Cutwidth Decomposition on Circuit-AIGs: Taming Verification Complexity of Arithmetic Circuits

Luca Müller
University of Bremen/DFKI
Bremen, Germany
lucam@uni-bremen.de

Mohamed Nadeem
University of Bremen
Bremen, Germany
mnadeem@uni-bremen.de

Rolf Drechsler
University of Bremen/DFKI
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Verification is an essential step in modern *Very Large Scale Integration (VLSI) Computer-Aided Design (CAD)* to avoid errors in circuits with ever-increasing complexity. Formal verification methods are gaining traction in both academia and industry, as they can ensure the complete correctness of a design. However, they come with the tradeoff of potentially exponential time and space complexity.**

This work aims to tame this complexity, establishing a formal verification approach based on the cutwidth decomposition on Circuit-*And-Inverter Graphs (AIGs)*, which can make use of different verification techniques like *Boolean Satisfiability (SAT)* and *Answer Set Programming (ASP)*. For circuits with a constant cutwidth, a linear verification time is formally proven. The practical applicability of this approach to arithmetic circuits is demonstrated by experimental evaluation of three *Arithmetic Logic Units (ALUs)*, showing that the theoretical bounds hold in practice and that our approach can outperform other formal verification approaches found in open-source software.

I. Introduction

Modern processor designs feature billions of transistors, with this number increasing over time in accordance with Moore's law [1]. In order to provide *Integrated Circuits (ICs)* which adhere to their specification and thus fulfill their intended behavior, verification plays a central role in the VLSI process, usually at a higher level of abstraction like the *Register-Transfer Level (RTL)* [2]. CAD techniques are employed to handle this complexity, as manual verification of the entire design would be infeasible [3]. While simulation-based methods can be very time-efficient for verification, only formal verification can completely ensure that the entire design is indeed correct [4], avoiding errors that only become visible in some edge-cases [5]. Due to this reason, formal methods have become ever more prevalent in both research [6] and industry [7], being integrated into commercial tools as well as open-source software [8].

One argument against formal verification lies in the fact that it employs techniques like SAT [9] and *Binary Decision Diagrams (BDDs)* [10], for which an exponential time or space complexity is observed in the worst case. Finding an optimal variable ordering for a BDD representing a given function is NP-complete [11], meaning an arbitrary ordering may result in a BDD requiring exponentially more space than the optimum. SAT itself is one of the first problems for which finding a solution was proven to be NP-complete [12].

The research field of *Polynomial Formal Verification (PFV)* addresses this issue by exploring circuit classes for which a polynomial behavior can be observed [13]. For instance, BDD sizes were shown to scale polynomially in the number of inputs for different types of adder circuits [14]–[16], and the verification of a simple ALU was conducted in polynomial time [17]. Apart from BDDs and SAT, ASP [18] has been shown to be a suitable method for PFV [19]. It makes use of the so-called *cutwidth* (also referred to as CW) decomposition [20] of the AIG [21] representation of a circuit to establish polynomial bounds. It was shown that a linear verification time can be achieved for circuits with a constant CW. However, this requires a specification of the circuit, which might not always be available. SAT is very similar to ASP in principle and some considerations for PFV have been made, likewise establishing linear bounds [22], but requiring deeper knowledge about the circuit structure. Also, both works only show experimental results for adder circuits so far. This work aims to address these shortcomings by connecting SAT and ASP with a common PFV approach, evaluating it on three different ALUs. In summary, these are the goals defined for this work:

- Establish a formal verification approach on the cutwidth decomposition of Circuit-AIGs using SAT and ASP.
- Prove linear time complexity bounds of circuits with constant cutwidth for both techniques, pointing out differences.
- Demonstrate that both SAT and ASP enable linear-time verification of three different ALUs, enhancing previous PFV bounds and outperforming SAT-based formal verification integrated in open-source software.

The remainder of this work is structured as follows: Section II introduces preliminary information and discusses the state of the art to motivate the proposed approach. The verification approach is presented in Section III. Complexity bounds of the approach are proven in Section IV and experimentally evaluated in Section V. Finally, Section VI concludes this work, giving an outlook on future work.

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project *PolyVer* (DR 287/36-1) and by the German Ministry for Research, Technology and Space (BMFTR) with project ExaVerse (grant number 01IW25003).

s	Operation
000	Constant 0
001	$b - a$
010	$a - b$
011	$a + b$
100	$a \oplus b$
101	$a \vee b$
110	$a \wedge b$
111	Constant 1

TABLE I: ALU operations on inputs a and b for all values of the select signal s

II. Preliminaries

A. Formal Verification of Arithmetic Circuits

There are several ways to ensure that a arithmetic circuit design adheres to its specification. One method which is frequently employed for SAT and also suitable for ASP is *Combinational Equivalence Checking* (CEC) [23]. The *Circuit under Verification (CuV)* is compared against a *Reference Circuit (RC)*, which is known to implement the correct behavior. For the purpose of CEC, a *miter* circuit [24] is typically constructed between the two, which outputs the value 1 if there is a mismatch between CuV and RC for any input combination.

