CellE: Automated Standard Cell Library Extension via Equality Saturation

Yi Ren[1,2,†], Yukun Wang[3], Xiang Meng[3], Guoyao Cheng[4], Baokang Peng[4]
Lining Zhang[4,5], Yibo Lin[1,5,6], Runsheng Wang[1,5,6], Guangyu Sun[1,5,6,*]
[1]School of Integrated Circuits, [2]School of Software and Microelectronics, Peking University, Beijing, China
[3]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
[4]School of Electronic and Computer Engineering, Peking University, Shenzhen, China
[5]Institute of Electronic Design Automation, Peking University, Wuxi, China
[6]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China

Abstract

Automated standard cell library extension is crucial for maximizing Quality of Results (QoR) in modern VLSI design. We introduce CellE, a novel framework that leverages formal methods to achieve exhaustive discovery of functionally equivalent subcircuits. CellE applies equality saturation to the post-mapping netlist, generating an e-graph to cluster all functionally equivalent implementations. This canonical representation enables an efficient pattern mining algorithm to select the most area-optimal standard cells. Experimental results show a 15.41% average area reduction (up to 23.64% over prior work). Furthermore, characterization in a commercial flow demonstrates an 8.00% average delay reduction, confirming CellE's superior QoR optimization capabilities.

CCS Concepts

• **Hardware → Standard cell libraries**; **Combinational circuits**; • **Mathematics of computing → Hypergraphs**; **Graph algorithms**.

Keywords

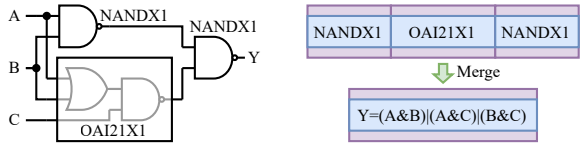
Standard cell library extension, Equality saturation, Frequent subgraph mining, Design technology co-optimization

ACM Reference Format:
Yi Ren, Yukun Wang, Xiang Meng, Guoyao Cheng, Baokang Peng, Lining Zhang, Yibo Lin, Runsheng Wang, and Guangyu Sun. 2026. CellE: Automated Standard Cell Library Extension via Equality Saturation. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3803942

This work is supported in part by New Generation Artificial Intelligence-National Science and Technology Major Project (2025ZD0122105), Beijing Natural Science Foundation (L243001), National Natural Science Foundation of China (62125401), National Key Research and Development Program of China (2021ZD0114702), Beijing Outstanding Young Scientist Program (JWZQ20240101004) and 111 Project (B18001).
[†]First author, e-mail: yiren20@stu.pku.edu.cn.
[*]Corresponding author, e-mail: gsun@pku.edu.cn.

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3803942



(a) A frequent sub-circuit in adder. (b) Area reduction by merging.

Figure 1: An example of reducing area by merging subcircuit.

1 Introduction

As integrated circuit manufacturing scales into deep submicron nodes, particularly at 22 nm and beyond, rising costs and physical constraints necessitate Design–Technology Co-Optimization (DTCO) [20, 26, 42], where the standard cell library serves as the fundamental building block determining final circuit quality. The manual and time-consuming nature of cell layout design, however, becomes a bottleneck in DTCO iteration, driving the need for automated layout generation [2, 4, 7, 10, 12–14, 16, 17, 21, 23, 27, 29, 34–36, 39, 44]. To address process non-idealities and complex circuit demands, Standard Cell Library Extension (SCLX) has emerged as a key DTCO enabler. As shown in the simplified example in Figure 1, by merging multiple basic cells into composite units, SCLX reduces gate-level redundancy and wiring congestion, thereby improving Quality of Results (QoR) [6, 8, 11, 18, 19]. Thus, developing an efficient and automated SCLX framework is essential for modern VLSI design automation.

Existing automated SCLX methods remain fundamentally constrained by their search space. Frameworks such as AutoCellLibX [19] and TeMACLE [11] identify candidate cells via frequent subgraph mining on a post-mapping netlist, e.g., extracting K-feasible cones and verifying equivalence using SAT. However, these approaches operate only on a single post-mapping netlist from logic synthesis, restricting discovery to one topological realization of the Boolean function. This structural dependence subjects SCLX to a "phase-ordering problem" [15]: if a more efficient subcircuit is absent from the initial netlist, it remains inaccessible to subsequent mining.

To overcome SCLX's dependence on a fixed netlist, a paradigm shift toward systematic exploration of logical equivalence is needed. *Equality saturation*, based on the *e-graph* (equality graph) data structure, offers such a solution [32]. The e-graph employs congruence closure to compactly encode equivalent expression structures through E-Classes [25]. During saturation, rewrite rules expand the graph to represent all equivalent implementations, resolving the phase-ordering problem by deferring optimization until extraction [38]. This approach has demonstrated effectiveness in compiler optimization [43], logic synthesis [5], and symbolic reasoning [41].

Integrating equality saturation into SCLX shifts the search from structural to semantic, enabling discovery of optimal cell patterns across logically equivalent netlists.

Based on the superior optimization capabilities of Equality Saturation, this paper proposes CellE: Automated Standard Cell Library Extension via Equality Saturation. As the first framework leveraging equality saturation for systematic standard cell enhancement, CellE introduces a semantic-driven methodology. Initially, equality saturation is applied to the post-mapping netlist, constructing an e-graph that compactly encodes all logically equivalent implementations. A modified subcircuit mining algorithm then operates on this e-graph, enabling discovery of high-frequency patterns across the full equivalence space. This approach reduces reliance on initial technology mapping decisions, thereby generating better cells to enhance overall circuit QoR. The main contributions of CellE are:

- The first SCLX framework to resolve the phase-ordering problem via equality saturation, enabling semantic exploration across massive logically equivalent netlists.
- A novel subcircuit mining algorithm on e-graph that extracts optimal composite cells from compressed equivalence classes, unlocking patterns inaccessible to traditional netlist traversal.
- Empirical demonstration of significant area optimization, achieving a 15.41% average area reduction and up to 23.64% reduction compared to state-of-the-art tools.
- Confirmed effectiveness in a commercial flow, showing an average 8.00% delay reduction and 1.27% area reduction.

