technology-independent evaluation of the boxing techniques discussed in this paper. The benchmarks are pre-processed with the balancing command `&b` in ABC [2] as a heuristic to increase the number of recognizable adders. Before optimization, half adders and full adders are identified and boxed by enumerating and analyzing the functions of 3-cuts [7]. In Table 1, statistics are reported for benchmarks before optimization, optimized normally without boxing, optimized with black-boxing, and optimized with transparent-boxing. In all cases, the optimization algorithm is simulation-guided resubstitution with a cut size (for divisor collection) of 8 and a maximum dependency circuit size of 20. Columns "A" show the number of free AIG nodes (nodes not in any boxes) and columns "A+B" show the number of free AIG nodes plus the number of boxes, which resembles the size of the mapped network using a standard cell library consisting of only AND2, half adder, and full adder. Columns "Time" show the optimization time in seconds.

It is observed that optimization with transparent-boxing achieves the most reduction in both the number of free ANDs and the mapped network size.

[1]Available: https://github.com/lsils/mockturtle

6 Conclusions

In this paper, we discuss methods to "box" special logic blocks in technology-independent optimization to preserve them. We argue that some don't-care-based optimization algorithms would take advantage from the information of the boxed logic, thus we propose the transparent-boxing technique to enhance optimization quality with boxes. We discuss potential problems and details to pay attention to in the implementation of the data structure. Finally, our preliminary experimental evaluation show that transparent boxing achieves the best results. It is our future work to evaluate different boxing strategies on industrial flows and benchmarks.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of IWLS*.

[2] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification, 22nd International Conference, CAV 2010*. 24–40.

[3] Alessandro Tempia Calvino and Giovanni De Micheli. 2023. Technology Mapping Using Multi-Output Library Cells. In *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 1–9.

[4] Kurt Keutzer. 1987. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference. Miami Beach, FL, USA, June 28 - July 1, 1987*. 341–347.

[5] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K. Brayton, and Giovanni De Micheli. 2022. A Simulation-Guided Paradigm for Logic Synthesis and Verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41, 8 (2022), 2573–2586.

[6] Alan Mishchenko and Robert Brayton. 2006. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*, Vol. 6. 15–22.

[7] Alan Mishchenko, Baruch Sterin, and Robert Brayton. [n.d.]. Structural Reverse Engineering of Arithmetic Circuits. ([n. d.]).

[8] Heinz Riener, Eleonora Testa, Winston Haaswijk, Alan Mishchenko, Luca G. Amarù, Giovanni De Micheli, and Mathias Soeken. 2019. Scalable Generic Logic Synthesis: One Approach to Rule Them All. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 70.

[9] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee,

Table 1. Evaluation of different boxing techniques on EPFL benchmarks.

	Before optimization		Without boxing		Black-boxing			Transparent-boxing		
Bench.	A	A+B	A+B	Time	A	A+B	Time	A	A+B	Time
adder	0	128	128	0.0	0	128	0.0	0	128	0.0
bar	3214	3214	3155	0.0	3155	3155	0.0	3155	3155	0.0
div	13784	17695	17615	2.7	13580	17436	0.4	13575	17518	1.8
hyp	113550	133666	114507	98.1	109682	122860	2.9	108642	113954	63.6
log2	20527	21981	21483	16.8	20451	21915	0.4	19822	21300	2.5
max	2865	2865	2862	0.0	2863	2863	0.0	2863	2863	0.0
multiplier	15384	16895	16826	2.1	15381	16890	0.5	15315	16824	0.2
sin	3554	3820	3743	1.1	3520	3787	0.1	3447	3728	1.5
sqrt	14734	16553	16467	5.2	14615	16438	0.1	14595	14602	2.7
square	7949	9327	8175	0.7	7238	8388	0.5	7067	8169	0.6
arbiter	11839	11839	11839	0.3	11603	11603	0.4	11603	11603	0.4
cavlc	672	679	616	0.2	633	643	0.3	598	614	0.2
ctrl	165	166	93	0.0	112	114	0.0	100	103	0.0
dec	304	304	304	0.0	304	304	0.0	304	304	0.0
i2c	1187	1191	1084	0.0	1105	1110	0.1	1089	1094	0.1
int2float	226	227	212	0.1	209	210	0.2	212	214	0.2
mem_ctrl	46543	46584	38083	2.7	40113	40150	4.3	38298	38353	4.2
priority	843	843	518	0.0	560	560	0.0	560	560	0.0
router	133	171	133	0.0	133	171	0.0	119	164	0.0
voter	1231	2558	2037	1.0	1231	2544	0.0	528	1493	0.6
Total	258704	290706	259880 (-10.6%)	131.0	246488 (-4.7%)	271269 (-6.7%)	10.2	241892 (-6.5%)	256743 (-11.7%)	78.7

Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, and Giovanni De Micheli. 2022. The EPFL Logic Synthesis Libraries. arXiv:1805.05121 http://arxiv.org/abs/1805.05121

[10] Alessandro Tempia Calvino and Giovanni De Micheli. 2024. Scalable Logic Rewriting Using Don't Cares. In *Proceedings of DATE*.