The specific verification approach for the miter circuit depends on its representation. Graph representations [25] allow for a comprehensive complexity analysis and are widely used for considerations of many different problems [26]. One graph representation commonly used for circuits are AIGs. An AIG [21] is a directed acyclic graph $G = (V, E)$ where all inner nodes $v \in V$ represent an AND gate, whose inputs can be inverted. A node without an incoming edge denotes an input node, while a node without outgoing edges denotes an output node. The AIG representation of a circuit, namely the *Circuit-AIG*, can be constructed for an arbitrary circuit by transforming its logic to only AND and inverters, which together are functionally complete.

ALUs are components which can often be found in ICs, providing multiple operations on their two inputs a and b of size n. The corresponding operation to be observed at the output of size m is indicated by a third select input s. In the following, we consider ALUs offering 8 operations, resulting in a bit-width of 3 for s as shown in Table I. We also assume $m = n$, so the size of the output is equal to the bit-width of a single input.

Example 1. Fig. 1 shows the Circuit-AIG G of a miter circuit between two simple ALUs with two inputs of bit-width $n = 2$, and the select signal set to $s = 011$ for the ADD operation, which is the most complex operation of the ALU. Both CuV and RC use the *Ripple Carry Adder (RCA)* as their adder component. The triangle shapes of G represent the inputs and outputs of the reduced ALU, with inputs being labeled with prefix I and outputs with prefix O. The total size of G is $|V| = 32$ nodes. There are $2 * n = 4$ inputs, $I_G = \{I2, I4, I6, I8\}$ and $m = n = 2$ outputs, $O_G = \{O62, O68\}$. The final three gates before an output represent the XOR gate of the miter, e.g. 58, 60 and 62 for output $O62$. Note that unlike a standard miter structure, the outputs are not connected via OR gates, meaning each output needs to be verified individually.

B. SAT

When using SAT for CEC, the basic idea is to encode the Circuit-AIG of the miter into a SAT instance. Then, a SAT solver is used to prove either satisfiability of this instance, which means there is an error in the DuV, or unsatisfiability, indicating correct behavior. A SAT instance φ is commonly provided to the SAT solver in *Conjunctive Normal Form* (CNF). The set of variables contained in CNF φ is denoted V_φ while the set of clauses is denoted C_φ.

Any Boolean formula φ can be transformed into an equisatisfiable formula ψ (i.e., ψ is satisfiable exactly if φ is satisfiable) by *Tseitin transformation* [27]. Tseitin transformation operates in linear time by introducing a new variable for each sub-formula in φ.

Example 2. Given the Circuit-AIG G depicted in Fig. 1. The CNF representation φ of G looks as follows in standard set notation:

$$\varphi = \{\{I2, 6, 38\}, \{\neg I2, \neg 38\}, \{\neg I6, \neg 38\}, \{\neg I2, I6, 40\}, \\ \{I2, \neg 40\}, \{\neg I6, \neg 40\}, ..., \{O62\}, \{O68\}\}$$

All variables in φ correspond to the nodes of the same name in G. The first six clauses are the result of Tseitin transformation of the two AND nodes 38 and 40, both with inputs $I2$ and $I6$. For node 40, only input $I6$ is inverted, resulting in a different clause representation than for node 38. The constraints on nodes $O62$ and $O68$ ensure that φ is unsatisfiable if the behavior of the CuV is correct. An input value can be encoded in a similar manner, e.g., $\{\neg I4\}$ for $I4 \mapsto 0$. For the remainder of G, Tseitin transformation is continued until all nodes are covered, i.e., all outputs are reached. The resulting CNF φ is also referred to as the *Circuit-CNF*.

While there are some known subclasses of SAT instances for which a solution can be found in polynomial time, most Circuit-CNF formulas do not belong to any of these classes. As evident by Example 2, Circuit-CNF formulas for example do not belong to the class of 2-SAT formulas [28], which contain at most two variables per clause, or Horn-SAT formulas [29], which contain at most one positive literal per clause. For further insight on SAT, please refer to standard literature [9].

C. ASP

ASP [30] [31] is a declarative programming framework based on first-order logic that is well-known in the area of knowledge representation and non-monotonic reasoning [32] to solve NP-hard search problems. These problems are mainly reduced to computing *Answer Sets* [33]. In the context of formal verification, the basic idea is to encode a Circuit-AIG G into a logic program $\mathcal{P}$ such that each rule represents one gate of G and its connections. Then, a query q (representing one input vector) is added to $\mathcal{P}$ (labeled as $\mathcal{P}^q$). Since we consider a *miter* circuit, a constraint is added to $\mathcal{P}$ to ensure the output values are 0. Finally, an ASP solver is used to check whether an answer set of $\mathcal{P}^q$ exists [34]. A query q is said

