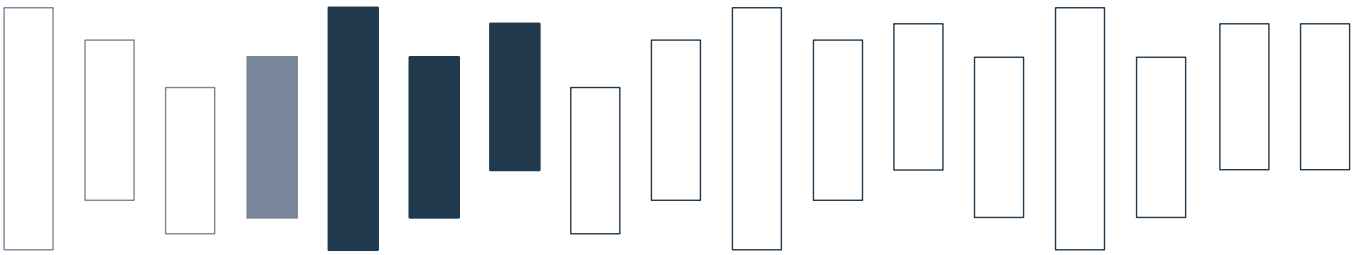
Technical Report AC-TR-24-001

January 2024

Improved Circuit Minimization with Exact Synthesis

Franz-Xaver Reichl, Friedrich Slivovsky and Stefan Szeider



Improved Circuit Minimization with Exact Synthesis

***Abstract*—Exact synthesis aims to find a smallest circuit matching a given specification. While SAT-based exact synthesis is currently limited to very small circuits, it can help minimize larger circuits by optimally resynthesizing subcircuits. The function implemented by a subcircuit can often be modified without affecting the function computed by the overall circuit. Fully capturing and exploiting such flexibilities, called "don't cares", is computationally expensive, but can be beneficial in high-effort optimization methods.**

A recently proposed approach uses Quantified Boolean Formulas (QBF) to succinctly encode resynthesis of multi-output subcircuits subject to don't cares. This paper describes two improvements of this approach. First, it presents a purely propositional encoding based on a Boolean relation characterizing the input-output behavior of the subcircuit under don't cares. This allows the use of a SAT solver for resynthesis, substantially reducing running times when applied to functions from the recent IWLS competition. Second, it proposes circuit partitioning techniques in which don't cares for a subcircuit are captured only with respect to an enclosing window, rather than the entire circuit. Circuit partitioning trades completeness for efficiency, and successfully enables the application of exact synthesis to some of the largest circuits in the EPFL suite, leading to improvements over the current best implementation for several instances.

I. Introduction

Modern integrated circuits have grown increasingly large and complex, making their design and optimization a significant challenge. Automation has become indispensable for the process of circuit design, including logic optimization and logic synthesis, which collectively lead to substantial reductions in the number of gates and circuit depth [1], [2].

However, finding a small circuit that implements a given Boolean function is challenging, and exact synthesis, which yields provably optimal results, does not currently scale beyond circuits of about 10 gates [3], [4]. To deal with larger circuits, one can first partition them into smaller subcircuits, and then minimize these using exact techniques [5].

In practice, this approach is typically restricted to single-output subcircuits [6], [7]. Although efficient, this does not fully exploit the implementation flexibility of multi-output subcircuits. Recently, a high-effort method for resynthesizing multi-output subcircuits based on Quantified Boolean Formulas (QBF) has been proposed [8], which has shown success in minimizing circuits from the IWLS'22 competition and the EPFL combinational benchmark suite. Specifically due to increasing prices of silicon wafers in recent years [9] such high-effort methods are gaining importance.

In this paper, we describe two improvements of this approach. First, we present a workflow purely based on SAT instead of QBF. Generating a SAT encoding requires computing the input-output relation on the care set (the complement of don't cares) for each subcircuit, whereas the QBF encoding captures don't cares implicitly. Our rationale is that the ability to use a SAT solver instead of a QBF solver will more than make up for this extra step when the relation is reasonably small. Second, we incorporate windowing to handle very large circuits. For such circuits, the QBFs encoding the existence of replacements for subcircuits become too hard for QBF solvers, and computing the input-output relation needed for the new SAT encoding takes too much time. To address this, we adapt a strategy from prior work on computing don't cares of single-output subcircuits [6]. Instead of ensuring that a resynthesized subcircuit preserves the Boolean function computed by the full circuit, we only require that it preserves the function computed by a "window" containing the subcircuit to be replaced. In theory, this means don't cares of a subcircuit are no longer fully captured. In practice, with windows containing hundreds of gates, we expect don't cares to be virtually identical.

As in the case of resynthesis of multi-output subcircuits, rewriting multi-output windows is non-trivial since acyclicity of the circuit obtained from replacing the window has to be ensured. We describe an algorithm that partitions the circuit into layers and generates windows that can be optimized independently and in parallel while ensuring acyclicity.

