SAT-Based Area Recovery in Structural Technology Mapping

Bruno Schmitt

Ecole Polytechnique Federale de Lausanne (EPFL)
bruno.schmitt@epfl.ch

Alan Mishchenko Robert Brayton

Department of EECS, UC Berkley
{alanmi, brayton}@berkeley.edu

Abstract— This paper proposes a fast SAT-based algorithm for recovering area applicable to an already technology mapped circuit. The algorithm considers a sequence of relatively small overlapping regions, called windows, in a mapped network and tries to improve the current mapping of each window using a SAT solver. Delay constraints are considered by interfacing the SAT solver with a timer. Experimental results are given for benchmarks that have been mapped already into 6-LUTs by a high-effort area-only synthesis/mapping flow. The new mapper starting from these results, many of which represented the best known area results at the time, achieved an additional average area reduction of 3-4%, while for some benchmarks the area reduction exceeded 10%. Runtime for any example was only a few seconds.

I. Introduction

Technology mapping expresses a Boolean network representing the functionality of a hardware design as a set of primitives from a technology library. In mapping into FPGAs based on lookup tables (LUT), the library is composed of K-input LUTs, where $K = 4$ or $K = 6$ for most commercial FPGAs.

LUT-based mappers measure area in terms of LUT count, and delay in terms of LUT level, defined as the largest number of LUTs on any path from primary inputs to primary outputs. Reducing area and delay of a LUT mapping is an important goal achieved using heuristic structural mappers, such as [11], followed by post-mapping Boolean re-synthesis, such as [12].

Previous SAT-based Boolean matching methods for technology mapping, such as [7][15], are mainly functional; that is, given a Boolean function, they synthesize a LUT structure. In contrast, this paper proposes a *structural* SAT-based method for reducing area of a given LUT mapping. The algorithm uses a SAT solver to find minimum-area K-LUT covers of a sequence of small multi-input and multi-output regions (windows) in an initial K-LUT mapping. Because the optimization performed is purely structural (it does not exploit the Boolean nature of the network nor does it use don't-cares [12][13]) it has fast runtime, yet often results in improved area and delay.

The method can optimize area under delay-constraints, but instead of encoding delay constraints in the conjuctive normal form (CNF), as in [9], the SAT solver is interfaced with a dedicated timer capable of determining if a mapping, found by the solver, has acceptable delay. A similar approach based on decoupling of a SAT-based enumeration and an application-specific evaluation has been used in [8].

Experiments show that the proposed engine gives sizeable improvements for circuits already mapped by a high-effort area-oriented synthesis and mapping flow. Although these improvements are counterintuitive, they may indicate that area-recovery heuristics used in mainstream mappers do not perform well for deep And-Inverter Graphs (AIG) with a homogenous logic structure. It can be shown that in such AIGs, the area-recovery heuristics used in state-of-the-art technology mappers often make incorrect decisions when choosing one local mapping among many candidates, due to the lack of clear winner among them. These mistakes accumulate during each run of the mapper, resulting in a substantial area penalty unless the area is further recovered by post-processing using methods similar to the one proposed in this paper.

The rest of the paper is organized as follows. Section II contains relevant background. Section III shows the high-level view of the proposed SAT-based engine and describes its components. Section IV contains experimental results, and Section V concludes the paper.

II. Background

A. Boolean Functions

Boolean variable x is a variable that takes one of the two values from the domain $\mathbb{B} = \{false, true\}$, or $\{0, 1\}$. A **positive literal** is the Boolean variable x and a **negative literal** is its complement $\overline{x}$. In this paper, **function** refers to a **completely specified Boolean function** $f(X) : \mathbb{B}^n \rightarrow \mathbb{B}$ of n variables $X = \{x_1, x_2, \ldots, x_n\}$. The **support** of f is the subset of variables that influence the output value of the function f. The support size is denoted by $|X|$.

B. Boolean Networks

A **Boolean network** (or circuit) is a directed acyclic graph (DAG) $G = (V, E)$ with nodes V and edges E. Every node is associated with a Boolean function and a Boolean variable, called the output variable, representing the node's output. The existence of an outgoing edge from node n_1 to node node n_2 means that the variable representing the output of n_1 is an input to the function represented by n_2. In this case, we say that n_1 is a **fan-in** of n_2, or that n_2 is a **fan-out** of n_1.