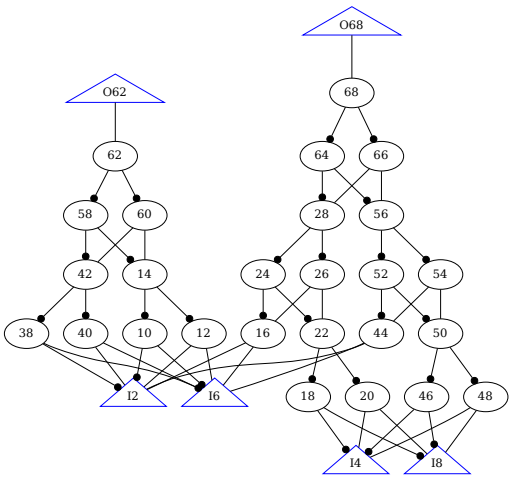


Fig. 1: Circuit-AIG G of a miter between two simple ALUs with input width $n = 2$ and select signal $s = 011$.

to be *Valid Input* if there exists an answer set of $\mathcal{P}^q$. Let Q be the set of queries representing all possible input vectors. Hence, $\mathcal{P}$ is *Valid Program* if, for every $q \in Q$, it holds that q is a valid input.

Example 3. Given the Circuit-AIG G depicted in Fig. 1. The horn ASP program $\mathcal{P}$ is defined as follows:

$$\mathcal{P} = \{val(invI2, X\hat{}1)\text{:- } val(I2, X).;$$
$$val(invI6, X\hat{}1)\text{:- } val(invI6, X).;$$
$$val(and38, X\&Y)\text{:- } val(invI2, X), val(invI6, Y).; \ldots\}.$$

All variables in $\mathcal{P}$ correspond to the nodes of the same name in G. The first two rules represents the negated inputs $invI2$ and $invI6$, computed w.r.t. $I2$ and $I6$, respectively (read: The value of $invI2$ ($invI6$) is the negation of X, under the assumption that X is equal to the value of $I2$ ($I6$)). The third rule represents the AND gate 38 with its inputs $I2$ and $I6$. Given the input vector $q = \{I2 \mapsto 0, I6 \mapsto 1, I4 \mapsto 0, I8 \mapsto 1\}$, it can be encoded into $\mathcal{P}$ using the atoms $\{val(I2, 0).; val(I6, 1).; val(I4, 0).; val(I8, 0).\}$.

D. Related Work

The verification of ALUs has been considered in previous work. Devi et al. [35] consider their own ALU implementation with more complex operations, but simulation is used for verification. Wong [36] uses formal methods for verification but employs a different approach with the help of theorem provers. Both of these works fail to address the scalability of their verification approaches for different input sizes. Drechsler et al. [17] previously reported the most relevant results for our work. They examine PFV of a simple ALU similar to the one considered in this work with the help of BDDs. However, there are differences to point out. Previous work proved quadratic bounds on the verification, while we aim to prove linear bounds for the considered ALU architecture. We also consider three ALUs with three different adder components. For the experimental considerations, we conduct a comparison of our work with formal verification approaches found in open-source software, which is omitted by Drechsler et al. entirely.

III. Verification Approach

To verify the Circuit-AIG as outlined in Section II-A, all possible combinations of primary inputs would have to be enumerated and evaluated to determine whether any primary output evaluates to 1. For Circuit-AIG G presented in Fig. 1, this may still be feasible, enumerating a total of 16 possible combinations for 4 primary inputs. In the general case however, the number of combinations in relation to n equates to 2^{2*n}, yielding an exponential behavior. Thus, to enable efficient verification, the Circuit-AIG is not simply translated to the respective SAT or ASP representation, but cutwidth decomposition is first conducted. Briefly, the cutwidth of a graph refers to the minimum number of edges crossing a cut between two consecutive vertices in a linear layout of the graph [20]. The core idea is to reduce the exponential factor in the verification complexity by only evaluating those input values which can actually occur given the structure of the circuit. To this end, the Circuit-AIG is partitioned into multiple subgraphs, which are verified in sequence, with an efficient mechanism of passing information between them. In the following, details about cutwidth decomposition and its verification are elaborated.