We performed an experimental evaluation of these improvements. The SAT-based workflow proved to be substantially faster for small circuits, showing a significant performance increase for instances in the IWLS'23 competition. For circuits from the EPFL combinational benchmark suite, the SAT and the QBF-based approach showed comparable performance. Using windowing, we were able to scale both approaches to the largest circuits in this set, which had previously been unmanageable.

A. Related Work

Methods that fully capture the properties of Boolean functions implemented by circuits (rather than considering them as polynomials, for instance) are deemed the most effective in logic synthesis [5]. However, these methods are also the most computationally expensive, and their application to large circuits is limited to resynthesizing small subcircuits. SAT-based exact synthesis [3], [4] and SAT-based resubstitution [10], [11], which aims to represent the function implemented by a specific gate as a function of a few existing gates in the circuit, are examples of this approach. Many of these methods are implemented in the industrial-strength tool ABC [12].

A SAT-based method for capturing don't cares close to ours has been previously considered for single-output subcircuits, including the use of windowing to improve scalability [6]. Boolean relations have been proposed as a means of representing don't cares of multi-output subcircuits [13]. The corresponding optimization workflow relies on a simulation-based under-approximation of don't cares in combination with

a divide-and-conquer algorithm for resynthesis, whereas we use a SAT solver for both don't care computation and resynthesis.

Exact resynthesis of subcircuits has also been explored for finding optimal circuits for symmetric functions in circuit complexity research, but without incorporating don't cares [14]. This is an instance of the *SAT-based Local Improvement Method (SLIM)*, a general optimization framework that has been applied to various AI problems (see, for instance, [15]).

In the realm of logic synthesis, QBFs have been employed for bi-decomposition [16], reversible quantum circuit synthesis [17], and lookup table (LUT) synthesis [18]–[20]. The latter two problems impose more constraints than the setting considered in this work, as they maintain a fixed circuit topology. In contrast, our synthesis tasks also involve determining a suitable topology.

II. Preliminaries

A *Boolean circuit* is a *directed acyclic graph*. We denote the source nodes of a circuit $\mathcal{C}$ as the *primary inputs* $\mathrm{in}(\mathcal{C})$ and the non-source nodes as *gates*. The *primary outputs* $\mathrm{out}(\mathcal{C})$ are a subset of the nodes in $\mathcal{C}$. If there is an edge from node n to node m then n is an *input* of m. Each gate corresponds to a Boolean function on its inputs. Let n be a node in $\mathcal{C}$ then the *transitive fanin cone* of n $TFI(n)$ is the set of all nodes in $\mathcal{C}$ from which n is reachable. Similarly, the *transitive fanout cone* of n $TFO(n)$ is the set of all nodes in $\mathcal{C}$ which can be reached from n. A node n *depends* on a node m if $m \in TFI(n)$. Moreover, for a node n we define its *level* $\mathrm{lv}(n)$ as 0 if n is a primary input and else as $1 + \max(\{\mathrm{lv}(x) \mid x \in inputs(n)\})$.

A *Quantified Boolean formula (QBF)* is of the form $\forall X_1 \exists X_2 \ldots \forall X_{k-1} \exists X_k . \varphi$, where the X_i are pairwise disjoint sequences of variables, and φ is a propositional formula called the *matrix*. The quantifiers range over the Boolean domain $\{0,1\}$, so that existential ($\exists$) quantifiers can be understood as abbreviating a disjunction $\exists x.\varphi \equiv \varphi[x \leftarrow 0] \vee \varphi[x \leftarrow 1]$, and universal ($\forall$) quantifiers as encoding a conjunction $\forall x.\varphi \equiv \varphi[x \leftarrow 0] \wedge \varphi[x \leftarrow 1]$. Evaluating QBFs is a PSPACE-complete task, and QBFs can succinctly encode problems arising in many areas [21]. For an overview of QBF, including solving techniques and proof complexity, see [22].

III. Exact Synthesis of Subcircuits

Our approach involves replacing a subcircuit with a smaller one, ensuring that the function computed by the encompassing circuit remains unchanged. For a fixed value ℓ, we use a logic solver (SAT or QBF) to determine whether there is replacement circuit of size ℓ. To find a smallest possible circuit, the value ℓ is decremented until the encoding becomes unsatisfiable.