The remainder of this paper is organized as follows. Section 2 discusses the preliminaries, including the problem formulation of SCLX and the introduction of equality saturation. Section 3 introduces the proposed SCLX framework. Section 4 details the experiment setting and evaluation of CellE, and concludes in Section 5.

2 Preliminaries

This section is divided into two parts. First, we formalize the key concepts, including netlists and subcircuits, and present the Standard Cell Library Extension (SCLX) problem. Next, we introduce the relevant concepts and definitions pertaining to equality saturation.

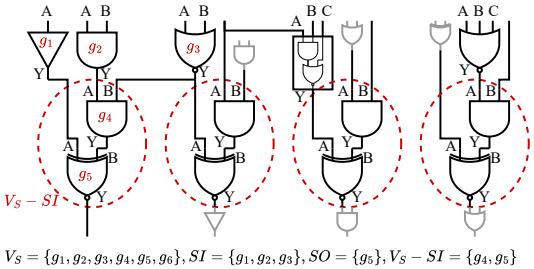
2.1 Problem Formulation

2.1.1 Technology Mapping. Technology mapping [3, 9, 30, 33], denoted by $\mathcal{M}$, is a graph covering problem transforming the technology independent Boolean network B into an optimal, technology specific netlist G. The transformation is constrained by the Standard Cell Library LIB, a set of characterized physical resources. $\mathcal{M}$ is a complex optimization designed to minimize the Quality of Results (QoR) cost, $Q(G)$. This requires G to be functionally equivalent to B while containing only cells from LIB. Formally:

$$\mathcal{M}_{LIB}(B) = \arg\min_{G}\{Q(G) \mid G \sim LIB, \text{Func}(G) \equiv \text{Func}(B)\}. \quad (1)$$

2.1.2 Post-mapping Netlist. The post-mapping netlist of a combinational circuit can be represented as a directed acyclic graph (DAG) $G(V, E, L)$ with a vertex set V, a directed edge set $E \subseteq V \times V$, a label mapping L, and no cycle. The definitions are shown below:

- Vertices V, also denoted as V_G, represent logic gates in the netlist, i.e., any vertex $v \in V$ is an instance of standard cell.



$V_S = \{g_1, g_2, g_3, g_4, g_5, g_6\}, SI = \{g_1, g_2, g_3\}, SO = \{g_5\}, V_S - SI = \{g_4, g_5\}$

Figure 2: Example of a frequent subcircuit.

- Edges E, also denoted as E_G, represent nets from driven pins to sink pins in the netlist, i.e., any edge $e = (u, v) \in E$ represents gate u drives gate v.
- Label mapping L, represents the labels of vertices and edges. Specifically, vertex label $L(v)$ means the standard cell type of v, while edge label $L(e) = L(u, v) = (O_u, I_v)$ for edge $e = (u, v)$ means the output pin named O_u of gate u drives the input pin named I_v of gate v.
- Primary inputs $PI \in V$ are defined as vertices with no incoming edge, while primary outputs $PO \in V$ are defined as vertices with no outgoing edge. Specifically, PI and PO of a netlist are considered as specific vertices with labels `input` and `output`, with pins `O` and `I`, respectively.

2.1.3 Subcircuits. Subcircuit S is a subgraph of netlist graph G. The input vertices and output vertices of S are denoted by SI and SO. The size of a subcircuit $|S|$ is defined as the number of vertices except primary inputs $|V_S - SI|$. The left most ellipse in Figure 2 shows a simple example of a subcircuit. Here, $V_S = \{g_1, g_2, \ldots, g_6\}$, $SI = \{g_1, g_2, g_3\}$, $SO = \{g_5\}$. The main body of the subcircuit is $V_S - SI = \{g_4, g_5\}$ and hence $|S| = 2$. Moreover, a valid subcircuit must satisfy the following constraints:

- *Single-output.* The same as TeMACLE [11], in this work we only consider subcircuits with exactly one output, i.e., $|SO| = 1$.
- *Connectivity.* All gates in S should be connected.
- *Source-sink.* Any input pin of gates $v \notin SI$ should be driven by one output pin of another gate u. Any output pin of gates $v \notin SO$ should drive at least one input pin of another gate u.

We define *functional equivalent* of two subcircuits as follows:

Definition 2.1 (functional equivalent). Consider two subcircuits S_1, S_2 with output vertices $SO_1 = \{z_1\}, SO_2 = \{z_2\}$ and input vertices $SI_1 = \{x_1, x_2, \ldots, x_m\}, SI_2 = \{y_1, y_2, \ldots, y_n\}$. Using boolean algebra, we can get boolean formulas for z_1, z_2, i.e., $z_1 = f_1(SI_1) = f_1(x_1, x_2, \ldots, x_m)$, $z_2 = f_2(SI_2) = f_2(y_1, y_2, \ldots, y_n)$. S_1 and S_2 is *functional equivalent*, denoted by $S_1 \equiv S_2$, if and only if $m = n$ and there exists a bijection $g : SI_1 \xrightarrow{\sim} SI_2$ such that $f_1(SI_1) \equiv f_2(g(SI_1))$ for any boolean assignment $SI_1 \in \{\textbf{True}, \textbf{False}\}^m$.