A. Cutwidth Decomposition

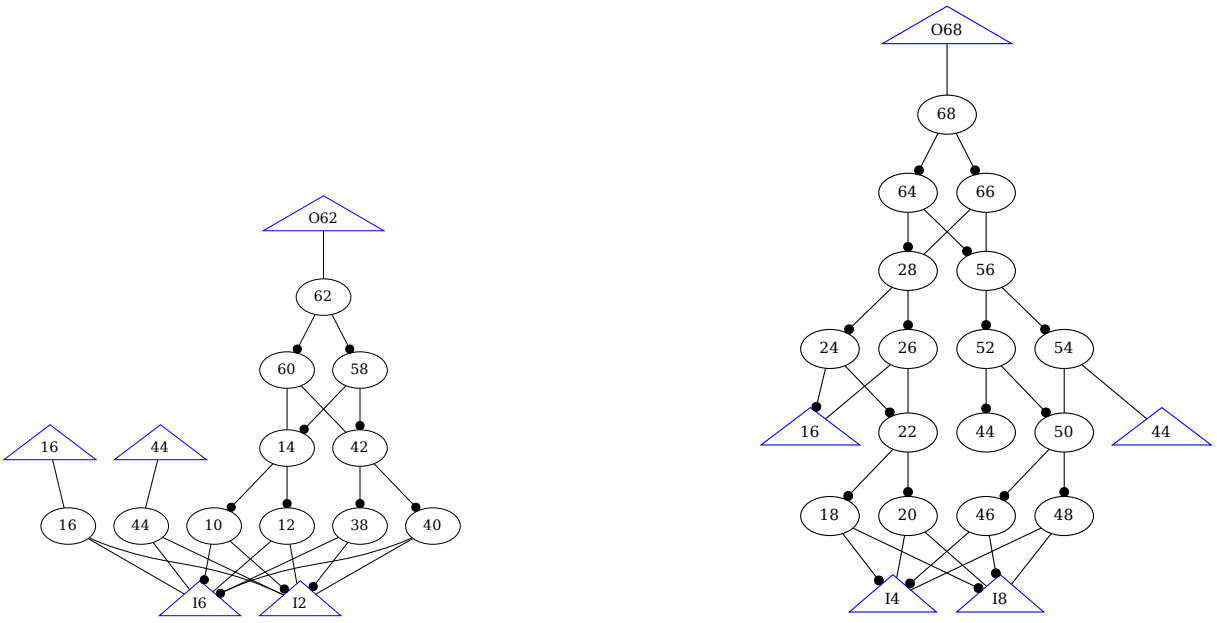
Cutwidth decomposition divides the Circuit-AIG G into m subgraphs, one for each output. Intuitively, each subgraph only contains nodes which are relevant to the given output. Nodes which influence the value of multiple outputs are added to multiple subgraphs as so-called cone nodes, enabling information passing between different subgraphs. Formally, a subgraph for a given output node is defined as follows:

Definition 1. Given a Circuit-AIG $G = (V, E)$ with output node $o_i \in O_G$ and $1 \leq i \leq m$. The output subgraph $G_{o_i} = (V_{G_{o_i}}, E_{G_{o_i}})$ is given as:

- $V_{G_{o_i}} = \{v \in \{\{o_i\} \cup \{v' \in V \mid (v', o_i) \in E\} \cup \{v'' \in V \mid \exists x \in V_{G_{o_i}} : (v'', x) \in E\}\} \mid v \notin C_{i-1}\}$
- $E_{G_{o_i}} = \{(a, b) \in E \mid a, b \in V_{G_{o_i}}\}$,

where $C_i = \bigcup_{j=1}^{i} C_{G_{o_j}}$ and $C_{G_{o_j}}$ denotes the set of cone nodes of output subgraph G_{o_j}. A cone node c is a vertex of output subgraph G_{o_i}, which shares an edge with a vertex $c' \notin V_{G_{o_i}}$, i.e., $C_{G_{o_j}} = \{c \in V \mid (c, c') \in E \vee (c', c) \in E, c' \in V \setminus V_{G_{o_i}}\}$.

Subgraphs are created by traversing all outputs of the Circuit-AIG G towards the inputs, adding nodes in accordance with Definition 1. Each resulting subgraph is in turn also a Circuit-AIG, meaning the same properties of AIGs apply and the cutwidth decomposition is suited for both SAT-based and ASP-based PFV. Cone nodes which appear as inputs (ingoing cone nodes) in a subgraph G_{o_i} are denoted CIN_i, while cone nodes appearing as outputs (outgoing cone nodes) are denoted $COUT_i$. To differentiate, primary inputs are referred to as



(a) Output subgraph G_{o_1}

(b) Output subgraph G_{o_2}

Fig. 2: Cutwidth decomposition for AIG G of Fig. 1

PIN_i, with $IN_i = PIN_i \cup CIN_i$ and the primary output as $POUT_i$, with $OUT_i = POUT_i \cup COUT_i$. The maximum number of outgoing cone nodes in a subgraph is referred to as the *cutwidth (CW)* of G, while the maximum number of input nodes in a subgraph, i.e., IN_i, is referred to as k. This number k is directly related to the CW of a Circuit-AIG, e.g., from a constant cutwidth, a constant k follows.

Example 4. The cutwidth decomposition for Circuit-AIG G of Fig. 1 is portrayed in Fig. 2. For $m = 2$ outputs, two subgraphs are created. Nodes in PIN_i and $POUT_i$ are differentiated from CIN_i and $COUT_i$ by prefixes I and O respectively, e.g., $I4$ is a primary input in G_{o_2}, while 16 is a cone input. They are both transformed to cone nodes during cutwidth decomposition, as they are influenced by inputs $I2$ and $I6$ present in G_{o_1} and influence output $O68$ in G_{o_2} themselves. For the decomposition, the cutwidth of G is $CW(G) = 2$. The maximum number of inputs is present in subgraph G_{o_2}, so $k = |IN_2| = 4$ (i.e., $IN_2 = \{I4, I8, 16, 44\}$).