Throughout this section, let $\mathcal{C}$ denote the encompassing circuit, $\mathcal{S}$ one of its subcircuits, and $\mathcal{T}$ the replacement circuit. Further, n is the number of inputs of the subcircuit $\mathcal{S}$ and m the number of outputs. Moreover, we assume that $\mathcal{C}$ is *k-regular*, i.e., each gate in $\mathcal{C}$ has k inputs. Let $\mathcal{C}[\mathcal{S} \leftarrow \mathcal{T}]$ denote the result of substituting the circuit $\mathcal{T}$ for the subcircuit $\mathcal{S}$ in $\mathcal{C}$. Our goal is to find a circuit $\mathcal{T}$ of size ℓ with n inputs and m outputs such that $\mathcal{C}$ and $\mathcal{C}[\mathcal{S} \leftarrow \mathcal{T}]$ are logically equivalent, formally $\mathcal{C} \equiv \mathcal{C}[\mathcal{S} \leftarrow \mathcal{T}]$. We initially set $\ell = |\mathcal{S}|$ and then decrement ℓ until the logic solver determines that no such circuit $\mathcal{T}$ exists, at which point we can conclude that the circuit $\mathcal{T}$ must have at least $\ell+1$ gates. A circuit $\mathcal{T}$ of this size can be constructed from a model of the last satisfiable encoding, and used to replace $\mathcal{S}$.

It may seem unnecessary to initially ask the logic solver to come up with a circuit of size $\ell = |\mathcal{S}|$, since we already know such a circuit exists. However, the new circuit is typically not equivalent to $\mathcal{S}$, and replacing $\mathcal{S}$ by $\mathcal{T}$ is often beneficial in the overall minimization process even though it does not immediately decrease size [8].

In the next two subsections, we will provide details on the QBF and SAT encodings we use. Both are closely related to and inspired by the *multi selection variable* SAT encoding for exact synthesis [4].

A. The QBF Encoding

The main advantage of a QBF encoding is that one can universally quantify over assignments of primary inputs and encode the value computed by each gate using a single variable, rather than introducing a variable for each line in the truth table representation of $\mathcal{C}$. In addition to these **gate variables** and primary inputs, the encoding contains the following groups of existentially quantified variables that determine the structure of the circuit:

Selection variables $S_i = \{s_{it} \mid 1 \leq t < i+n\}$. These variables determine the inputs of the i^{th} gate. If s_{it} is true then node t is an input of gate i.

Function variables $F_i = \{f^i_{a_1 \ldots a_k} \mid \bigwedge_{l=1}^{k} 0 \leq a_l \leq 1\}$. These variables describe the Boolean function at gate i. If $f^i_{a_1,\ldots,a_k}$ is true then the function yields true for the inputs $a_1, \ldots, a_k$, and false otherwise.

Output variables $O_j = \{o_{tj} \mid 0 \leq t \leq n + \ell\}$. These variables fix the output gates of the circuit. If o_{tj} is true then the j^{th} output is given by the constant value false if $t = 0$ and otherwise by the t^{th} node.

The matrix of the QBF encodes the following constraints:

- Each gate must have exactly k inputs, i.e., at each gate i exactly k selection variables must be true. This can be enforced by using a sequential counter [23]. We denote this constraint by $Count(S_i, k)$.
- Each output must correspond to a single gate, i.e., for each output j exactly one output variable is true. We denote this constraint by $Count(O_j, 1)$.
- For each gate i the assignment of the gate variable must be consistent with the function determined by the function variables. Assume that the assignment for the function variables describes the function F and that the values of the inputs of i are given by $i_1, \ldots, i_k$. This constraint ensures that the gate variable for i is assigned to $F(i_1, \ldots, i_k)$. We denote this constraint by $Comp_i$.
- Replacing $\mathcal{S}$ by $\mathcal{T}$ must preserve the function computed by $\mathcal{C}$. To express this, one can use Tseitin transformation and two sets of gate variables to encode both the specification $\mathcal{C}$ and the circuit $\mathcal{C}' = \mathcal{C}[\mathcal{S} \leftarrow \mathcal{T}]$. For an output gate o, let v_o and v'_o denote the gate variable in the encoding of $\mathcal{C}$ and $\mathcal{C}'$, respectively. We add a constraint $v_o \Leftrightarrow v'_o$ and denote their conjunction over all outputs by $Corr$.

Let $S = \bigcup_{1 \leq i \leq k} S_i$ denote the selection variables, F the gate definition variables, O the output variables, I the input variables, and G, G' the two sets of gate variables. The complete QBF encoding has the following form:

$$\exists S, F, O \, \forall I \, \exists G, G'. \, Corr \wedge \bigwedge_{1 \leq j \leq m} Count(O_j, 1) \wedge \bigwedge_{1 \leq i \leq k} (Count(S_i, k) \wedge Comp_i).$$

B. The SAT Encoding

While the QBF encoding handles don't cares implicitly, the SAT-based approach outlined in this subsection separates the tasks of computing don't cares and synthesizing a subcircuit. More specifically, it first computes a *Boolean relation* [2], [24] representing the input-output behavior of the subcircuit $\mathcal{S}$ on the *care set* (the complement of don't cares), and then uses the *multi selection variable* SAT-encoding [4] to obtain a circuit.