2.1.4 Standard Cell Library Extension (SCLX). We further define patterns to conveniently describe the SCLX problem. A *pattern group* $PG = \{S_1, S_2, \ldots, S_p\}$ is a set of p functional equivalent subcircuits, i.e., $\forall S_i, S_j \in PG, S_i \equiv S_j$. Figure 2 shows an example of pattern group with size 4. Each pattern group PG can be generated into a standard cell c with tools like ASTRAN [44], AutoCellGen [2]. We denote the corresponding pattern group of c as PG_c. Then, we have the following SCLX problem:

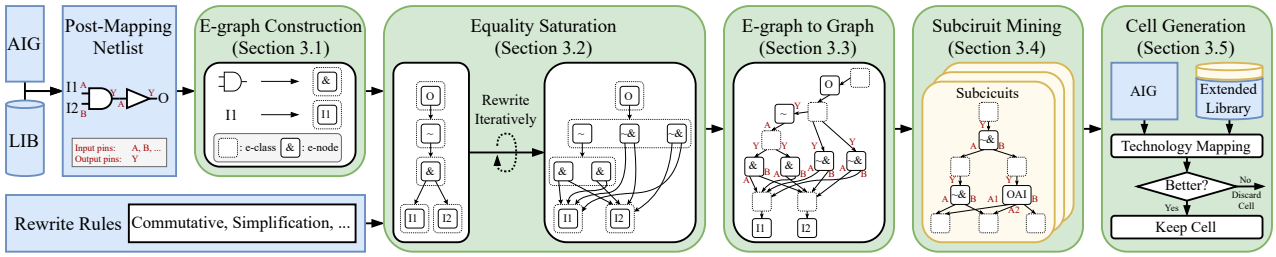


Figure 3: The workflow of proposed CellE.

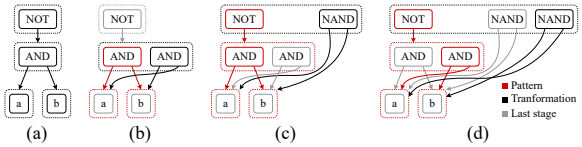


Figure 4: Simple e-graph rewriting example.

Inputs:

1) A given Boolean network B.
2) An original standard cell library LIB.
3) A technology mapping tool $\mathcal{M}$.

Outputs: An extended standard cell library $LIB' = LIB \cup EX$, where $EX = \{c_1, c_2, \ldots, c_t\}$ is the set of extended cells.

Constraints:

1) At most T new cells can be added to the extended library LIB', i.e., $|EX| = t \leq T$.
2) The number of original cells in any subcircuit is at most N, i.e., $\forall c_i \in EX, \forall S_j \in PG_{c_i}, |S_j| \leq N$.
3) Any cell has at most K input pins, i.e., $\forall c_i, \forall S_j \in PG_{c_i}, |SI| \leq K$.
4) Any cell has one output pin, i.e., $\forall c_i, \forall S_j \in PG_{c_i}, |SO| = 1$.

Goal: Minimize the QoR cost function $Q(\mathcal{M}_{LIB'}(B))$ of the final mapped netlist $\mathcal{M}_{LIB'}(B)$: $\min_{LIB'} Q(\mathcal{M}_{LIB'}(B))$.

2.2 Equality Saturation

Equality saturation [24, 25, 32, 38, 41] is a rewriting optimization technique powered by an e-graph (equivalence graph) data structure, which represents an equivalence relation over expressions.

2.2.1 E-graph. An *e-graph* $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{N}, \mathcal{C}, \mathcal{L})$, where:

- $\mathcal{N}$ is a set of vertices, called *e-nodes*, where each e-node represents a distinct expression or sub-expression.
- $\mathcal{C} = \{C_1, C_2, \ldots, C_p\}$ is a partition of $\mathcal{N}$ into p sets. Each e-node set C_i represents an equivalence class known as an *e-class*. E-nodes within the same e-class are considered equivalent.
- $\mathcal{L} : \mathcal{N} \to \Sigma \times \mathcal{C}^k$, called *language*, is a labeling function that assigns each e-node to an operator and an ordered list of child e-classes, where Σ represents the set of operators with k operands and $\mathcal{C}^k$ denotes an ordered tuple of k child e-classes from $\mathcal{C}$.

Figure 4(a) shows a simple e-graph example of NOT(AND(a, b)), where the arrows (from left to right) originating from each e-node represent the ordered child e-classes.

2.2.2 Rewrite rules. Rewrite rules $\mathcal{R}$ is a set of rules to modify the e-graph. Each rule is a pair of patterns (l, r), also know as $l \to r$, which means the left pattern l is equivalent to the right pattern r. Each pattern is a sub-expression like NOT(x), AND(x, y) and NAND(x, y). To apply a rule, new e-nodes may be added and some e-classes may be extended or merged to represent new equivalence.

Algorithm 1: E-graph Construction

Input : Post-mapping netlist G
Output: E-graph $\mathcal{G}$

```
Initialize empty e-class mapping vmap and e-graph G;
order ← TopoSort(G);                    // From inputs to outputs
for each vertex v in order do
    inputs ← [ ];                       // Collect input e-classes
    for each input i of vertex v do
        inputs.push(vmap[i]);           // Same pin order
    cid ← G.insert(v, inputs);          // Add e-node
    vmap[v] ← cid;
```

Figure 4(b-d) shows the rewriting procedure of rules:

$$\text{AND}(x, y) \to \text{AND}(y, x),\ \text{NOT}(\text{AND}(x, y)) \to \text{NAND}(x, y). \quad (2)$$

In each step, the gray part remains unchanged, the red pattern is matched and the black transformation is applied.

3 CellE

This section proposes an automated Standard Cell Library Extension (SCLX) framework via equality saturation, namely CellE. The overall workflow is illustrated in Figure 3. The framework accepts a Boolean network as input, such as an And-Invertor Graph (AIG). Initially, CellE maps the Boolean network into a post-mapping netlist using given technology mapping tools. Subsequently, CellE converts the post-mapping netlist into an e-graph (Section 3.1). Then, the rewrite engine of CellE applies rewriting rules to expand the e-graph until it is saturated (Section 3.2). Next, CellE converts the e-graph into a normal graph version (Section 3.3), preparing for subcircuit mining with graph mining algorithm (Section 3.4). Finally, CellE generates new standard cells for mined subcircuits and minimizes the QoR cost function of the post-mapping netlist generated with the extended standard cell library (Section 3.5).