B. Information Passing between Subgraphs

Information between subgraphs is passed over cone nodes. Once values are computed for the outgoing cone nodes of a given subgraph, they are stored in a table. To this end, databases [37] are used for efficient storage and table manipulation. The values of outgoing cone nodes of subgraph C_{o_i} are stored in the outgoing table τ_i. Outgoing tables are realized as a mapping between cone node values of the CuV and cone node values of the RC. Thus, the set $COUT_i$ is divided into two disjoint subsets, $COUT_i^{CuV}$ and $COUT_i^{RC}$. The cone nodes in $COUT_i^{CuV}$ and $COUT_i^{RC}$ are then mapped to binary values 0 and 1 and stored in the table. For the population of ingoing cone nodes CIN_i of a given subgraph C_{o_i}, an ingoing table τ_i^I is constructed by linear traversal of previous outgoing tables.

$COUT_1^{CuV}$	$COUT_1^{RC}$
{0}	{0}
{1}	{1}

TABLE II: Outgoing table τ_1 for subgraph G_{o_1} of Fig. 2a

Example 5. Table II shows the outgoing table τ_1 for subgraph G_{o_1} of Fig. 2a, resulting from the evaluation of all possible combinations of PIN_1. The node 16 represents the cone output of the CuV, meaning $16 \in COUT_1^{CuV}$ and $44 \in COUT_1^{RC}$. Values for both cone outputs are directly determined by the two AND nodes 16 and 44, which in turn directly depend on the primary inputs $I2$ and $I6$. Thus, the possible values for 16 can simply be defined as a function f over the values of $I2$ and $I6$ with $f(0,0) = 0, f(0,1) = 0, f(1,0) = 0, f(1,1) = 1$. Since the values for 16 and 44 depend on the exact same inputs, there are two records in τ_1: When the value of cone output 16 is 0, the value of cone output 44 is 0 and vice versa. As there is only one previous subgraph for G_{o_2}, its ingoing table τ_2^I is equivalent to τ_1 and the two records are then used as ingoing values to cone nodes $16, 44 \in CIN_2$.

Algorithm 1: Subgraph verification

Input : Subgraphs $G_o = \{G_{o_1}, \ldots, G_{o_m}\}$
Output : *Verification Success* or *Verification Fail*

1 **for** $G_{o_i} \in G_o$ **do**
2 $pin \leftarrow$ all combinations of PIN_i
3 **if** $i = 1$ **then**
4 $val \leftarrow pin$
5 **end if**
6 **else**
7 $val \leftarrow \tau_i^I \bowtie pin$
8 **end if**
9 **for** $v \in val$ **do**
10 $pout, cout \leftarrow$ value of $POUT_i$ and values of nodes in $COUT_i$ after evaluation of G_{o_i} under v
11 **if** $pout \mapsto 1$ **then**
12 **return** *Verification Fail*
13 **end if**
14 $\tau_i \leftarrow \tau_i \cup \{cout\}$
15 **end for**
16 **end for**
17 **return** *Verification Success*

C. Subgraph Verification

Algorithm 1 outlines how the verification of the cutwidth decomposition of a Circuit-AIG is conducted. Subgraphs are verified in sequence, from the first output toward the last. For the first subgraph, all combinations of primary input nodes PIN_1 are simply considered, while for subsequent ones, they are cross-joined [38] with values in incoming table τ_i^I. The evaluation of C_{o_i} referenced in Line 10 depends on the employed verification method. The subgraph is encoded into a SAT or ASP formulation as explained in Sections II-B and II-C respectively. Additionally, valuation v and the miter output are encoded as clauses or rules and a solver is employed to evaluate the formulation. If any of the given evaluations show that a primary output can exhibit the value 1, the algorithm returns a verification failure. Otherwise, the values of outgoing cone nodes $COUT_i$ are stored in outgoing cone table τ_i. Finally, when all subgraphs are evaluated and the value 1 never occurs at any primary output, the verification is successful.

16	44	I4	I8
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

TABLE III: All valuations for subgraph G_{o_2} of Fig. 2b

Example 6. Consider subgraph G_{o_2} shown in Fig. 2b. The set of all primary input combinations pin and ingoing table τ_2^I are constructed, the latter being equivalent to table τ_1 shown in Table II. The set of valuations val for G_{o_2}, created by cross-joining these two sets, is shown in Table III. This small example already shows the upside of the proposed verification flow. Instead of having to evaluate Circuit-AIG G of Fig. 1 for all $2^4 = 16$ combinations of primary inputs, Algorithm 1 evaluates subgraph G_{o_1} under $2^2 = 4$ valuations for the primary inputs PIN_1 and subgraph G_{o_2} under the 8 valuations shown in Table III, i.e., a total of 12 combinations.