A Boolean relation R for $\mathcal{S}$ maps each input assignment to a set of permissible output assignments, formally $R : \{0,1\}^{\mathrm{in}(\mathcal{S})} \to (\mathcal{P}(\{0,1\}^{\mathrm{out}(\mathcal{S})}) \setminus \emptyset)$. A circuit C *implements* the relation R if $C(\sigma) \in R(\sigma)$ for each $\sigma \in \{0,1\}^{\mathrm{in}(C)}$. A Boolean relation R can be represented by a set of clauses ψ, containing for each $\sigma \in \{0,1\}^{\mathrm{in}(C)}$ and each $\rho \in \{0,1\}^{\mathrm{out}(C)} \setminus R(\sigma)$ the clause $\neg\sigma \vee \neg\rho$. This means that for each $\sigma \in \{0,1\}^{\mathrm{in}(C)}$ and $\rho \in \{0,1\}^{\mathrm{out}(C)}$ the assignment $\sigma \cup \rho$ satisfies ψ iff $\rho \in R(\sigma)$.

The core idea for computing the relation is to first obtain a copy $\mathcal{C}'$ of $\mathcal{C}$ by removing all gates from $\mathcal{S}$[1]. Due to the removal of gates $\mathcal{C}'$ contains an additional primary input for each output of $\mathcal{S}$. We denote these new inputs by $\mathcal{I}$. Next we compute every assignment σ_1 for $\mathrm{in}(\mathcal{S})$ and σ_2 for $\mathcal{I}$ such that $\mathcal{C}$ and $\mathcal{C}'$ differ in at least one common primary output. This means that replacing $\mathcal{S}$ by a circuit that yields σ_2 for σ_1 changes the entire Boolean function. Thus, (σ_1, σ_2) must not be contained in the relation. Replacing $\mathcal{S}$ by any circuit implementing the resulting relation does not change the computed Boolean function.

To realize this idea we first compute clausal encodings φ_1 for $\mathcal{C}$ and φ_2 for $\mathcal{C}'$ by Tseitin transformation, introducing a propositional variable for each node. For each node x in $\mathcal{C}$ we denote the corresponding variable by $v(x)$ and for each node x in $\mathcal{C}'$ by $v'(x)$. Similarly, we define $v \,/\, v'$ for sets of nodes. Next we introduce for each common primary output o the constraint $v(o) \Leftrightarrow v'(o)$, denoting the set consisting of all these clauses by $equiv$.

This encoding can now be used to compute the relation with incremental SAT solving. The algorithm maintains a set of blocking clauses B, which is empty initially. We now ask the SAT solver for an assignment σ that satisfies $\varphi_1 \wedge \varphi_2 \wedge B$ and falsifies $equiv$. Based on the above considerations we could directly add $\neg\sigma|_{v(\mathrm{in}(\mathcal{S}))} \vee \neg\sigma|_{v'(\mathcal{I})}$ to the clausal representation of the relation. To obtain a more compact encoding of the relation, we can try to reduce $\sigma|_{v'(\mathcal{I})}$ as follows. The formula $\varphi_1 \wedge \varphi_2 \wedge equiv$ is unsatisfiable under the assumption $\sigma|_{v(\mathrm{in}(\mathcal{C}))} \wedge \sigma|_{v'(\mathcal{I})}$. Using the SAT solver, we can compute a subset $\hat{\sigma}$ of failed assumptions. As $\hat{\sigma}$ suffices to make $\varphi_1 \wedge \varphi_2 \wedge equiv$ unsatisfiable it is rather easy to conclude that for each assignment μ with $\hat{\sigma}|_{v'(\mathcal{I})} \subseteq \mu$, the pair $(\sigma|_{v(\mathrm{in}(\mathcal{S}))}, \mu)$ must not be contained in the relation. Thus, we can add $\neg\sigma|_{v(\mathrm{in}(\mathcal{S}))} \vee \neg\mu$ to the clausal representation of the relation. Moreover, to avoid the same inconsistency in subsequent iterations, we add the blocking clause $\neg\sigma|_{v(\mathrm{in}(\mathcal{S}))} \vee \neg\mu$ to B.

As mentioned above, a circuit implementing the relation R can be synthesized using a slight adaptation of the SAT encoding for exact synthesis by Haaswijk et al. [4].

[1] Actually, it suffices to consider $TFO(\mathcal{S})$ instead of a copy of the entire circuit

IV. Minimization by Subcircuit Resynthesis

We use exact synthesis of subcircuits as a subroutine in a circuit minimization algorithm that repeatedly selects subcircuits for resynthesis. To obtain a subcircuit for resynthesis, we start from a *root gate*, and then expand by incorporating successors of previously chosen gates until reaching a predetermined size. Root gates are chosen randomly from the circuit. To expand the root gate we visit gates which use previously selected gates as inputs in a breadth-first search like manner. We then randomly decide whether to include this gate in the subcircuit. Unlike to previous work [8] we use a fixed bound for the size of subcircuits. This bound is decreased in case individual checks timeout.