A node n might have zero or more fan-ins and zero or more fan-outs. **Primary inputs** are nodes without fan-ins. **Primary outputs** are a subset of nodes that connect the networks to the environment. A **transitive fan-in** (**fan-out**) **cone** (TFI/TFO) of a node is a subset of nodes of the network, that are reachable through the fan-in (fan-out) edges of the node. The **TFO support** of a node is the set of primary inputs reachable through the fan-outs of the node.

978-1-5090-0602-1/18/$31.00 ©2018 IEEE

C. And-Inverter Graph (AIG)

An **And-Inverter Graph** (AIG) is a combinational Boolean network composed of two-input AND gates and inverters. An AIG for a Boolean network can be derived by factoring the functions of the logic nodes found in the network. The AND/OR gates in the factored forms are converted into two-input ANDs and inverters using De Morgan's law and added to the AIG in a topological order.

A **cut** C of a node n is a subset of nodes, called leaves of the cut, such that each path from a primary input to n passes through at least one leaf. Node n is called the **root** of cut C. The cut size is the number of its leaves. A trivial cut of a node is the cut composed of the node itself. A cut is K**-feasible** if the number of nodes in the cut does not exceed K. Cut enumeration is used by a cut-based technology mapper, such as [11], to compute cuts using dynamic programming, starting from primary inputs and ending at primary outputs.

D. LUT Mapping

A K**-input lookup table (**K**-LUT)** is a hardware device, which can implement any Boolean function up to K inputs. ABoolean network can be **mapped** into K-LUTs by a software package called **technology mapper**. The mapper takes a **subject graph**, which is a technology-independent representation of the network, such as an AIG, and returns a **mapping**, which is a set of LUTs covering the subject graph. Each internal node of the subject graph is either used in the mapping (if the mapping includes a LUT rooted in this node) or not used in the mapping (otherwise). A mapping is **valid** if the internal nodes driving the primary outputs of the network are used in the mapping and, for each LUT of the mapping, its fan-ins are used in the mapping, or are primary inputs.

In this work, we assume that the Boolean network is already mapped using K-LUTs and the resulting mapping is valid. Furthermore, we do not consider any technology-dependent information associated with the LUTs, except their connectivity.

E. Boolean Satisfiability

A **satisfiability problem** (SAT) takes a propositional formula representing a Boolean function and decides if the formula is satisfiable or not. The formula is **satisfiable** (SAT) if there is an assignment of variables that evaluates the formula to 1. Otherwise, the formula is **unsatisfiable** (UNSAT). A software program that solves SAT problems is called a **SAT solver**. SAT solvers provide a satisfying assignment when the problem is satisfiable.

Modern SAT solvers can accept a set of **assumption**, each of which enforces a value to a variable. The process of determining the satisfiability of a problem under given assumptions, is called **incremental SAT solving** [4].

F. Conjunctive Normal Form (CNF)

To represent a propositional formula in the SAT solver, important aspects of the problem are encoded using Boolean variables. The presence or absence of a given aspect is represented by a positive or negative literal of a variable. A disjunction of literals is called a **clause**. A conjunction of clauses is called a **conjunctive normal form** (CNF). CNFs are processed by CNF-based solvers, such as MiniSAT [4].

CNF is composed of clauses encoding different aspects of the SAT problem. For example, some CNF clauses may encode covering constraints, that is, requirements for each node that, if a node is mapped, its fan-ins are mapped. Another important constraint type present in many SAT problems, is introduced in the next section.

G. Cardinality Constraints

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a fixed finite set of Boolean variables of a propositional formula represented in CNF. A **cardinality constraint** states that, among the n Boolean variables in X, m or less have value 1. Meaning that a satisfiying assingment for this propositional formula will have at most m true variables.

A naive way to represent this constraint is to generate $\frac{n!}{m!(n-m)!} + \frac{n!}{(m-1)!(n-m+1)!} + \ldots + 1$ clauses, each ruling out a specific subset of variables, of size m or less, that have value 1. For values appearing in practice, this can be prohibitive, e.g. $(n, m) = (64, 16)$ leads to more than $4.8 * 10^{14}$ clauses.