3.1 E-graph Construction

After mapping the Boolean network into a post-mapping netlist G, CellE constructs an e-graph $\mathcal{G}$ in preparation for subsequent equality saturation. The algorithm for egraph construction is detailed in Algorithm 1. CellE first computes a topological order of the netlist vertices, from primary inputs PI to primary outputs PO. It then iterates through the vertices in this order. For each vertex v, it gathers the e-class IDs of its inputs using a mapping $vmap$. The input order for each e-node corresponds to the pin order for each gate. Finally, it adds a new e-node representing v and its inputs to the e-graph $\mathcal{G}$, and updates $vmap$ with the resulting e-class ID. The complexity is $O(|V_G| + |E_G|)$.

Table 1: Example of Rewrite Rules.

Name	Pattern (LHS)	Transformation (RHS)
Commutativity	AND(a,b) XOR(a,b)	AND(b,a) XOR(b,a)
De Morgan's	OR(NOT(a), NOT(b)) AND(NOT(a), NOT(b))	NAND(a,b) NOR(a,b)
Simplification	NOT(AND(a,b)) NOT(XOR(a,b))	NAND(a,b) XNOR(a,b)
Involution	NOT(NOT(a))	a

Algorithm 2: E-graph to Normal Graph

Input : E-graph $\mathcal{G} = (\mathcal{N}, C, \mathcal{L})$, library pins $(\mathcal{I}, \mathcal{O})$
Output: Graph $G = (N, C, E)$

Initialize empty graph G;
Create vertex mapping $M_{\mathcal{N}}, M_C$ for $\mathcal{N}$ and C, respectively;
for *e-class* $C \in C$ **do**
  for *e-node* $n \in C$ *with output pin* $\mathcal{O}(n)$ **do**
    Add edge: $M_C[C] \rightarrow M_{\mathcal{N}}[n]$ with label $\mathcal{O}(n)$;
for *e-node* $n \in \mathcal{N}$ **do**
  for *e-class and pin* $(C, I) \in zip(\mathcal{L}(n), \mathcal{I}(n))$ **do**
    Add edge: $M_{\mathcal{N}}[n] \rightarrow M_C[C]$ with label I;

3.2 Equality Saturation

The rewriting procedure utilizes equality saturation to exhaustively explore equivalent circuit representations. This iterative process repeatedly applies a predefined set of rewrite rules, such as the examples shown in Table 1, to the e-graph $\mathcal{G}$. When a rule's left-hand side (pattern) is matched within the e-graph, the corresponding right-hand side (transformation) is added to the same e-class, effectively asserting their equivalence. CellE continues this application until saturation is reached, meaning no new equivalent structures can be added. The resulting saturated e-graph $\mathcal{G}'$ thus compactly encodes a vast space of functionally equivalent circuits. The complexity is $O(r|\mathcal{N}|^k)$ with r rules and maximal pattern size k.

3.3 E-graph to Graph Conversion

CellE notes that the e-graph $\mathcal{G}$ is an unusual data structure, primarily designed for equivalence management rather than direct graph mining. Specifically, it lacks explicit gate pin information and represents alternatives compactly, making it unsuitable for direct application of subcircuit mining algorithms. Therefore, CellE introduces Algorithm 2 to transform the saturated E-graph $\mathcal{G} = (\mathcal{N}, C, \mathcal{L})$ into a standard graph G suitable for mining. The algorithm initializes G and creates corresponding vertices for all e-nodes $\mathcal{N}$ and e-classes C, recording these mappings as $M_{\mathcal{N}}$ and M_C, respectively. The conversion proceeds in two main stages, incorporating the library pin information (inputs $\mathcal{I}$ and outputs $\mathcal{O}$). First, for every e-class C, directed edges are added from the vertex representing C (via M_C) to all contained e-nodes n (via $M_{\mathcal{N}}$), labeled with the corresponding output pin $\mathcal{O}(n)$. Second, for every e-node n, directed edges are added from the vertex representing n via $(M_{\mathcal{N}})$ to the vertices representing its child e-classes C (via M_C), labeled with the respective input pin I (obtained from the language $\mathcal{L}(n)$ and $\mathcal{I}(n)$). This yields a bipartite graph structure G with explicit connectivity and pin labeling. Figure 5 shows an example of converted graph. The complexity is $O(|V_G| + |E_G|)$.

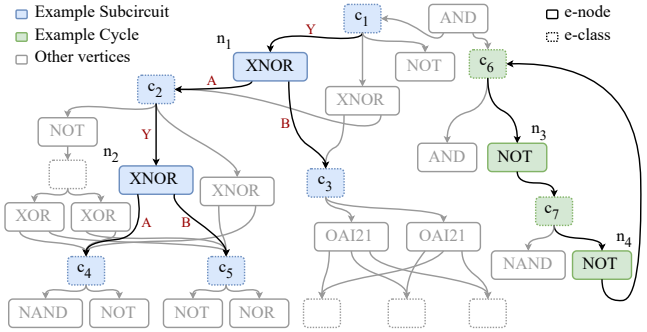


Figure 5: Example of graph version e-graph.