IV. Complexity Bounds

For a detailed complexity analysis of the verification approach introduced above, all its steps have to be considered. The first step is the decomposition, where the Circuit-AIG $G = (V, E)$ is traversed starting from its output nodes. As nodes are only visited once, the complexity for the decomposition can be expressed as $Complexity(Decomp) = \mathcal{O}(|V| + |E|)$, i.e. linear in the size of G. Another factor are the database operations. Databases are chosen for their efficiency, allowing for constant-time access to entries in a single table. The most complex operation here is the construction of ingoing table τ_i^I, which can be performed in linear time regarding the number of tables τ_j with $1 \leq j < i \leq m$ under our assumptions. The part where the complexity for SAT and ASP differs is the evaluation of a subgraph for a given valuation, conducted in Line 10 of Algorithm 1. For ASP, it has been shown that every Circuit-AIG G can be represented as a horn ASP program $\mathcal{P}$ (i.e., "&" and "X^1" represent the logic functions AND and inverters, respectively) in [19]. Also, it has been shown that checking the consistency of the horn ASP program $\mathcal{P}$ can be achieved in linear time [39] w.r.t. size of $\mathcal{P}$. This means that the complexity of evaluation is constant for ASP w.r.t. the number of vertices in the subgraph, i.e. $Complexity(ASP) = \mathcal{O}(1)$. For SAT on the other hand, the complexity for evaluating a subgraph for a given valuation is still bounded by $Complexity(SAT) = 2^{var}$. However, in the following we show that, when increasing the number of outputs for a given circuit, the maximum number of variables in the SAT encoding of a given subgraph var_{max} can be expected to be constant w.r.t. the output size m in real designs.

Lemma 1. Given a Circuit-AIG G^C with m output nodes. If the number of nodes $|V_{G^C}|$ is bounded by a linear function in m, the maximum number of variables in any CNF instance var_{max} of subgraph G_o is bounded by a constant c.

Proof. If the number of total nodes $|V_{G^C}|$ is bounded by a linear function in m, at most a constant number of variables c is introduced for each output o of G^C, i.e., for each increase in m, at most c nodes will be added to $|V_{G^C}|$. Since each output o has its own subgraph G_o^C, the number of variables of other subgraphs is unaffected even when m is increased. The size of subgraph G_o^C in turn is bounded by the constant number of nodes c which are introduced. Hence, the maximum number of variables which can occur in one subgraph, var_{max}, also remains constant. □

Lemma 1 shows that in practice, especially for arithmetic circuits, where the logic for computing each output is the same, the complexity for subgraph evaluation with SAT can be expected to be bounded by a constant factor var_{max} regardless

n	RCA ALU					CSKA ALU					CLA ALU				
	CW	k	$\|V\|$	var_{max}	DT	CW	k	$\|V\|$	var_{max}	DT	CW	k	$\|V\|$	var_{max}	DT
64	4	9	5493	93	0.07	5	14	5789	109	0.08	6	16	5883	112	0.09
128	4	9	10997	93	0.22	5	14	11613	109	0.31	6	16	11803	112	0.33
256	4	9	22005	93	0.89	5	14	23261	109	1.46	6	16	23643	112	1.61
512	4	9	44021	93	3.80	5	14	46557	109	4.26	6	16	47323	112	7.10
1024	4	9	88053	93	21.43	5	14	93149	109	32.96	6	16	94683	112	33.54
2048	4	9	176117	93	104.56	5	14	186333	109	133.78	6	16	189403	112	144.27
3072	4	9	264181	93	262.79	5	14	279517	109	267.76	6	16	284123	112	268.06
4096	4	9	352245	93	470.88	5	14	372701	109	479.69	6	16	378843	112	488.04
5120	4	9	440309	93	761.70	5	14	465885	109	778.20	6	16	473563	112	803.49

TABLE IV: Properties of the miter Circuit-AIG of each ALU architecture for different input sizes

of the number of outputs m. For example, when adding more inputs to the miter of the ALU in Fig. 1, another subgraph equivalent to G_{o_2} is simply added. This result shows that, while larger runtimes may have to be expected for SAT when the number of vertices per subgraph is large, we assume a constant behavior for an increasing circuit size. With all these considerations at hand, the complete asymptotic complexity of the verification approach can be examined.

Theorem 1. Given the cutwidth decomposition of Circuit-AIG G with m outputs and maximum number of inputs k for any subgraph. Then G can be verified in time $\mathcal{O}(m * 2^k)$.