Efficient CNF representation has been a topic of active research for more than a decade. It has been shown [5] that pseudo-Boolean constraints (a generalization of cardinality constraints) can be expressed efficiently by using **sorting neworks**. The resulting CNFs lead to faster SAT because sorting networks contain only ANDs and ORs. Other proposed representations of cardinality use adders, boolean decision diagrams (BDDs), etc, contain XORs and MUXes.

Moreover, it has been shown [1] that exactly one half of the clauses used to encode a cardinality constraint represented as a sorting network do not have to be added to the SAT solver. This new representation is called a **cardinality network**. For a comprehensive background on this, we refer the reader to [1].

Finally, [3] shows that among the two equal-cost implementations of sorting networks, the one known as the pair-wise sorting network [14] has better implicativity (generates more implications) and therefore leads to substantially faster SAT solving. Hence, our representation of cardinality constraints uses cardinality networks derived from pair-wise sorting networks.

III. SAT Based Engine

This section describes the components of the proposed SAT-based structural re-mapping engine shown in Figure 1. The input is an AIG mapped into K-input LUTs. The output is the same AIG but with a different K-input LUT mapping assigned. The new mapping is expected to have the same or better area and delay, compared to the original mapping.

LUTs of the current mapping are considered in a topological order, although a different order could be used. For each LUT, a window is computed (Subsection A). The window contains the given LUT together with other LUTs in its TFI and TFO. The complete set of structural K-LUT covers of the window is represented in CNF (Subsection B). The SAT solver takes this CNF and looks for an improved mapping (Subsection C).

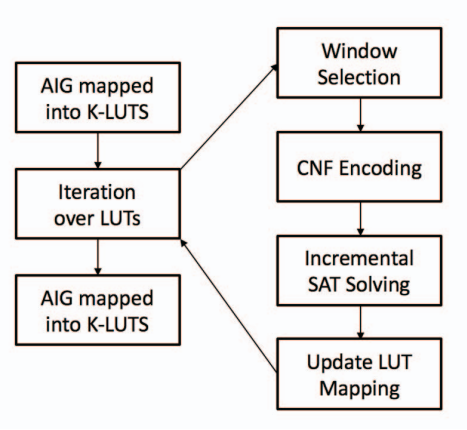

Fig. 1. Overview of the proposed engine

If found, the current mapping is updated (Subsection D). Delay constraints are handled as shown in Subsection E.

When computation for a given LUT is finished, the mapper moves to the next LUT in the order. When all LUTs have been considered or when a resource limit, such as a global timeout, has been reached, the mapper outputs an improved LUT mapping.

A. Window Selection

A window is a small region of the subject graph. The size of a window is the number of subject graph nodes it contains. A window, computed for a LUT-mapped AIG, is represented as a combination of LUT root nodes and one of theirs structural cuts. Initially, the window contains only one LUT. The windowing algorithm extends the window by including adjacent LUTs, that is, LUTs that are fan-ins or fan-outs of the LUT(s) currently included in the window. When deciding which neighboring LUTs should be added, priority is given to the LUT that increases the size of the window least, because such windows have a higher optimization potential. Figure 2 shows the window selection process for a simple mapped AIG.

We do not limit the number of LUTs contained in the window but limit the number of AIG nodes. This is because each AIG node adds one SAT solver variable and one input to the cardinality constraint. Therefore, controlling the number of AIG nodes in the window is important for ensuring the scalability of the method.

Although our current implementation scales up to 128 AIG nodes, it was found experimentally that a good tradeoff between runtime and quality is achieved for windows containing up to 32 nodes. Smaller windows often do not lead to much improvement, while larger windows are better but are more likely to timeout.

A computed window is represented as an ordered set of internal AIG nodes included in the window. The window leaves (the nodes not included, having at least one fan-out that is included) and window roots (the nodes included, having at least one fan-out that is not included) are computed easily. These two sets are used for CNF construction described in the next subsection.

This algorithm, when applied to different nodes, can result in identical windows. To avoid this, window selection employs a hash table, which caches windows that have been tried and did not lead to improvement.

B. CNF encoding

The CNF used to present the set of all possible structural mappings of the window contains the following variables:

- Variable $n_i = 1$ is used to represent that AIG-node is used in the mapping.
- Variable $c_k^i = 1$ is used to represent that cut k of internal AIG node i is used in the mapping.

