

# An Enhanced Resubstitution Algorithm for Area-Oriented Logic Optimization

Andrea Costamagna  
EPFL  
Lausanne, Switzerland

Alan Mishchenko  
U.C.Berkley  
Berkley, California

Satrajit Chatterjee  
Kepler AI

Giovanni De Micheli  
EPFL  
Lausanne, Switzerland

**Abstract**—*Logic synthesis* is an ensemble of algorithms that optimizes digital circuit representations and maps them to a chosen technology. Minimizing the number of gates is essential to reduce area occupation and power consumption. As the problem is intractable, heuristic logic transformations are used. In particular, *resubstitution* attempts to express the function of a node using other nodes already present in the network. State-of-the-art *resubstitution* engines can only identify new implementations with support of up to three inputs or being simply decomposable. This work aims at extending *resubstitution* to non-decomposable functions with more than three inputs and it outperforms previous methods. We apply our method on highly optimized designs from the ISCAS and EPFL benchmarks, achieving additional average improvements of 18.50% and 8.36%.

**Index Terms**—logic optimization, resubstitution, SPFD, XAIG.

## I. INTRODUCTION

THE operation of digital chips relies on switching millions of gates within the combinational circuits they contain. Each of these circuits is one of many possible implementations of complex Boolean functions. Finding implementations minimizing the gate count is an effective approach to obtain smaller chip sizes, fewer wires, and reduced power consumption.

Logic synthesis is an ensemble of algorithms that optimizes network representations before mapping to a specific technology library [1]–[3]. Reducing the number of nodes in the network representation is crucial to reduce the number of gates after mapping. As the problem is intractable [4], various logic heuristic transformations are used.

*Resubstitution* (also called substitution) is a very effective transformation that tries to express (*resynthesize*) the function of a node using a set of candidate nodes already present in the network [1]. The transformation is accepted if the new implementation reduces the node count. The exploited nodes are named *divisors* because resynthesis generally occurs through a decomposition presenting the mathematical structure of a *division* [5], [6].

There are two key sub-problems in *resubstitution*:

- 1) Select a support from a set of candidate divisors.
- 2) Resynthesize the function using this support.

State-of-the-art *resubstitution* engines address these problems by combining functional simulation, decomposition and SAT-solving [7]–[10]. These engines are highly effective when:

- 1) The number of support divisors is 3 or less.
- 2) The functionality of the node is simply decomposable.

Omitted for blind review

This work extends *resubstitution* to functions with more than three inputs and that are not simply decomposable. Our method relies on *set of pairs of functions to be distinguished* (SPFD), a powerful functional representation for investigating the dependency of different nodes [7], [11]–[13]. In this work:

- 1) We define an SPFD-based cost-function  $\mathcal{H}(x_i)$  for evaluating whether a divisor  $x_i$  should be selected.
- 2) We devise a support selection algorithm based on  $\mathcal{H}(\cdot)$ .
- 3) We devise a resynthesis algorithm based on  $\mathcal{H}(\cdot)$ .
- 4) We use these algorithms in a *resubstitution* engine.

The experimental results show that our approach enables *resubstitution* to achieve far better results than state-of-the-art engines. We apply our method on designs for which state-of-the-art *resubstitution* [10] could not find any further optimization. Our heuristic unlocks additional average improvements of 18.50% and 8.36% on the ISCAS and EPFL benchmarks.

## II. BACKGROUND

### A. The Boolean Network Representation

A *Boolean network* is a directed acyclic graph in which nodes represent logic gates, and edges represent wires. Xor-And Inverter Graphs (XAIGs) are Boolean networks in which nodes are two-inputs XORs or ANDs, and negated edges are inverters. XAIGs are efficient representations because a single node can encode any binary Boolean operation [4], [14].

If there is a path from a node  $x_i$  to a node  $x$ ,  $x_i$  is in the *transitive fanin* (TFI) of  $x$ . The primary inputs (PIs) are nodes without fanins in the network. The *maximum fanout free cone* (MFFC) of a node  $x$  is the set of nodes that are  $x$ , or a node in the TFI of  $x$  whose fanout is uniquely in the MFFC.