V. Window Selection

Both the SAT and the QBF-based rewriting approach do not only depend on the selected subcircuits but also on the entire circuit. This is necessary in order to make use of don't cares. But this also means that in general rewriting subcircuits gets harder for larger circuits. In order to overcome this issue, we only consider don't cares with respect to a *window* (a subcircuit) [6]. The window is chosen such that the size is still manageable for our QBF/SAT-based approach. In this manner, we can minimize windows, and since the functions computed by windows are preserved, the optimized implementation can be used to replace the original window.

Additionally, using windows allows us to rewrite subcircuits simultaneously. For this purpose, we compute pairwise disjoint windows and rewrite them separately. In the end we then combine the optimized new implementations for the windows to obtain a new implementation for the original circuit.

A major challenge is that in the recombination step, cyclic dependencies can be introduced into the circuit. This is possible because rewriting a window can add additional inputs of the window to the TFI of one of its outputs. To be more precise, since multi-output subcircuits are used, inputs in the TFI of one output may be added to the TFI of another output. These additional inputs can then close a cycle, as illustrated in Fig. 1.

In order to ensure acyclicity, we introduce what we call the *level constraint* for a window W. The level constraint asserts that inputs of W have smaller levels than any output, formally:

$$\max(\{\mathrm{lv}(x) \mid x \in \mathrm{in}(W)\}) \leq \min(\{\mathrm{lv}(x) \mid x \in \mathrm{out}(W)\}).$$

By ensuring that for each window the level constraint holds we can show that rewriting windows does not introduce cycles.

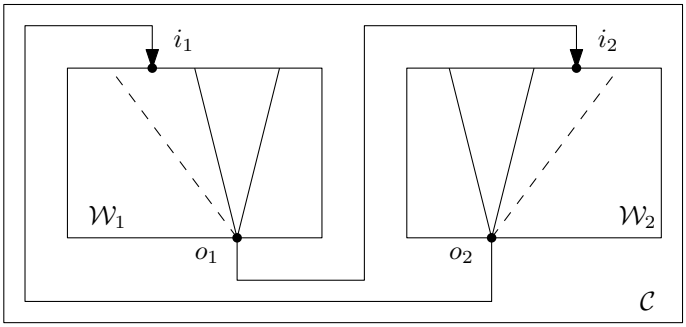


Fig. 1: We consider two windows $\mathcal{W}_1$ and $\mathcal{W}_2$ in the circuit $\mathcal{C}$. Initially, $o_1(o_2)$ does not depend on $i_1(i_2)$. Minimizing the windows changes the TFIs of o_1 and o_2. The changed cones are indicated by the dashed lines. The changed cones cause a cycle as i_1 is reachable from itself via o_1, i_2 and o_2.

Proposition 1. *Let $\mathcal{C}$ be a circuit and $W_1, \ldots, W_w$ be pairwise disjoint subcircuits of $\mathcal{C}$, where $w \in \mathbb{N}$. If each window satisfies the level constraint then replacing the windows by equivalent implementations in $\mathcal{C}$ does not introduce cycles.*

Similarly as for the computation of subcircuits the computation of windows satisfying the level constraints involves the computation of a root gate and the expansion of the root. To compute the root we randomly select either a primary input or an output of a previously computed window. For expanding the root we apply Algorithm 1. The core idea of this algorithm is to add successors of previously added gates to the window until a sufficiently large window is obtained. Since we require pairwise disjoint windows the algorithm takes the set of already used gates as an argument ($taboo$). Gates in $taboo$ are not allowed to be added to the window. Adding a gate g to the window can violate the level constraint because the level of one of its inputs may be greater than the current minimal level of an output ($level$). In order to repair such a violation, the algorithm computes the set of all gates in $TFI(g)$ whose level is greater than $level$ ($repair$). By adding $repair$ to the window the level constraint can be maintained. If $repair$ contains an element of $taboo$ we cannot add $repair$ to win. This means that the level constraint cannot be reestablished and so g is discarded. Finally, all new successors of the window which are not contained in $taboo$ are added to the set of potential new gates.

VI. Experiments

We implemented the presented techniques mainly in Python and some parts in C++.[2] As backend solvers we used QFUN [25] for QBF and CaDiCaL [26] for SAT. We evaluated our tool on the instances from the IWLS'23[3] programming contest and the instances from the EPFL benchmark suite [27]. All experiments were conducted on a cluster with AMD EPYC 7402 processors at 2.8 GHz running 64-bit Linux. We used a memory limit of 4 GB. For the parallelized minimization we used a memory limit of 4 GB per thread.