The resulting CNF contains four types of constraints:

- If node i is used, one of its cuts is used: $n_i \rightarrow \bigvee_k c_k^i$.
- If cut k is used, all cut leaves are used: $c_k^i \rightarrow \bigwedge_l n_l$.
- The nodes driving the outputs are used: $\bigwedge_o n_o$.
- The cardinality constraint $\sum_i n_i < m$ holds, where m is the LUT count of the current mapping in the window.

The window leaves do not have to be represented by SAT variables because they are always used in the mapping. Thus, if a cut leaf is a window leaf, its SAT variable is assumed to have value 1. Similarly, the window roots do not have to be represented by SAT variables because they are always used in the mapping. In this case, the third constraint ($\bigwedge_o n_o$) is omitted and the respective n_i variables in other constraints can be assumed to have value 1.

The CNF generated for a typical window with 32 AIG nodes contains roughly a thousand CNF clauses and about half of them are due to the cardinality constraint.

C. Incremental SAT solving

The computed CNF is loaded into the SAT solver. The cardinality constraint, m, is set to a value that is smaller by one than the number of LUTs in the window. If such solution is found, the cardinality constraint is tightened again. A sequence of incremental SAT calls continues until the solver returns UNSAT, or until a resource limit is reached. The last feasible solution whose quality is strictly better than that of the initial mapping is used to update the current mapping of the window.

D. Update LUT Mapping

To enable efficient window selection and updating of the LUT mapping, the internal nodes of the AIG representing the subject graph are annotated with their status in the current LUT mapping. For this, each AIG node used in the mapping is put in correspondence with a K-feasible cut used to represent this cut in the current mapping.

When a mapping of the window is updated by the SAT-based mapper, only the old annotation of the internal nodes of the window need be invalidated and the new annotation is created to reflect the update. As a result, the mapping of the subject graph is valid after each update. Thus, if the computation exceeds a resource limit, the current mapping can be returned, resulting in a valid mapping of the original AIG.

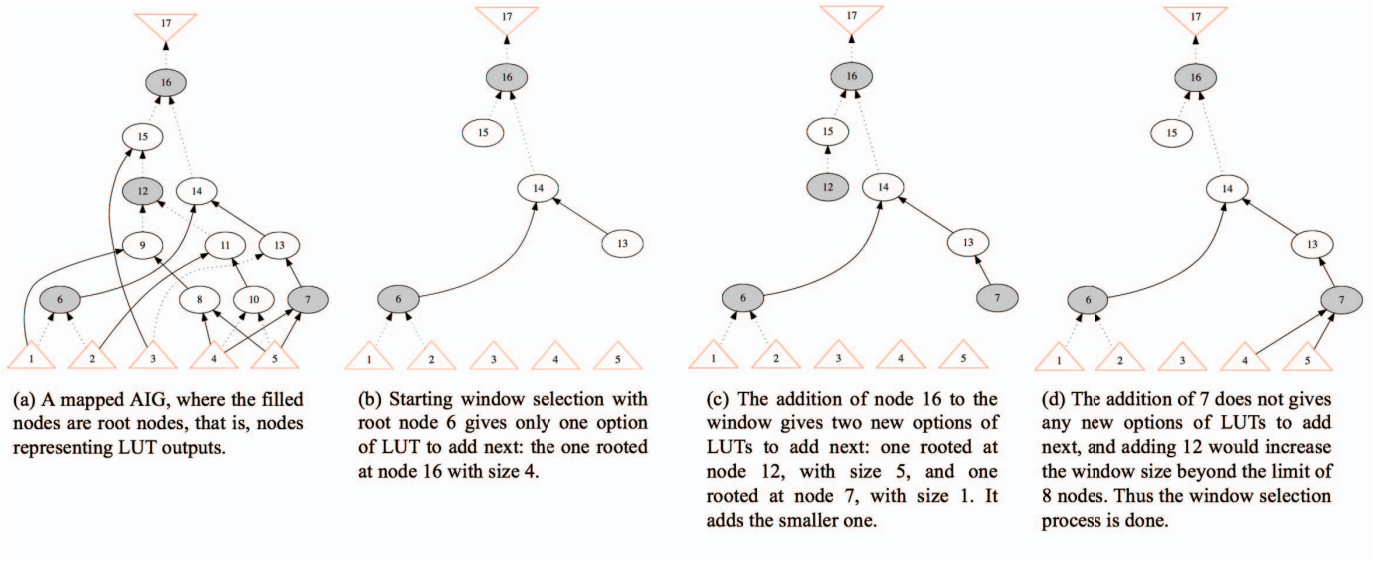

Fig. 2. Window selection in an AIG mapped into 4-LUT and window size limit set to 8 nodes.