A *cut* of a node  $x$ , is a set of nodes, named *leaves* such that any path from a PI to  $x$  passes through one leaf. The *local function* of node  $x$  with respect to a cut is the Boolean function of the subcircuit identified by the cut. The *global function* of node  $x$  is the Boolean function of the sub-circuit defined from node  $x$  to the PIs. With an abuse of notation we refer to the *global function* of  $x$  as  $x : \mathbb{B}^n \mapsto \mathbb{B}$ .

### B. Sets of Pairs of Functions to be Distinguished

Let  $x : \mathbb{B}^n \mapsto \mathbb{B}$  be the *global function* of a node. A *minterm*  $M \in \mathbb{B}^n$  is a possible inputs assignment. Then,  $x$  induces a partition of  $\mathbb{B}^n$  in two subsets: the *onset* and the *offset*, such that  $x(M) = 1$  and  $x(M) = 0$ , respectively.

The *global sets of pairs of functions to be distinguished* (gSPFD) at node  $x$  is a function  $\Upsilon_x : \mathbb{B}^n \times \mathbb{B}^n \rightarrow \mathbb{B}$ :

$$\Upsilon_x(M_i, M_j) \doteq x(M_i) \oplus x(M_j) \quad \forall M_i, M_j \in \mathbb{B}^n \quad (1)$$

This function verifies if  $x$  *distinguishes*  $M_i$  and  $M_j$  because one is in the *onset* and the other is in the *offset* [7], [12], [13].

When exhaustively simulating a network is intractable, it is customary to select a subset of the input minterms  $\mathcal{M} \subseteq \mathbb{B}^n$ , perform the functional simulation of the network, and treat the resulting partial truth tables  $\tilde{x}$ , named *simulation signatures*, as an approximation of the *global functions*. The *approximate SPFD* (aSPFD)  $\Upsilon_{\tilde{x}}$  is obtained by evaluating Eq. 1 on  $\tilde{x}$  [15].

The *information graph* (IG) is a graphical representation of an SPFD where the vertices are the minterms, and the adjacency matrix is defined by  $\Upsilon_{\tilde{x}}$  [16]. The black graphs in Fig. 1 are the IGs of the nodes in a simple network.

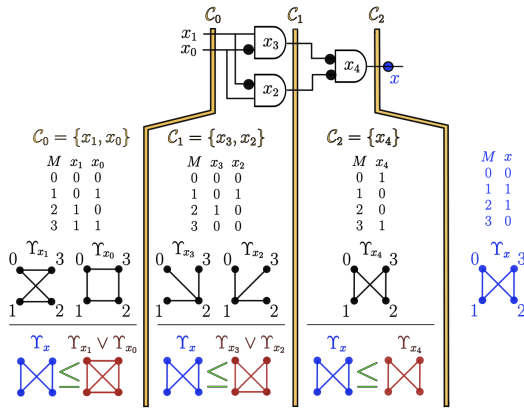


Fig. 1: From top to bottom: global functions at the cut nodes, their IGs, and the covering check of the target IG at each cut.

### C. Dependency and Graph Covering

A *dependency function* for node  $x$  and set  $\mathcal{C} = \{x_{c(l)}\}_{l=1}^k$ , is a function  $g : \mathbb{B}^k \rightarrow \mathbb{B}$  s.t.  $x = g(\mathcal{C})$  [17]. Given a node  $x$  and a set of variables  $\mathcal{C}$ , a necessary and sufficient condition for the existence of a *dependency function* is that,  $\forall M_i, M_j \in \mathbb{B}^n$ ,

$$x(M_i) \neq x(M_j) \Rightarrow \exists x_l \in \mathcal{C} \text{ s.t. } x_l(M_i) \neq x_l(M_j). \quad (2)$$

When  $\mathcal{C}$  is a *cut* of a node  $x$ , the *local function* constitutes a *dependency function*, and Eq. 2 is necessarily satisfied.

Using the definition of SPFDs (Eq. 1), we rewrite Eq. 2 as

$$\Upsilon_x(M_i, M_j) \leq \bigvee_{l=1}^k \Upsilon_{x_{c(l)}}(M_i, M_j) \quad (3)$$

Where  $a \leq b$  (or  $a \Rightarrow b$  [4]) is 0 only if  $a = 1$  and  $b = 0$ . Eq. 3 identifies a covering condition on the IGs: the IG of node  $x$  is covered by the union of the IGs of the nodes in  $\mathcal{C}$  for which there is a *dependency function*. Fig. 1 illustrates the satisfaction of Eq. 3 at the cuts  $\{x_0, x_1\}$ ,  $\{x_2, x_3\}$ , and  $\{x_4\}$ : the IG of  $x$  is covered by the union of the IGs at each cut.