[2]https://github.com/Anonymous-753/DATE24-138
[3]https://github.com/alanminko/iwls2023-ls-contest

Algorithm 1 Computing Disjoint Windows

```
1:  procedure EXPAND(root, taboo, size)
2:      toConsider ← GETSUCCESSORS(root) \ taboo
3:      win ← ∅, level ← ∞
4:      while toConsider ≠ ∅ ∧ |win| < size do
5:          g ← pop g from toConsider with minimum level
6:          repair ← {x ∈ TFI(g) | lv(x) > level}
7:          if taboo ∩ repair ≠ ∅ then
8:              continue
9:          win ← win ∪ repair ∪ {g}
10:         level ← min({lv(x) | x ∈ out(win)})
11:         UPDATE(toConsider, taboo, repair ∪ {g})
12:     return win
```

A. IWLS Instances

The IWLS'23 instances consist of 100 instances, given as truth tables. The goal is to compute an *And-Inverter Graph* (AIG) with as few gates as possible. In order to compute AIGs the encodings needed to be slightly updated, such that only valid AIG gates are synthesized. This was realized by adding additional constraints for the function variables, which rule out XOR-gates.

Since the instances are given as truth tables, and our tool requires that specifications are given as circuits, we had to preprocess the instances using ABC [12]. Because a naive transformation of truth tables to circuits by using ABC in general results in relatively large circuits, we used ABC to reduce the size of the initial circuit. For this purpose, we determined by hand several sequences of ABC commands to compute a circuit from a truth table. We then applied each sequence of commands to each instance and selected the circuit with the fewest gates.

In our evaluation setup, we considered our tool both in the QBF-based and the SAT-based configuration. As only very few of the IWLS instances are sufficiently large for a reasonable application of windowing we did not evaluate it here. We compared our tool with the QBF-based minimization tool CIOPS [8] and the DEEPSYN procedure from the state-of-the-art synthesis tool ABC [12]. We chose DEEPSYN because it allows anytime optimization of circuits, like our tool. Thus, unlike other ABC commands and scripts DEEPSYN is able to take advantage of the long run times which we used for our experiments. Preliminary tests showed that for some of the smallest instances DEEPSYN terminates before the time budget is exhausted due to an internal limit on the number of iterations. For a better comparison we slightly modified ABC by increasing this limit.

In our experiments, we alternated between 30-minute runs of our reduction tool (respectively CIOPS) and exhaustive heuristic minimization with ABC for inprocessing. The choice of ABC commands was informed by preliminary testing results. We repeated this alternating pattern eight times.

Since the time needed for the inprocessing steps varied depending on the analyzed circuit, the total runtimes of our tool and CIOPS differ for different inputs. In order to still

TABLE I: Average reduction (%) of gates compared to the preprocessed IWLS'23 instances, by configuration and initial size and standard deviations of the average reductions per configuration.

#Gates	CIOPS		DEEPSYN		QBF		SAT	
10-44	6.62	0.2	3.32	0	5.6	0.29	5.65	0.28
45-142	12.99	0.51	6.82	0	18.29	0.49	20.03	0.56
147-499	11.63	0.24	15.85	0.02	21.29	0.85	24.88	0.45
516-7171	6.53	0.15	23.89	0.13	15.75	0.63	22.04	0.45
Overall	9.44	0.16	12.47	0.04	15.23	0.44	18.15	0.31

give a fair comparison with DEEPSYN we computed for each instance the maximal total runtime among all runs of our tool and used them as timeouts for DEEPSYN. In general, all tools benefited from longer runtimes. Nevertheless, we limited the runs to roughly four hours due to constraints on the available computational infrastructure. Additionally, we set the initial bound for the subcircuit size to 6 both for our tool and CIOPS (cf. Section IV).

Instances were grouped into four subsets of 25 based on the initial number of gates. For each configuration and instance group, we determined the average size reduction (in %) for circuits in that group. We performed 5 independent runs of each configuration and report averages and standard deviations. Results are given in Table I.

The experiments show that our tool with the QBF configuration clearly outperformed CIOPS. This was mainly due to two simple but apparently effective changes. First, our tool uses fixed bounds for the sizes of selected subcircuits, while CIOPS tries to increase the initially given bounds as far as possible. Since larger circuits are usually harder to analyze this indicates that in general it is advantageous to consider more but simpler (smaller) circuits. Second, our tool computes subcircuits by expanding root gates in a randomized breadth-first-like manner, while CIOPS applies an expansion strategy that aims at keeping the number of outputs low.

Moreover, the SAT-based strategy outperformed the QBF-based strategy. This was possible as in general the SAT-based approach allowed a faster analysis of subcircuits. Thus, more subcircuits could be analyzed in total. Furthermore, the experiments show that our tool is a viable alternative to DEEPSYN.

Additionally, if we consider the best implementations for each instance among the 5 SAT runs we can observe an average reduction of 20.22%. This indicates that it may be advantageous to consider multiple runs of our approach. As our method makes use of a randomized subcircuit selection, different runs result in different sequences of replaced subcircuits. It is not only possible that in one run subcircuits are selected which are just easier to reduce. It is also possible that one sequence leads to some local minimum, which is difficult to escape.