Proof. By Algorithm 1, verification is conducted for each subgraph, of which there are m many. The complexity of verifying subgraph C_{o_i} depends on the number of valuations in set val. As discussed above, the complexity of the evaluation itself can be disregarded in the analysis of the asymptotic complexity, as for SAT var_{max} can be expected to be constant by Lemma 1 and for ASP, the Horn formulation provides a linear verifiability. At most, val contains an entry for all combinations of primary inputs and cone inputs, i.e. $|PIN_i| \cup |CIN_i| = |IN_i|$. The maximum of this $|IN_i|$ is denoted by k, so $Complexity(Verification) = \mathcal{O}(m * 2^k)$. □

With a naive approach, SAT and ASP have the size of the problem as an exponential factor in their time complexity, e.g., the number of variables for SAT. By Theorem 1 however, for the presented verification approach, the only remaining exponential factor is k. As mentioned in Section III-A, when the cutwidth of the decomposition of Circuit-AIG G is constant, this factor k is also constant. Thus, by Theorem 1 we can identify a class of circuits with constant cutwidth for which a linear verification time w.r.t. input size n can be ensured.

V. Experimental Evaluation

To demonstrate that the established class of circuits with constant cutwidth is useful in practice, three ALUs are considered, since they are advanced arithmetic circuits which have not been studied for SAT and ASP before. Although the proposed approach could in principle be applied to more general circuits, we only evaluate ALUs to keep the scope of this work focused. As shown in Table I, next to some logical operations, addition and subtraction are the two arithmetic operations realized by the ALU architecture under consideration. For this, an adder component is required, as subtraction can simply be realized by inverting the subtrahend in accordance with two's complement. Thus, the three simple ALUs we examine feature different adder components, namely RCA, *Carry Skip Adder (CSKA)*, and *Carry Look-Ahead Adder (CLA)*. Accordingly, we refer to these three ALUs by RCA ALU, CSKA ALU and CLA ALU respectively. All ALUs are generated as Verilog files with the help of Ariths-Gen [40] and transformed into an AIG using Yosys [8].

A. Properties of ALU Circuit-AIGs

Before looking at the verification runtime for all three ALUs, their properties are studied to show they belong to the class of constant cutwidth circuits and be able to make predictions about their runtime behavior. To this end, the construction of a miter between two input AIGs for CuV and RC is realized, and the cutwidth decomposition on the resulting Circuit-AIG is implemented in Python [41]. Our goal is to check the functional correctness of the CuV w.r.t. the RC. These circuits are expected to be different in terms of the architecture, but functionally equivalent. As demonstrated by previous work [19], RCA-based adder circuit has the least cutwidth and consequently is the fastest architecture to verify. Hence, we have selected ALU RCA as our RC across all different ALU architectures. Five different properties are portrayed for each of the three miter circuits for increasing values of n in Table IV: The cutwidth of their decomposition CW, the maximum number of inputs for a given subgraph k, the total number of nodes in the Circuit-AIG $|V|$, the maximum number of variables in a SAT instance of a subgraph var_{max} and the runtime of the cutwidth decomposition in seconds DT. The first observation to make is that for all three ALUs, the cutwidth is constant. As a result, the maximum number of inputs in a given subgraph k is also constant, meaning a linear verification time can be expected according to Theorem 1. In the general case, the verification could reject circuits for which the cutwidth is non-constant. Both CW and k are greater for more complex adder components, such that *CW*(RCA ALU) $<$ *CW*(CSKA ALU) $<$ *CW*(CLA ALU). The result of Lemma 1 is underlined by column var_{max}. With the number of nodes increasing in a linear manner, as shown by column $|V|$, var_{max} remains constant. Hence, the complexity of the evaluation remains constant, even when n increases, as proven in Lemma 1. This behavior holds true for all three of the

n	RCA ALU					CSKA ALU					CLA ALU				
	CW SAT		CW ASP		CEC	CW SAT		CW ASP		CEC	CW SAT		CW ASP		CEC
	Verify	Total	Verify	Total		Verify	Total	Verify	Total		Verify	Total	Verify	Total	
64	20.76	20.98	2.08	2.44	0.93	90.34	90.59	3.81	4.12	1.33	165.30	165.56	6.42	6.76	1.47
128	42.47	43.06	4.47	5.04	3.08	189.93	190.60	8.29	9.95	3.62	341.59	342.34	13.47	14.32	4.27
256	85.07	86.95	9.37	11.68	9.16	390.06	392.05	20.97	24.35	14.98	704.32	706.31	28.60	30.85	20.65
512	171.80	179.14	18.79	27.22	46.86	784.98	792.56	45.04	56.67	55.69	1397.09	1403.90	62.86	70.95	65.26
1024	346.03	378.87	38.49	72.22	213.31	1487.79	1619.79	87.96	130.14	265.63	2816.09	2846.48	123.22	158.83	308.75
2048	698.55	843.94	77.48	221.36	1025.38	3165.30	3302.86	149.94	313.93	1347.27	T.O.	T.O.	233.35	364.38	1547.09
3072	1050.84	1385.01	112.7	449.61	2764.91	T.O.	T.O.	244.53	593.72	T.O.	T.O.	T.O.	353.07	657.18	T.O.
4096	1388.26	1941.64	154.91	761.19	T.O.	T.O.	T.O.	345.91	977.17	T.O.	T.O.	T.O.	490.91	1056.64	T.O.
5120	1737.68	2641.64	228.85	1168.04	T.O.	T.O.	T.O.	435.40	1447.38	T.O.	T.O.	T.O.	636.10	1579.68	T.O.