### D. Resubstitution and Related Works Based on SPFDs

*Boolean resubstitution* (RS) is a logic minimization heuristic that can optimize Boolean networks in which:

- 1) There are nodes with constant global function.
- 2) There are nodes with the same function (up to negation).
- 3) There are nodes that can be resynthesized with a new set of variables reducing the node count of the network.

An important sub-problem to address in RS is the *support selection* problem, i.e., finding a new set of nodes to resynthesize a target node. Since a valid support must satisfy Eq. 3, several RS methods can be understood using this equation.

Originally, gSPFDs were computed exhaustively [18]. More recently, Zhang et al. [7] combined partial simulation and SAT-solving to reconstruct gSPFDs and identified valid supports from a set of candidates. Reconstructing the gSPFD with SAT-solving makes the method computationally intensive [15].

In logic rectification, Yang et al. [15] addressed this problem using aSPFDs. Nevertheless, their method relies on expensive heuristics that iterate through all the edges of the aSPFDs.

Goldberg et al. [19] proposed an RS-like algorithm for subcircuit replacement. However, they only use the initial subcircuit's inputs, restricting the method to local optimizations.

### E. The Simulation-Guided Paradigm For Resubstitution

Algorithm 1 shows the structure of state-of-the-art RS engines [8]–[10]. We name *urs* our implementation. First, *urs* simulates the network with  $|\mathcal{M}|$  simulation patterns. Next, for each node  $x$ , *urs* identifies a list of divisor  $\mathcal{D}$  from a structural exploration of the network, and uses unateness-based heuristics to select subsets of divisors as candidate supports for resynthesis. For each of the candidate supports, if the node can be resynthesized via decomposition and the transformation reduces the node count, a SAT-solver verifies whether the transformation can be applied preserving functional equivalence. This method effectively identifies resubstitutions with supports smaller than 4, or with simply decomposable functions, i.e., functions such that every node in the resynthesized circuit has at least a fanin that is a support variables.

---

#### Algorithm 1 $\langle \text{METHOD} \rangle_{rs}(\mathcal{N}; |\mathcal{M}|, N_{max})$

---

- 1: random\_simulation( $\mathcal{N}, |\mathcal{M}|$ )
  - 2: **for all**  $x \in \mathcal{N}$  **do**
  - 3:     **while** iteration <  $N_{max}$  **do**
  - 4:          $\mathcal{D} \leftarrow$  collect\_divisors()
  - 5:          $x_{new} \leftarrow$  resynthesis $\langle \text{METHOD} \rangle(x)$
  - 6:         **if** #(removed nodes) > #(new nodes) **then**
  - 7:             equivalent $\leftarrow$  SAT-CEC( $x_{new}, x$ )
  - 8:             **if** equivalent **then**
  - 9:                 **return** the subcircuit to perform optimization
  - 10:             **else**
  - 11:                 update\_simulation( $\mathcal{N}$ )
- 

In the next section, we describe a *support selection* and a *resynthesis* algorithm allowing us to go beyond the limitations of *urs*. By characterizing algorithm 1 with these techniques, we obtain a new RS-engine that we name *irs*.

### III. SPFD-BASED RESUBSTITUTION

#### A. Covering Processes and SPFD Representations

Given a node  $x$  and an ordered set of nodes  $\mathcal{C} = (x_{c(t)})_{t=1}^T$ , a *covering process* is the sequence  $\Upsilon_x^0 \rightarrow \Upsilon_x^1 \dots \Upsilon_x^t \dots \rightarrow \Upsilon_x^T$ , where  $\Upsilon_x^0 = \Upsilon_x$ , and at each step  $t$  we remove the edges covered by  $\Upsilon_{x_{c(t)}}$ . Fig 2 shows an example where we use simulation signatures of length  $|\mathcal{M}| = 5$ . We name  $\mathcal{H}(\Upsilon_x^t)$  the number of edges of  $\Upsilon_x^t$ . To implement this process, we