B. EPFL Instances

In order to evaluate our tool for circuits with non-binary gates, we considered the *EPFL Combinational Benchmark Suite* [27]. This benchmark set consists of twenty circuits[4]. The goal is to find LUT-6 implementations of the specifications with small size. In addition to a specification given as circuit with binary gates the benchmark suite also provides the best known realizations so far. We used the best known realizations as initial specifications for our tool.[5]

TABLE II: Results for small EPFL LUT-6 instances. The best results are marked in boldface.

Instance	Initial	CIOPS	QBF	SAT
Lookahead XY router	**19**	19	19	19
int to float converter	20	20	**18**	19
Alu control unit	**25**	25	25	25
Coding-cavlc	54	52	**49**	53
Priority encoder	94	94	93	**92**
Adder	**129**	129	129	129
I2c controller	182	178	179	**177**
Decoder	**264**	264	264	264
Round-robin arbiter	273	273	272	**267**
Max	**511**	511	511	511
Barrel shifter	**512**	512	512	512

We ran our reduction tool for 12 hours both with the SAT and the QBF configuration. After each hour we applied the ABC command &MFS as an inprocessing step. Additionally, we also applied our tool with windowing enabled. Here we recombined the windows for the inprocessing step and computed new windows afterwards. We compared our tool with CIOPS. We want to point out that the initial realizations we use have been highly optimized by different methods. Thus, we think that any improvement of these circuits can be considered a success.

In addition to a bound for the size for the subcircuits, we also used a limit of 10 on the number of inputs of the subcircuits considered for resynthesis. Preliminary tests showed that such a limit is required to reliably generate the Boolean relation for the SAT-based approach within time and memory limits. Additionally, we always set the initial bound for the subcircuit size to 4. In the experiments with windowing we used two different window sizes, 500 and 1000. First we minimized single windows and second up to 8 windows concurrently. We only applied windowing for instances with at least 1000 gates.

Results for instances with at most 1000 gates are given in Table II and for instances with more than 1000 gates in Table III. The used window sizes are given in the column titles, e.g., *QBF500* corresponds to QBF-based minimization with windows of size 500.

Since the initial circuits are already highly optimized by state-of-the-art methods, the relative improvements for the EPFL instances were small compared to the IWLS instances, and it is difficult to draw any definitive conclusions about the superiority of any configuration from these results. Nevertheless, the results suggest that parallel optimization was able to beat single-threaded optimization. Similarly, the results indicate that both configurations of our tool outperformed CIOPS.

[4]We did not consider the MtM instances as the EPFL repository does not contain the best implementations for these circuits.

[5]Note that the best known results are continuously updated. We used the best known results as of May 22nd, 2023 (commit *42c1f31*).

TABLE III: Results for large EPFL LUT-6 instances. The best results are marked in boldface.

			No Windowing		Single Window				Up to 8 Windows			
Instance	Initial	CIOPS	QBF	SAT	QBF500	QBF1000	SAT500	SAT1000	QBF500	QBF1000	SAT500	SAT1000
Sine	1114	1111	1095	1085	1076	1096	1069	1088	1057	1095	**1036**	1089
Voter	1217	1217	1217	1179	1184	1215	1180	**1166**	1177	1210	1172	1174
Memory controller	1735	1731	**1722**	1727	1731	1731	1730	1730	1726	1724	1734	1731
Square-root	2994	2994	2994	2994	2991	2992	2985	2989	2992	2994	**2980**	2988
Square	3018	3018	3014	2997	2992	2996	2997	2994	**2942**	2962	2949	2943
Divisor	3096	3096	3096	3096	3096	3096	3096	**3095**	3096	3096	**3095**	3096
Multiplier	4360	4360	4358	4346	4347	4346	4349	4346	4326	4331	**4317**	4323
Log2	6133	6133	6132	6109	6127	6127	6129	6129	6078	6082	6069	**6063**
Hypotenuse	39452	39452	39452	39452	39276	39230	39251	39296	38535	**38459**	38815	38781

Moreover, our tool could improve on the best implementation for the majority of instances.

VII. Conclusion

The experimental analysis shows that the SAT-based approach outperforms the QBF-based version for instances from the IWLS'23 programming contest. For circuits from the EPFL suite, the two variants perform very similarly, with either approach having a slight edge on some instances. Moreover, the experiments show that using windowing allows the largest circuits from the EPFL suite to be further reduced.

The respectable performance of the QBF-based approach on larger circuits hints at the potential of adopting techniques from QBF solving, such as counterexample-guided expansion [28]. Computing the entire Boolean relation upfront can be prohibitive, and generating constraints during the substitution process could be more efficient. Specifically, constraints could be added on-the-fly when the substitution of a synthesized circuit alters the function.