Algorithm 3: Pattern Groups Collection

Input : Graph G, MinSupport θ_{min}
Output: Frequent Patterns $\mathcal{F}$

$\mathcal{F} \leftarrow \emptyset$;
$E_{freq} \leftarrow$ `FindFrequent1EdgeWithProjections`(G, θ_{min});
for $(code, P) \in E_{freq}$ **do**
  `SubgraphMining`$(G, \theta_{min}, \mathcal{F}, code, P)$;
return $\mathcal{F}$;
Function `SubgraphMining`$(G, \theta_{min}, \mathcal{F}, code, P)$
  if *(not* `IsMinimal`$(code)$*) or* `IsIllegal`$(code)$ **then**
    return ; `// Prune non-canonical or illegal paths`
  if `SatisfiesConstraints`$(code)$ **then**
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(code, P)\}$;
  $Exts \leftarrow$ `FindExtensions`$(G, \theta_{min}, code, P)$;
  for $(code', P') \in Exts$ **do**
    `SubgraphMining`$(G, \theta_{min}, \mathcal{F}, code', P')$;

3.4 Subcircuit Mining

Following the conversion from the e-graph, CellE obtains a normal, bipartite graph G that explicitly represents both gates (e-nodes) and equivalence points (e-classes) with explicit pin labeling. To identify frequently occurring and valuable subcircuits (or pattern groups), CellE employs a subcircuit mining phase. This phase uses a frequent subgraph mining algorithm, adapted from the gSpan [40] method, which is renowned for discovering canonical subgraph patterns in general graphs (even with cycles). Note that the saturated e-graph $\mathcal{G}$ may contain cycles and thus the converted graph G (e.g., green parts in Figure 5) also may. Therefore, mining methods designed for DAGs, like K-feasible cones in TeMACLE [11] are not suitable.

3.4.1 Subgraph Constraints. It is important to note that in this bipartite graph $G(N, C, E, L)$—where N is the e-node set, C is the e-class set, E is the directed edge set, and L is the label mapping—the definition of a subcircuit is different from that in Section 2.1.3. A subcircuit is a subgraph S that satisfies the following constraints:

- ***Completeness.*** All input SI and output SO vertices must be e-class vertices, to ensure that all pins of each gate are included in the subgraph, i.e., $SI, SO \subseteq C_S$.
- ***Uniqueness.*** Only one e-node can be chosen in any e-class, i.e., any e-class $c \in C_S - SO$ must have one outgoing edge.
- *Single-output.* The same as TeMACLE [11], S has exactly one output vertex, i.e., $|SO| = 1$.
- *Connectivity.* All vertices in S should be connected.

Table 2: Experimental results on the EPFL benchmark [1] and FreePDK45 library [31].

Circuit	Original			TeMACLE [11]				CellE					
	gates	area (μm^2)	depth	area (μm^2)	reduction (%)	depth	size	e-nodes	area (μm^2)	reduction (%)	depth	time (s)	size
adder128	768	2373.25	193	1884.15	20.61	129	3/5	2037	1438.80	39.37	128	6.04	2/3
bar	2283	5426.52	10	4607.51	15.09	9	3/2/2/4	6521	4483.56	17.38	9	18.90	2/3/3/2/3
cavlc	532	1232.38	11	1187.73	3.62	10	2/2/2/2/2	1721	1186.28	3.74	11	0.21	2/2/2/4/4
div	42220	109675.88	2213	86576.50	21.06	2200	2/2/2/2/3	100550	84004.44	23.41	2190	444.13	2/2/3
hyp	187027	505059.72	10749	415107.84	17.81	10733	2/2/2/2/2	485906	413745.44	18.08	10695	136.75	2/2/2/4/2
i2c	1094	2461.95	11	2368.51	3.80	11	2/2/3/2/3	3796	2327.08	5.48	11	0.77	2/2/4/2/2
int2float	182	420.49	9	407.74	3.03	9	2/2/3	635	402.35	4.31	9	0.37	2/2/2/2/2
log2	21827	58706.61	232	52795.23	10.07	221	2/2/2/2/2	65421	52868.23	9.95	221	27.86	2/2/2/2/2
max	2694	6006.57	178	5002.03	16.72	176	2/2/3/2/2	6092	4901.63	18.40	177	12.77	2/2/2/3/3
mem_ctrl	36517	81936.96	74	73855.93	9.86	74	2/2/2/3/2	113405	79494.13	2.98	74	31.25	2/2/2/2/2
multiplier	16684	48909.51	179	43687.63	10.68	178	2/2/2/3/3	38048	43483.38	11.09	178	10.54	2/2/2/2/2
priority	1072	2344.15	64	1691.72	27.83	64	2/2/3/2	2751	2133.25	9.00	64	20.37	2/2/2
router	240	569.73	25	542.09	4.85	25	2/2/3/2/2	821	540.23	5.18	25	2.94	2/2/2/2
sin	4145	10882.60	116	9609.23	11.70	113	2/2/2/2/3	12603	9285.37	14.68	113	4.97	2/2/2/2/2
sqrt	18140	47752.68	3039	32940.58	31.02	3038	2/2/2/5/4	55225	33461.75	29.93	3039	944.94	2/2/2/2
square	13895	37336.10	167	33215.22	11.04	167	2/2/2/4/2	35789	30100.20	19.38	125	1.68	2/2/3/2/2
voter	9973	28922.49	38	23512.72	18.70	33	2/2/2/3/2	25064	23311.05	19.40	32	2.75	2/2/3/4
gmean	-	11425.74	114.36	9785.17	14.36	108.82	-	-	9665.16	15.41	107.33	-	-

- *Source-sink.* All pins of a gate must either drive another gate or be driven by another gate, i.e., any e-node $n \in N_S$ has exactly one incoming edge and all its outgoing edges are within E_S.

The blue part in Figure 5 shows an example of a subcircuit S with e-class inputs $SI = \{c_3, c_4, c_5\}$ and outputs $SO = \{c_1\}$.

3.4.2 Subcircuit Encoding. To efficiently manage and compare subgraphs, CellE uses a canonical representation based on a Depth-First Search (DFS) code, as inspired by gSpan [40]. A subgraph pattern is encoded as a `DFSCode`, which is defined as a sequence of `DFSEdges`. Each `DFSEdge` is a 5-tuple (i, j, L_i, L_{ij}, L_j), representing an edge from pattern vertex i to pattern vertex j, with node labels L_i and L_j (either an e-class or an e-node with its cell type) and an edge label L_{ij} (the pin name).