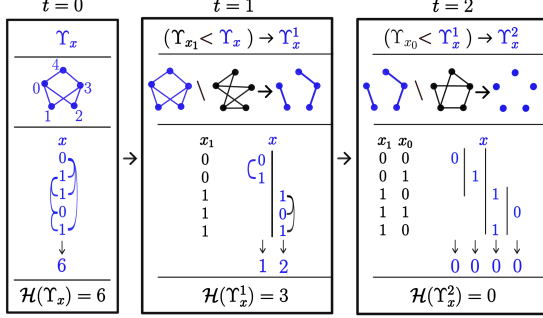


Fig. 2: From top to bottom: covering using IGs and simulation signatures, and number of edges  $\mathcal{H}(\Upsilon_x^t)$  after covering.

need an efficient representation for SPFDs.

Using the truth table of the function in Eq. 1 allows for a straightforward implementation, because covering is a bitwise operation  $\Upsilon_x^t = \Upsilon_{x_{c(t)}} < \Upsilon_x^{t-1}$ , and counting the edges is a popcount operation  $\mathcal{H}(\Upsilon_x^t) = \frac{\text{popcount}(\Upsilon_x^t)}{2}$ . However, the memory cost of this representation is quadratic in the number of minterms  $\mathcal{O}(|\mathcal{M}|^2)$ , limiting the size of the simulation signatures and, consequently, the expressiveness of aSPFDs.

An alternative approach is to perform covering using the  $|\mathcal{M}|$ -dimensional simulation signatures. Covering the graph with  $\Upsilon_{x_{c(t)}}$  corresponds to exploiting the information of  $x_{c(t)}$ , i.e., to separate the graph in two disconnected parts identified by  $x_{c(t)}$ . This amounts to partitioning the signature of  $x$  using the *onset* and *offset* of  $x_{c(t)}$ . The bottom part of Fig. 2 shows the partitioning of the signature. At each step,  $\mathcal{H}(\Upsilon_x^t)$  is the sum of the products of the number of 0s and 1s in each part. This representation requires a  $\mathcal{O}(2^T |\mathcal{M}|)$ -memory occupation.

The choice of a representation depends on  $T$  and  $|\mathcal{M}|$ .

#### B. The Support Selection Algorithm

Let  $x$  be a target node and  $\mathcal{D} = \{x_{d(i)}\}_{i=1}^D$  a set of candidate divisors. The *support selection problem* consists of finding a subset  $\mathcal{C} \subset \mathcal{D}$  satisfying Eq. 3. We define two algorithms, both relying on the definition of a *covering process*: at each step, we choose a divisor, we update the support, and we cover  $\Upsilon_x^t$ . We aim for sets  $\mathcal{C}$  satisfying Eq. 3 and having small sizes: small supports contain more informative variables, which are more likely to result in compact resynthesis sub-circuits.

The first approach we consider is a greedy strategy [20]. Let  $\Upsilon_x^t$  be the partially covered IG at iteration  $t$ , and  $\mathcal{H}(\Upsilon_{x_i} < \Upsilon_x^t)$  the number of remaining edges after covering  $\Upsilon_x^t$  with  $\Upsilon_{x_i}$ . In *greedy support selection* we choose the divisor  $x_i$  minimizing  $\mathcal{H}(\Upsilon_{x_i} < \Upsilon_x^t)$ , and we break ties at random.

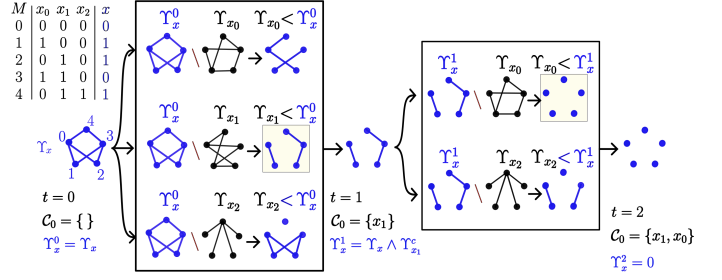


Fig. 3: Greedy support selection: At  $t = 1$  two  $x_0$  and  $x_1$  have the same cost  $\mathcal{H}(x_i) = 3$ . The tie is broken at random and  $x_1$  is selected. At  $t = 2$  the divisor  $x_0$  completes the covering.

Fig. 3 illustrates the greedy support selection using the 5-dimensional *simulation signatures* of three divisors: at each step  $t$ , the divisors are tested based on the IG covering they induce, until obtaining the support  $\mathcal{C}_0 = \{x_1, x_0\}$ .