E. Handling Delay Constraints

In the case of the LUT count optimization, the mapper focuses on reducing area without considering delay. If delay constraints are given, they could be converted into CNF and solved as part of the SAT problem [9]. However, in our experience, delay constraints substantially increase CNF size and slow down the SAT solver. To avoid this problem, one could, for example, use the method presented in [10]. In the present paper, a simpler approach inspired by [8] is used.

The main idea is to interface the SAT solver with a dedicated timer capable of determining if a mapping found by the solver is acceptable from the delay point of view. If it does not meet the timing constraints, the timer produces a critical path, which is a subset of LUTs used in the mapping that do not meet the timing. These LUTs are represented by specific SAT variables having value 1 in the current solution.

The found critical path can be ruled out by generating a blocking clause, which states that all the SAT variables representing LUTs on the critical path, cannot be 1 at the same time. When this clause is added to the SAT solver, it will guarantee that any future mapping generated by the SAT solver as a valid solution to the problem, does not contain this particular critical path.

The number of delay-violating critical paths in a small structural window is usually small and does not exceed 10. The proposed integration of the SAT solver and the timer quickly finds a timing-feasible solution, or enumerates mapping containing critical paths, blocks each of them, and concludes that no feasible solution exists.

IV. Experimental Results

The presented SAT-based remapping is implemented as command *&satlut* in ABC [2]. The implementation has been tested using a suite of EPFL benchmarks [6] mapped into 6-LUTs. The best area mappings available for these benchmarks were used as input to *&satlut*.

Command *&satlut* was run twice: first with the default settings (-N 32, and -C 100) and second with the high-effort settings (-N 64 and-C 10000), where switches N and C for the command specify the limit on AIG node count in the window and the limit on SAT conflicts, respectively. The results produced were checked for correctness using the combinational equivalence checker (*&cec*).

The detailed results are reported in I. The first section of the table shows the parameters for the area-optimized versions of the designs available from [7]. The parameters include the LUT count (column # LUT) and the number of levels (column # Level). The column # Nodes shows the number of nodes in the underlying AIG representation when the LUT network in BLIF format is given as input and converted into a mapped AIG using command *&get -m*.

The second section of Table I presents an evaluation of applying *&satlut* with default settings in terms of LUT count (column # LUT), the number of levels (column # Level), and the runtime in seconds (column Time, s). The third section of the table shows results using high-effort settings. At the bottom, reduction ratios relative to the original mapping are given; they were calculated using the geometric mean.

The delay limit imposed for each example in the experiments was the delay of the original LUT mapping. The table shows that *&satlut* reduces both area and delay. It is remarkable that an average area reduction of 3.5% is achieved with the default settings in a very short time. For several arithmetic benchmarks, the area reduction is very substantial: div (9.5%), log2 (7.5%), mult (11.7%), square (11.3%). We speculate that the large improvements are possible because these benchmarks have regular circuit structures, which can mislead area-recovery heuristics used in the structural LUT mapper.

The high-effort settings lead to even better results, but require substantially longer runtime. On some benchmarks, the high-effort results are worse than those produced by the default

TABLE I
THE RESULTS OF APPLYING *&satlut* TO EPFL BENCHMARKS MAPPED INTO 6-LUTS FOR AREA.