The sequence of edges in the `DFSCode` corresponds to a specific DFS traversal of the subgraph. By using a *DFS lexicographic order* on these sequences of 5-tuples, CellE can generate a unique, minimal `DFSCode` for any given subgraph topology. This canonical encoding is critical, as it allows for direct comparison of patterns and pruning of redundant search paths. A *pattern group* is thus defined not just by its canonical `DFSCode` (the pattern's structure), but also by its associated set of *projections*—a list of all mappings from the pattern's nodes to the nodes in the main graph G where that specific pattern occurs. Each projection corresponds to a subcircuit. The size of this projection set is the pattern's *support*.

3.4.3 Pattern Groups Collection. The collection of frequent pattern groups is performed using a depth-first search on the lattice of possible subgraphs, as detailed in Algorithm 3. The process begins by identifying all single-edge patterns (1-edge `DFSCodes`) in the graph G that meet a minimum support threshold, θ_{min} (line 2). These initial patterns, along with their projection lists, form the starting point for the recursive mining process.

The core of the algorithm is the recursive `SubgraphMining` function. This function takes a `DFSCode` and its list of projections P. First, it checks if the `DFSCode` is in its minimal (line 7-8) canonical form using DFS lexicographical ordering to avoid redundant exploration of the same subgraph discovered via different traversal paths. Next, CellE checks if the pattern satisfies structural constraints, such as the maximum size, the number of inputs and constraints from Section 3.4.1 (line 7-10). Valid and frequent patterns are saved to set $\mathcal{F}$. The function then recursively attempts to extend the current pattern by one edge (line 11-13). This process continues until no further frequent extensions can be found. The complexity is $O(|E_G| \times |V_G|^N)$ where N is the maximal subcircuit size.

3.5 Standard Cell Generation

The final stage involves generating and selecting new standard cells from the frequent pattern groups $\mathcal{F}$ identified in Section 3.4. Each pattern group is treated as a candidate standard cell. Following the approach of TeMACLE [11], CellE first calculates each cell's minimized function expression using the Quine-McCluskey algorithm [22, 28]. Subsequently, CellE uses SAT to filter functional equivalent subcircuits. Afterward, CellE generates SPICE netlists and employs automated tools, such as ASTRAN [44] and AutoCellGen [2], to perform standard cell generation. Since it is impractical to generate and include cells for all mined patterns into the library, a selection phase is required. CellE evaluates the utility of the candidate cells by performing technology mapping for the original Boolean network using the original library augmented with the new candidates. The primary objective is to identify the subset of candidate cells that minimizes a predefined QoR cost function. This selected set of high-value cells forms the final extended standard cell library, which is the ultimate output of the CellE framework.

4 Experimental Results

The experiments were conducted on a system with a Dual-Socket AMD EPYC 9754 128-Core Processor @ 1.50GHz and 1.5 TiB of memory. CellE is implemented using Rust language for cell finding (the translation between graphs and e-graphs, equality saturation and subcircuit mining), and using Python language for the cell generation. The *egg* [38] framework was used for equality saturation. For fair area comparison, the experiment chose the same setup as TeMACLE [11], which were the EPFL benchmark [1], the synthesis tool mockturtle [30], the standard cell layout generation tool ASTRAN [44] and the original standard cell library FreePDK45 [31] with standard cells AND2X2, AOI21X1, BUFX2, INVX1, NAND2X1,

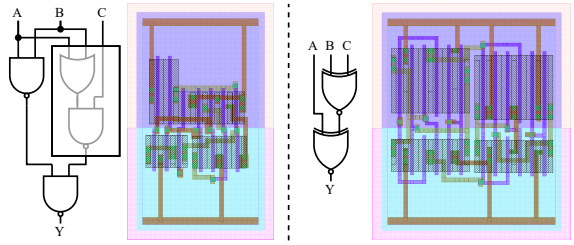


Figure 6: New standard cells generated for "adder128".

NAND3X1, NOR2X1, NOR3X1, OAI21X1, OR2X2, XNOR2X1, and XOR2X1. Following the setup of TeMACLE, the maximal number of extended standard cells T was set to 5. The maximal number of original cells in any new cell N was set to 5. The maximal number of inputs for any new cell K was set to 3. For rewrite rules, CellE chose 14 commulative, 4 De Morgan's, 6 simplication, and 1 involution.

4.1 Area Comparison

Table 2 details the area results, comparing the Original baseline library mapping against the extended libraries from TeMACLE and CellE. The table lists the final area, percentage area reduction (reduction (%)), depth, and new cell composition with cell sizes (size). For CellE, the number of e-nodes (e-nodes) and the total runtime (time (s)) of its subcircuit finding stages are also listed.

On average, CellE achieves a 15.41% geometric mean (gmean) area reduction over the original library, surpassing TeMACLE's 14.36%. This highlights the effectiveness of our equality-saturation-based approach. The benefit is pronounced on specific circuits; for instance, on "adder128", CellE achieves a 23.64% further area reduction compared to TeMACLE's result. For a few other circuits, the improvements are smaller because their logic exposes fewer semantic alternatives, while TeMACLE's additional iterative remapping can occasionally surface patterns that align particularly well with its mining procedure. The number of e-nodes remains roughly three times the number of gates, keeping the e-graph size bounded. Moreover, subcircuit finding runtime is negligible or comparable to cell generation time (tens to hundreds of seconds per cell).

This "adder128" case demonstrates our advantage. TeMACLE identified 3-input MAJ and XOR subcircuits that included extra, area-increasing inverters, likely artifacts favored by the initial mapping. In contrast, CellE's equality saturation engine explored equivalent representations and identified the more compact, *non-inverted* 3-input MAJ and XOR functions as optimal candidates (function and layouts are shown in Figure 6). While the inverted versions might seem preferable when built from discrete gates, the non-inverted versions discovered by CellE are significantly more area-efficient when generated as single, compact standard cells. This demonstrates that exploring functional equivalences is crucial for identifying the most beneficial cells.