The greedy heuristic does not guarantee finding the smallest size support or the best support for the resynthesis problem. To increase the capability of the algorithm to explore small size solutions, we define a statistical version of the algorithm. At each step  $t$  of the *covering process*, we define a probability distribution over the candidate divisors, and we sample a divisor from it. The probability distribution reads

$$p(x_i; \beta, t) \propto e^{-\beta \mathcal{H}(\Upsilon_{x_i} < \Upsilon_x^{t-1})} \quad \forall x_i \in \mathcal{D} \quad (4)$$

The divisors covering the largest number of edges have the highest probabilities. In the limit  $\beta \rightarrow \infty$ , the algorithm degenerates to the greedy approach: the only divisors with non zero probability are the ones with the lowest  $\mathcal{H}(\Upsilon_{x_i} < \Upsilon_x^{t-1})$ . As  $\beta$  decreases, the likelihood of selecting locally sub-optimal (but potentially globally optimal) divisors increases, until all divisors are equally likely in the limit  $\beta \rightarrow 0$ .

The flexibility of the statistical approach comes at the cost of an extra  $\mathcal{O}(D)$  runtime at each iteration for sampling.

#### C. SPFD-Based Cut-by-Cut Resynthesis

Let  $x$  be a target node, and  $\mathcal{C}_0$  a set of divisors satisfying Eq. 3, i.e., a support. The *resynthesis problem* consists of finding a sub-circuit taking the nodes in  $\mathcal{C}_0$  as the input, and having the output node with the same *global function* as  $x$ . We *synthesize* circuits one cut at the time. We aim for networks with a close-to-minimum number of nodes to maximize the optimization potential of resynthesis.

The set  $\mathcal{C}_0$  contains the inputs of a small network, or a set of divisors in a bigger circuit. At each level  $l$ , we add a new cut to the netlist, i.e., a set  $\mathcal{C}_l = \{x_{c_l(i)}\}_{i=1}^k$ . Following the discussion in Sec. II-B, each cut  $\mathcal{C}_l$  must satisfy Eq. 3, i.e.,  $\mathcal{C}_l$  must have enough information to resynthesize  $x$ .

Synthesizing a circuit one cut at the time has the advantage that, at each level  $l$ , there is a small number of candidate new nodes to add: each element of  $\mathcal{C}_l$  is either a wire pushing a variable in  $\mathcal{C}_{l-1}$  forward, or a binary Boolean operation from the set  $\{\bar{\vee}, <, >, \wedge, \oplus\}$  between two variables in  $\mathcal{C}_{l-1}$ . The enumeration of these candidates identifies a set  $\mathcal{D}_l$ , from which

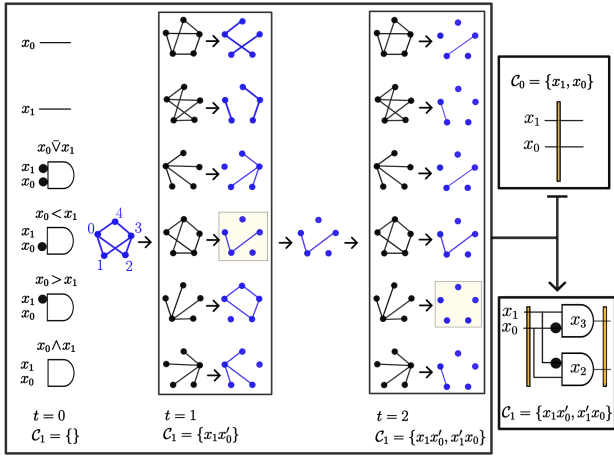


Fig. 4: Adding the cut  $\mathcal{C}_1$  to the circuit. From left to right: enumeration of the candidate nodes and selection of the cut.

we can find a cut  $\mathcal{C}_i$  by solving the *support selection* problem. We iterate the procedure until there is a cut of size 1. Fig. 4 shows the first step of the procedure for  $\mathcal{C}_0 = \{x_0, x_1\}$ . We remove the XOR to make the example non-trivial. Then,  $\mathcal{D}_1 = \{x_0, x_1, x_0 \bar{\vee} x_1, x_0 < x_1, x_0 > x_1, x_0 \wedge x_1\}$ , from which we obtain the cut  $\mathcal{C}_1 = \{x_1 < x_0, x_1 > x_0\}$ .