	Area-optimized statistics			*&satlut* -N 32 -C 100			*&satlut* -N 64 -C 10000		
Design	# Nodes	# LUT	# Level	# LUT	# Level	Time, s	# LUT	# Level	Time, s
adder	1981	201	73	201	73	0.05	201	73	0.11
arbiter*	1542	429	24	418	24	0.12	413	23	21.97
barrel	3840	512	4	512	4	0.08	512	4	0.25
cavlc	951	107	6	106	6	0.03	106	6	1.10
ctrl	169	28	2	28	2	0.01	28	2	0.01
dec	1072	272	2	272	2	0.02	272	2	0.02
div*	37378	3813	1542	3454	1211	1.03	3435	1217	6.07
i2c	1114	215	7	213	6	0.04	213	6	0.51
int2float	255	34	4	34	4	0.02	34	4	0.19
log2 *	46748	7344	142	6796	126	2.87	6633	121	284.48
max	3108	532	192	528	190	0.38	528	190	103.81
mem_ctrl*	10680	2125	23	2106	22	0.62	2103	23	87.43
mult *	54684	5681	120	5019	80	1.43	4923	90	11.32
priority	492	118	27	114	26	0.12	114	26	31.15
router	111	26	6	26	6	0.02	26	6	2.83
sin*	10209	1347	62	1285	55	0.42	1242	53	59.98
sqrt*	38306	3286	1180	3209	1116	0.75	3202	1109	3.41
square*	36468	3798	116	3371	88	1.16	3270	86	6.80
voter*	25699	1521	18	1395	17	0.39	1354	17	1.70
geomean:		**1.000**	**1.000**	**0.965**	**0.923**	**1.000**	**0.957**	**0.924**	**17.854**
geomean*:		**1.000**	**1.000**	**0.934**	**0.864**	**1.000**	**0.918**	**0.865**	**23.244**

settings. This is because (1) when looking at a larger window, it is often harder to find a feasible re-mapping even when the conflict limit is higher, (2) both algorithms are greedy, that is, accept improvements as they appear, which may preclude other optimization opportunities later on.

A delay improvement was achieved in most examples because the original mapping was done just for area.

After removing 10 smaller examples whose area did not change or changed by less than 5 LUTs (because they were likely near minimum already), we are left with 9 examples marked with an asterisk in the table. For this subset, the default settings improve area and delay by 7.6% and 13%, respectively, while the high-effort settings improve by 8.2% and 13.5%, respectively. These ratios are listed on the Geomean* row in Table I.

Similarly, Table II lists results for delay-optimized versions of the EPFL benchmarks. In this case, *&satlut* performed only area-recovery while constraining the delay to not exceed the original maximum delay for each testcase. More area would be saved if the engine considered the slack on near-critical paths. However, the engine is configured to not increase the delay on any path without using the slack.

V. CONCLUSIONS AND FUTURE WORK

The paper describes a fast SAT-based engine for recovering area after AIGs have been mapped into K-input LUTs. The method is purely structural and does not exploit functional properties of the design. However, using the default settings, even on examples that have been mapped already for minimum area using state-of-the-art mapping, area and delay are further reduced while runtime consumes only a few seconds. Substantial reductions in area are surprising and indicate that current area recovery heuristics are relatively weak and sub-optimal for some circuit types.

The engine, *&satlut*, can optimize area under delay-constraints. Delay constraints are not encoded in the CNF, as in past work [9]. Instead, the SAT solver is interfaced with a dedicated timing engine, which repeatedly detects timing violations on the critical paths and converts them into blocking clauses that are added to the SAT solver.

Currently, we use this SAT-based area recovery re-mapper as a post-processing step after a mainstream structural mapper. In the future, SAT-based mapping may become an integrated part of main-stream mappers, replacing complicated, error-prone, and often contradictory heuristics used to recover area under delay constraints. This integration may lead to mappers that are faster and produce better results. A conceptually similar approach [9][10] was used to find detailed placement, where a SAT-based placer provided improved runtime and quality of results, compared to previous methods based on simulated annealing.

To our knowledge, the proposed engine is the first that can find exact minimum-area solutions for non-trivial multi-input and multi-output AIGs, given that the AIG entirely covered by a window. Our experiments indicate that, given state-of-the-art SAT solvers and CNF encoding, exact solutions to structural mapping can be found for relatively small netlists composed of 10-20 LUTs.

Future work will proceed in the following directions:

- Further improving CNF representation of cardinality constraints. The motivation is that these constraints take about

TABLE II
THE RESULTS OF APPLYING *&satlut* TO EPFL BENCHMARKS MAPPED INTO 6-LUTS FOR DELAY.