4.2 Applicability

We tested the extended library on adders of varying scales to evaluate the general applicability of the cells mined from "adder128". As shown in Figure 7, the normalized area, depth, and gate count reductions are highly consistent for bit widths from 16 to 256. The

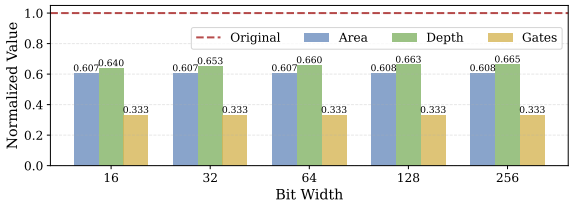


Figure 7: QoR reduction for adders with varying bit widths using original and CellE extended standard cell library.

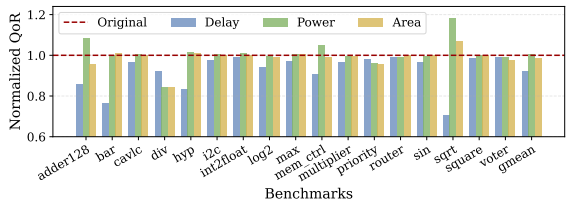


Figure 8: QoR reduction for EPFL benchmark [1] using original ASAP7 [37] and CellE extended standard cell library.

metrics remain stable, with area at ~0.608, depth at ~0.66, and gates at 0.333. This demonstrates that the discovered cells possess strong applicability, providing stable benefits across different circuit sizes.

4.3 Complete Characterization

To further evaluate our approach, we used CellE with cell generation tool AutoCellGen [2] to extend ASAP7 7.5-track library [37] (using the same cells as FreePDK45 [31]) and characterized the generated cells with Cadence Liberate 19.2.1.215. Then Cadence Genus 19.12-s121_1 was used for logical synthesis. Figure 8 shows the delay, power and area for EPFL benchmark [1] using original library and the best extended library found by CellE. The QoR cost function was chosen as $\text{Delay}^2 \times \text{Power} \times \text{Area}$, since delay is the most critical objective in general VLSI flow. As shown in Figure 8, the extended library achieves delay reduction on all benchmarks and also reduces area. On average (gmean), delay decreased by 8.00% and area decreased by 1.27%, while power remained unchanged. This is because cell fusion simplified logic and reduced depth, lowering delay. Furthermore, the commercial synthesis tool, usually optimizing delay first, prioritizes this delay reduction. This demonstrates CellE's effectiveness in a standard commercial VLSI flow.

5 Conclusion

This paper presented CellE, a novel standard cell library extension framework utilizing equality saturation to exhaustively explore functionally equivalent subcircuits. By converting a post-mapping netlist into an e-graph, CellE enables the superior discovery of highly optimized cell candidates. We introduced a customized mining algorithm to efficiently select the most beneficial patterns. Experimental results on EPFL benchmarks demonstrate CellE's effectiveness, achieving a 15.41% average area reduction over the original library, and showcasing an area reduction of up to 23.64% over prior work. Furthermore, when characterized in a commercial flow using ASAP7, the extended library shows an 8.00% average reduction in delay, confirming CellE's practical utility for QoR optimization. Future work will focus on integrating physical cell properties into the equality saturation and mining to further optimize for QoR.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. *Hypotenuse* 256, 128 (2015), 214335.

[2] Kyeonghyeon Baek and Taewhan Kim. 2021. Simultaneous transistor folding and placement in standard cell layout synthesis. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[3] Robert Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*. Springer, 24–40.

[4] Maicon S Cardoso, Gustavo H Smaniotto, Andrei AO Bubolz, Matheus T Moreira, Leomar S da Rosa, and Felipe de S Marques. 2018. Libra: an automatic design methodology for CMOS complex gates. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 10 (2018), 1345–1349.

[5] Chen Chen, Guangyu Hu, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2025. E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis. *arXiv preprint arXiv:2504.11574* (2025).

[6] Chung-Kuan Cheng, Chia-Tung Ho, Daeyeal Lee, Bill Lin, and Dongwon Park. 2021. Complementary-FET (CFET) standard cell synthesis framework for design and system technology co-optimization using SMT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 6 (2021), 1178–1191.

[7] Chung-Kuan Cheng, Andrew B Kahng, Hayoung Kim, Minsoo Kim, Daeyeal Lee, Dongwon Park, and Mingyu Woo. 2021. PROBE2. 0: A systematic framework for routability assessment from technology to design in advanced nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 5 (2021), 1495–1508.

[8] Suhyeong Choi, Jinwook Jung, Andrew B Kahng, Minsoo Kim, Chul-Hong Park, Bodhisatta Pramanik, and Dooseok Yoon. 2023. PROBE3.0: a systematic framework for design-technology pathfinding with improved design enablement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 4 (2023), 1218–1231.

[9] Giovanni De Micheli. 2023. Logic Synthesis for Emerging Technologies. In *2023 IEEE 15th International Conference on ASIC (ASICON)*. IEEE, 1–4.

[10] Calebe Micael de Oliveira Conceição and Ricardo Augusto da Luz Reis. 2019. Transistor count reduction by gate merging. *IEEE Transactions on Circuits and Systems I: Regular Papers* 66, 6 (2019), 2175–2187.

[11] Rongliang Fu, Chao Wang, Bei Yu, and Tsung-Yi Ho. 2025. TeMACLE: A Technology Mapping-Aware Area-Efficient Standard Cell Library Extension Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025).

[12] Mohan Guruswamy, Robert L Maziasz, Daniel Dulitz, Srilata Raman, Venkat Chiluvuri, Andrea Fernandez, and Larry G Jones. 1997. CELLERITY: A fully automatic layout synthesis system for standard cell libraries. In *Proceedings of the 34th annual Design Automation Conference*. 327–332.

[13] Kyeongrok Jo, Seyong Ahn, Jungho Do, Taejoong Song, Taewhan Kim, and Kyumyung Choi. 2019. Design rule evaluation framework using automatic cell layout generator for design technology co-optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 8 (2019), 1933–1946.