Compared to previously proposed cut-by-cut resynthesis methods [19], our approach is simpler, and its design space exploration capabilities are noteworthy in their own right.

#### IV. EXPERIMENTS

##### A. SPFD-Based Synthesis

The goal of this experiment is to show that, if provided with the support divisors, our resynthesis method can identify high-quality solutions in the design space. We target the Boolean function  $f: \mathbb{B}^5 \rightarrow \mathbb{B}$  with truth table 0x43B86C25, considered the hardest to synthesize 5 input function using decomposition. The minimum size XAIG for this function has 12 nodes. We used our engine to synthesize 10 circuits at 150 different calls (10 designs for each statistically selected first cut). We repeated the experiment varying  $\beta$ . The small size of the problem allows us to use the gSPFD instead of the aSPFD. Fig. 5 shows that we can find an optimum XAIG. Furthermore,  $\beta$  influences the time needed to find a good solution. We use this result to fix  $\beta = 100$  and we validated the effectiveness of the resynthesis engine through extensive experiments on other functions, that we do not report for space limitation reasons.

##### B. SPFD-Based Resubstitution

We show now the effectiveness of our technique by applying it to a set of standard benchmark circuits. In this experiment we show that *irs* can find optimization transformations not accessible to *urs*. First, we iteratively optimize industrial designs with *urs* until no further improvement can be found ( $\text{urs}^\infty$ ). Next, we apply one round of our strategy. We use the notation  $\text{irs}_{K,S,I}$ :  $K$  is the maximum allowed support size,  $S$  is the maximum number of supports we sample from the candidate divisors for optimizing a node, and  $I$  is the number

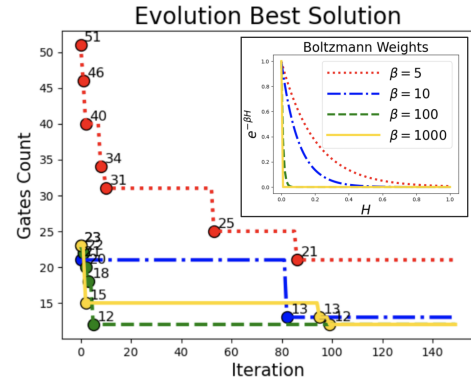


Fig. 5: Function 0x43B86C25: size of the best XAIG found during the exploration of the design space. Experiment used to set  $\beta = 100$  based on the convergence time to the optimum.

of calls to the statistical resynthesis engine for a given support. Increasing these parameters enables higher effort optimization.

We used  $N_{max} = 100$  for both *urs* and *irs*. Despite the high effort optimization of the state-of-the-art engine, *irs* unlocks additional improvements in most designs.

TABLE I: *irs* on the optimized ISCAS benchmarks.

Design	$\text{urs}^\infty$ [10]		$\text{urs}^\infty \rightarrow \text{irs}_{4,1,1}$		$\text{urs}^\infty \rightarrow \text{irs}_{7,10,10}$	
	XAIG	time[s]	XAIG	time[s]	XAIG	time[s]
c17	6	0.00	6	0.00	6	0.00
c432	166	0.01	166	0.01	166	0.15
c499	388	0.02	<b>246</b>	0.03	<b>214</b>	0.15
c880	296	0.02	<b>269</b>	0.02	<b>261</b>	0.27
c1355	420	0.03	<b>263</b>	0.03	<b>202</b>	0.14
c1908	280	0.03	<b>181</b>	0.02	<b>165</b>	0.15
c2670	532	0.06	<b>484</b>	0.04	<b>455</b>	0.65
c3540	787	0.35	<b>750</b>	0.08	<b>730</b>	1.40
c5315	1277	0.16	<b>1211</b>	0.10	<b>1168</b>	1.27
c6288	1480	0.25	<b>1426</b>	0.07	<b>1420</b>	0.48
c7552	1291	0.18	<b>1146</b>	0.11	<b>1039</b>	1.41
			-13.84%		-18.50%	

#### V. CONCLUSIONS

This paper proposes a new simulation-guided resubstitution procedure based on SPFDs, and shows that this procedure pushes the optimization capabilities of *resubstitution* beyond the limits of state-of-the-art engines on industrial-like designs.