	Area-optimized statistics			*&satlut* -N 32 -C 100			*&satlut* -N 64 -C 10000		
Design	# Nodes	# LUT	# Level	# LUT	# Level	Time, s	# LUT	# Level	Time, s
adder	2839	419	6	410	6	0.14	415	6	1.86
arbiter *	1834	542	6	533	6	0.13	533	6	2.67
barrel	3840	512	4	512	4	0.09	512	4	0.29
cavlc	956	120	4	119	4	0.03	115	4	2.87
ctrl	169	28	2	28	2	0.01	28	2	0.01
dec	1072	272	2	272	2	0.01	272	2	0.02
div *	74605	14576	238	14530	238	4.42	14553	238	18.55
i2c	1137	234	3	230	3	0.07	229	3	1.78
int2float	261	44	3	41	3	0.03	41	3	1.89
log2 *	57309	9275	55	9140	55	4.25	9221	55	33.27
max	4504	899	10	893	10	0.31	882	10	116.39
mem_ctrl*	10874	2234	6	2215	6	0.59	2216	6	16.13
mult *	54813	7095	29	6951	29	4.87	6944	29	8.57
priority	568	158	4	156	4	0.16	157	4	34.53
router	110	30	4	30	4	0.05	30	4	1.74
sin *	11087	1835	30	1801	30	0.65	1803	30	4.36
sqrt *	62832	11745	254	11711	254	3.36	11687	254	8.63
square *	41891	4201	11	4094	11	2.38	4036	11	8.09
voter *	26500	1515	12	1501	12	0.63	1469	12	2.23
geomean:		**1.000**	**1.000**	**0.987**	**1.000**	**1.000**	**0.983**	**1.000**	**11.844**
geomean*:		**1.000**	**1.000**	**0.987**	**1.000**	**1.000**	**0.984**	**1.000**	**5.737**

50% of the clauses in a typical SAT instance. As a result, any reduction in their number translates into increased scalability of the mapper.

- Extending SAT-based mapping to work for standard cells. A preliminary implementation confirmed that the approach is practical and leads to area savings.
- Creating a hybrid structural/functional SAT-based optimization mapping engine, which exploits both the efficient structural solution of the covering problem and the functional nature of the underlying Boolean network.

ACKNOWLEDGMENTS

This work was partly supported by NSF/NSA grant "Enhanced equivalence checking in cryptoanalytic applications" at University of California, Berkeley. We thank Jie-Hong Roland Jiang for discussions.

REFERENCES

[1] R. Asin, R. Nieuwenhuis, A. Oliveras, and E. Rodriguez-Carbonell, "Cardinality networks and their applications", *Proc. SAT09*, Springer, LNCS 5584, pp. 167-180.

[2] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification.

[3] M. Codish and M. Zazon-Ivry, "Pairwise cardinality networks", *Proc. LPAR10*, Springer, LNCS 6355, pp. 154-172.

[4] N. Een and N. Sorensson, "An extensible SAT-solver", *Proc. SAT03*, LNCS 2919, pp. 502-518.

[5] N. Een and N. Srensson, "Translating pseudo-Boolean constraints into SAT", *Journal of SAT*, Vol. 2, 2006, pp. 1-26.

[6] EPFL Benchmarks. http://lsi.epfl.ch/benchmarks

[7] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan. "Efficient SAT-based Boolean matching for FPGA technology mapping." *Proceedings of the 43rd annual Design Automation Conference (DAC '06)*.

[8] V. Ganesh, C. W. ODonnell, M. Soos, S. Devadas, M. C. Rinard, and A. Solar-Lezama "Lynx: A programmatic SAT solver for the RNA-folding problem", *Proc. SAT12*, LNCS 7317, pp. 143-156.

[9] A. Mihal and S. Teig, "A constraint satisfaction approach for programmable logic detailed placement", *Proc. SAT13*, LNCS 7962, pp. 208223.

[10] A. Mihal, "A difference logic formulation and SMT solver for timing-driven placement", *Proc. SMT13*.

[11] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts", *Proc. ICCAD '07*, pp. 354-361.

[12] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. "Scalable don't-care-based logic optimization and resynthesis", *ACM TRETS*, Vol. 4(4), April 2011, Article 34.

[13] A. Mishchenko, R. Brayton, T. Besson, S. Govindarajan, H. Arts, and P. van Besouw, "Versatile SAT-based remapping for standard cells", *Proc. IWLS'16*.

[14] I. Parberry, "The pairwise sorting network", *Parallel Processing Letters*, 2 (2, 3), 1992, pp. 205211.

[15] A. Mishchenko, R. Brayton, W.Feng, and J. Greene. 2015. "Technology Mapping into General Programmable Cells." *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*