[14] Robert Karmazin, Carlos Tadeo Ortega Otero, and Rajit Manohar. 2013. celltk: Automated layout for asynchronous circuits with nonstandard cells. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*. IEEE, 58–66.

[15] Prasad A Kulkarni, David B Whalley, Gary S Tyson, and Jack W Davidson. 2006. Exhaustive optimization phase order space exploration. In *International Symposium on Code Generation and Optimization (CGO'06)*. IEEE, 13–pp.

[16] Tai-Cheng Lee, Cheng-Yen Yang, and Yih-Lang Li. 2020. i TPlace: machine learning-based delay-aware transistor placement for standard cell synthesis. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–8.

[17] Yih-Lang Li, Shih-Ting Lin, Shinichi Nishizawa, and Hidetoshi Onodera. 2020. Mcell: multi-row cell layout synthesis with resource constrained max-sat based detailed routing. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–8.

[18] Yih-Lang Li, Shih-Ting Lin, Shinichi Nishizawa, Hong-Yan Su, Ming-Jie Fong, Oscar Chen, and Hidetoshi Onodera. 2022. NCTUcell: A DDA-and delay-aware cell library generator for FinFET structure with implicitly adjustable grid map. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 12 (2022), 5568–5581.

[19] Tingyuan Liang, Jingsong Chen, Lei Li, and Wei Zhang. 2022. AutoCellLibX: Automated Standard Cell Library Extension Based on Pattern Mining. *arXiv preprint arXiv:2207.12314* (2022).

[20] Mark Liu. 2021. 1.1 unleashing the future of innovation. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 9–16.

[21] Purnabha Majumder, Balakrishna Kumthekar, Nimish Rameshbhai Shah, John Mowchenko, Pramit Anikumar Chavda, Yoshihisa Kojima, Hiroaki Yoshida, and Vamsi Boppana. 2011. Method of IC design optimization via creation of design-specific cells from post-layout patterns. https://patents.google.com/patent/US7941776B2/en

[22] Edward J McCluskey. 1956. Minimization of Boolean functions. *The Bell System Technical Journal* 35, 6 (1956), 1417–1444.

[23] Dipti Motiani, Veerbhan Kheterpal, and Lawrence T. Pileggi. 2013. Method for the definition of a library of application-domain-specific logic cells. https://patents.google.com/patent/US8589833B2/en

[24] Greg Nelson and Derek C Oppen. 1980. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)* 27, 2 (1980), 356–364.

[25] Robert Nieuwenhuis and Albert Oliveras. 2005. Proof-producing congruence closure. In *International Conference on Rewriting Techniques and Applications*. Springer, 453–468.

[26] Greg Northrop. 2011. Design technology co-optimization in technology definition for 22nm and beyond. In *2011 Symposium on VLSI Technology-Digest of Technical Papers*. IEEE, 112–113.

[27] Dongwon Park, Daeyeal Lee, Ilgweon Kang, Sicun Gao, Bill Lin, and Chung-Kuan Cheng. 2020. SP&R: Simultaneous Placement and Routing framework for standard cell synthesis in sub-7nm. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 345–350.

[28] Willard V Quine. 1955. A way to simplify truth functions. *The American mathematical monthly* 62, 9 (1955), 627–631.

[29] Haoxing Ren and Matthew Fojtik. 2021. Nvcell: Standard cell layout in advanced technology nodes with reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1291–1294.

[30] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee, Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, et al. 2018. The EPFL logic synthesis libraries. *arXiv preprint arXiv:1805.05121* (2018).

[31] James E Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W Rhett Davis, Paul D Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, et al. 2007. FreePDK: An open-source variation-aware design kit. In *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*. IEEE, 173–174.

[32] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 264–276.

[33] Eleonora Testa, Mathias Soeken, Luca Gaetano Amar, and Giovanni De Micheli. 2018. Logic synthesis for established and emerging computing. *Proc. IEEE* 107, 1 (2018), 165–184.

[34] J Togni, Felipe Ribeiro Schneider, Vinícius P Correia, Renato P Ribas, and André Inácio Reis. 2002. Automatic generation of digital cell libraries. In *Proceedings. 15th Symposium on Integrated Circuits and Systems Design*. IEEE, 265–270.

[35] Kaushik Vaidyanathan, Lars Liebmann, Andrzej Strojwas, and Larry Pileggi. 2014. Sub-20 nm design technology co-optimization for standard cell logic. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 124–131.

[36] Pascal Van Cleeff, Stefan Hougardy, Jannik Silvanus, and Tobias Werner. 2019. Bonncell: Automatic cell layout in the 7-nm era. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2872–2885.

[37] Vinay Vashishtha, Manoj Vangala, and Lawrence T Clark. 2017. ASAP7 predictive design kit development and cell design technology co-optimization. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 992–998.

[38] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–29.

[39] Xiaoqing Xu, Nishi Shah, Andrew Evans, Saurabh Sinha, Brian Cline, and Greg Yeric. 2017. Standard cell library design and optimization methodology for ASAP7 PDK. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 999–1004.

[40] Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. IEEE, 721–724.

[41] Jiaqi Yin, Zhan Song, Chen Chen, Qihao Hu, and Cunxi Yu. 2025. BoolE: Exact Symbolic Reasoning via Boolean Equality Saturation. *arXiv preprint arXiv:2504.05577* (2025).

[42] Kevin Zhang. 2024. 1.1 semiconductor industry: Present & future. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 67. IEEE, 10–15.

[43] Yifan Zhao, Hashim Sharif, Vikram Adve, and Sasa Misailovic. 2024. Felix: Optimizing tensor programs with gradient descent. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 367–381.

[44] Adriel Mota Ziesemer and Ricardo Augusto da Luz Reis. 2014. Simultaneous two-dimensional cell layout compaction using milp with astran. In *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 350–355.