#### REFERENCES

- [1] G. D. Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.
- [2] S. Chatterjee, *On algorithms for technology mapping*. University of California, Berkeley, 2007.
- [3] A. T. Calvino, H. Riener, S. Rai, A. Kumar, and G. De Micheli, "A versatile mapping approach for technology mapping and graph optimization," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 410–416.
- [4] D. E. Knuth, *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.
- [5] R. K. Brayton, "The decomposition and factorization of boolean expressions," *ISCAS-82*, pp. 49–54, 1982.
- [6] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

TABLE II: *irs* on the optimized EPFL benchmarks.

design	$urs^\infty$ [10]		$urs^\infty \rightarrow irs_{4,1,1}$		$urs^\infty \rightarrow irs_{7,10,10}$	
	XAIG	time[s]	XAIG	time[s]	XAIG	time[s]
adder	892	0.03	<b>653</b>	0.07	<b>643</b>	0.22
bar	2968	0.56	<b>2904</b>	0.35	<b>2840</b>	6.43
div	38942	34.97	<b>33304</b>	11.09	<b>32876</b>	20.81
hyp	205329	82.41	<b>171170</b>	114.97	<b>169450</b>	178.07
log2	29390	86.63	<b>26316</b>	33.75	<b>24858</b>	78.61
max	2862	0.21	2862	0.24	2862	3.64
multiplier	25403	7.41	<b>20341</b>	5.94	<b>19103</b>	27.40
sin	4929	9.58	<b>4578</b>	1.63	<b>4444</b>	7.82
sqrt	18450	25.22	<b>18015</b>	3.92	<b>16559</b>	19.58
square	16199	2.86	<b>14261</b>	1.79	<b>13460</b>	7.47
arbiter	11839	0.91	11839	0.90	11839	16.37
cavlc	591	2.30	591	0.17	<b>588</b>	2.78
ctrl	85	0.10	85	0.01	85	0.10
dec	304	0.01	304	0.01	304	0.01
i2c	1151	0.44	<b>1145</b>	0.10	<b>1145</b>	1.52
int2float	204	2.36	204	0.03	<b>202</b>	0.50
mem_ctrl	33579	83.56	<b>33303</b>	3.73	<b>32745</b>	48.34
priority	484	0.29	484	0.04	484	0.48
router	205	0.11	<b>181</b>	0.03	<b>177</b>	0.26
voter	7325	5.53	<b>6922</b>	0.72	<b>6416</b>	5.68
			<b>-6.52%</b>		<b>-8.65%</b>	

- [7] J. S. Zhang, S. Sinha, A. Mishchenko, R. K. Brayton, and M. Chrzanowska-Jeske, "Simulation and satisfiability in logic synthesis," *computing*, vol. 7, p. 14, 2005.
- [8] S.-Y. Lee and G. De Micheli, "Heuristic logic resynthesis algorithms at the core of peephole optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
- [9] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.
- [10] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2573–2586, 2021.
- [11] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method-design of logic networks based on permissible functions," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1404–1424, 1989.
- [12] R. K. Brayton, "Understanding spfds: A new method for specifying flexibility," in *Notes of International Workshop on Logic Synthesis (IWLS'97)*, May, 1997.
- [13] S. Sinha, *SPFDs: A new approach to flexibility in logic synthesis*. University of California, Berkeley, 2002.
- [14] I. Háleček, P. Fišer, and J. Schmidt, "Are xors in logic synthesis really necessary?" in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2017, pp. 134–139.
- [15] Y.-S. Yang, S. Sinha, A. Veneris, and R. K. Brayton, "Automating logic rectification by approximate spfds," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 402–407.
- [16] L. Józwiak, "Information relationships and measures: an analysis apparatus for efficient information system synthesis," in *EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No. 97TB100167)*. IEEE, 1997, pp. 13–23.
- [17] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*. Springer, 2004, pp. 268–280.
- [18] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean resubstitution with permissible functions and binary decision diagrams," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1991, pp. 284–289.
- [19] E. Goldberg, K. Gulati, and S. Khatri, "Toggle equivalence preserving (tep) logic optimization," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*. IEEE, 2007, pp. 271–279.
- [20] V. V. Vazirani, *Approximation algorithms*. Springer, 2001, vol. 1.