Another promising avenue for future work is the use of SAT solvers specifically designed for logic synthesis [29].

References

[1] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[2] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[3] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "Finding efficient circuits using sat-solvers," in *SAT*, ser. LNCS, vol. 5584. Springer, 2009, pp. 32–44.

[4] W. Haaswijk, M. Soeken, A. Mishchenko, and G. D. Micheli, "Sat-based exact synthesis: Encodings, topology families, and parallelism," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 4, pp. 871–884, 2020.

[5] E. Testa, L. G. Amarù, M. Soeken, A. Mishchenko, P. Vuillod, P. Gaillardon, and G. D. Micheli, "Extending boolean methods for scalable logic synthesis," *IEEE Access*, vol. 8, pp. 226 828–226 844, 2020.

[6] A. Mishchenko and R. K. Brayton, "Sat-based complete don't-care computation for network optimization," in *DATE*. IEEE Computer Society, 2005, pp. 412–417.

[7] H. Riener, W. Haaswijk, A. Mishchenko, G. D. Micheli, and M. Soeken, "On-the-fly and dag-aware: Rewriting boolean networks with exact synthesis," in *DATE*. IEEE, 2019, pp. 1649–1654.

[8] F.-X. Reichl, F. Slivovsky, and S. Szeider, "Circuit minimization with QBF-based exact synthesis," in *AAAI*. AAAI Press, 2023.

[9] D. Gai, "Tsmc price hike indicates capacity tightness to persist in 2022, may hit smartphone shipments," online, 2021. [Online]. Available: https://www.counterpointresearch.com/tsmc-price-hike/

[10] A. Mishchenko, R. K. Brayton, J. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 34:1–34:23, 2011.

[11] H. Riener, A. Mishchenko, and M. Soeken, "Exact dag-aware rewriting," in *DATE*. IEEE, 2020, pp. 732–737.

[12] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *CAV*, ser. LNCS, vol. 6174. Springer, 2010, pp. 24–40.

[13] T.-Y. Lee, C.-C. Wu, C.-C. Lin, Y.-C. Chen, and C.-Y. Wang, "Logic optimization with considering boolean relations," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 761–766.

[14] A. S. Kulikov, D. Pechenev, and N. Slezkin, "Sat-based circuit local improvement," in *MFCS*, ser. LIPIcs, vol. 241. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 67:1–67:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.MFCS.2022.67

[15] V. P. Ramaswamy and S. Szeider, "Learning large Bayesian networks with expert constraints," in *UAI*, ser. PMLR, 2022, p. 180:1592–1601.

[16] H. Chen, M. Janota, and J. Marques-Silva, "QBF-based boolean function bi-decomposition," in *DATE*. IEEE, 2012, pp. 816–819.

[17] R. Wille, H. M. Le, G. W. Dueck, and D. Große, "Quantified synthesis of reversible logic," in *DATE*. ACM, 2008, pp. 1015–1020.

[18] M. Fujita, S. Jo, S. Ono, and T. Matsumoto, "Partial synthesis through sampling with and without specification," in *ICCAD*. IEEE, 2013, pp. 787–794.

[19] M. Fujita, "Toward unification of synthesis and verification in topologically constrained logic design," *Proc. IEEE*, vol. 103, no. 11, pp. 2052–2060, 2015.

[20] M. Fujita, Y. Kimura, X. Le, Y. Miyasaka, and A. M. Gharehbaghi, "Synthesis and optimization of multiple portions of circuits for ECO based on set-covering and QBF formulations," in *DATE*. IEEE, 2020, pp. 744–749.

[21] A. Shukla, A. Biere, L. Pulina, and M. Seidl, "A survey on applications of quantified boolean formulas," in *ICTAI*. IEEE, 2019, pp. 78–84.

[22] O. Beyersdorff, M. Janota, F. Lonsing, and M. Seidl, "Quantified boolean formulas," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, vol. 336, pp. 1177–1221.

[23] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *CP*, ser. LNCS, P. van Beek, Ed., vol. 3709. Springer, 2005, pp. 827–831.

[24] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multi-level networks," in *DAC*. IEEE Computer Society Press, 1990, pp. 297–301.

[25] M. Janota, "Towards generalization in QBF solving via machine learning," in *AAAI*. AAAI Press, 2018, pp. 6607–6614.

[26] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

[27] L. Amarú, P.-E. Gaillardon, and G. D. Micheli, "The EPFL combinational benchmark suite," in *International Workshop on Logic & Synthesis (IWLS)*, 2015. [Online]. Available: https://github.com/lsils/benchmarks

[28] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement," *Artif. Intell.*, vol. 234, pp. 1–25, 2016.

[29] H. Zhang, J. R. Jiang, and A. Mishchenko, "A circuit-based SAT solver for logic synthesis," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*. IEEE, 2021, pp. 1–6.