TABLE V: Runtime of ALU verification in seconds for each method

considered ALUs. Finally, the cutwidth decomposition time reveals some expected behavior. As the number of nodes of the Circuit-AIG increases, the decomposition time does as well. Between different ALUs, it is very similar for equal values of n, since the number of nodes is not significantly different.

B. Experimental Results

For the evaluation of the verification runtime, the procedure outlined in Algorithm 1 is also implemented in Python and applied to the three ALUs for different values of n. As solver backends, the state-of-the-art solver CaDiCaL [42] is used for SAT and the well-known Clingo [43] is employed for ASP, both with default configurations. For comparison, a standard CEC procedure integrated in Yosys is run on the same AIG inputs. Yosys is an established open-source tool used for synthesis, design and verification [44] [45] and offers a fairer comparison opposed to other tools like ABC [46], which employs structural hashing [47] in addition to CEC. The miter for the CEC flow circuit is constructed by the Yosys command `miter -equiv gold gate miter`. The SAT command `sat -prove trigger 0 miter` finally carries out the verification itself. Each verification is run five times and mean values are calculated, reducing noise in the execution times. Results are provided in seconds, rounded to ten milliseconds. A timeout (T.O.) is set to one hour. All experiments are run on the same machine with an 8-core AMD Ryzen™ 7 PRO 5850U CPU and 42GB of main memory.

Table V shows the runtime for verifying each ALU with the given method in seconds. Each combination of n and ALU refers to the exact same input AIGs for CuV and RC as the corresponding entry in Table IV. For SAT and ASP, the column *Verify* refers to the runtime of all steps in Algorithm 1, i.e., database operations, encoding and evaluation, while the column *Total* refers to the wall clock time of the complete approach, including for example the cutwidth decomposition. Discrepancies between the sum of verification runtime plus decomposition time and the total runtime can be attributed to processing steps, e.g., reading the input AIGs of CuV and RC.

The results of Table V confirm the theoretical considerations of Section IV and the assumptions made based on Table IV. For all three ALUs and both verification methods, SAT and ASP, the verification runtime behaves linearly in n, as can be seen in the *Verify* column. Fig. 3 underlines the linear behavior, by plotting the runtimes for each value of n. As can be seen, the dashed lines for SAT and the dotted lines for

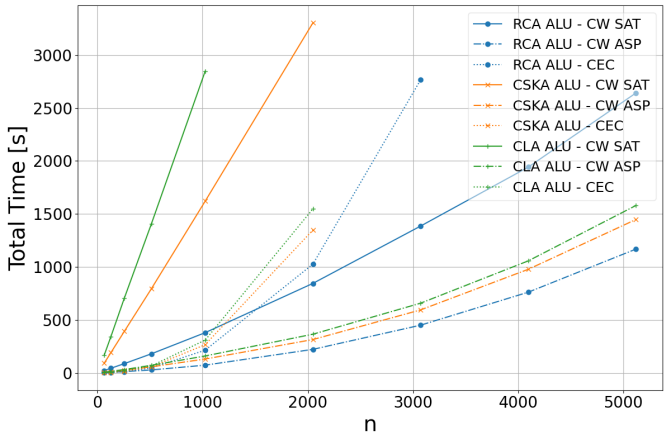


Fig. 3: Runtime comparison for different verification methods

ASP progress in a linear fashion, with the small dents in the curve caused by an increase in decomposition time. Runtimes for ASP are greatly decreased compared to SAT for all ALUs and all values of n. Here, the upside of Horn formulas as described in Section II-C becomes apparent. While the number of variables in a SAT instance behaves constantly in n, the values are still relatively high, starting from 93 variables for RCA ALU. Hence, ASP can evaluate each valuation much more efficiently, resulting in a greatly reduced runtime.

Comparing the proposed verification approach to the standard CEC flow reveals that the main upside lies in its scalability. As can be observed in Fig. 3, the plot for CEC grows exponentially, and timeouts are reached quickly. ASP outperforms CEC starting from $n = 512$ for RCA ALU and starting from $n = 1024$ for CSKA ALU and CLA ALU. Overall, these experimental results underline our theoretical considerations and confirm that standard formal verification approaches can indeed be outperformed, additionally providing predictability of the runtime behavior. Moreover, unlike the work [17], which establishes polynomial-time verification of an ALU, our approach improves upon the state-of-the-art by showing that an ALU can be verified in linear time.

C. Runtime Improvements

An improvement of the runtime of our verification approach presented above is possible with parallelization for the evaluation of subgraphs under a given valuation. Since the set of valuations is known before the actual evaluation,

Line 9 of Algorithm 1 can be parallelized completely, as the evaluation of one valuation does not depend on the other. In the implementation of this concept, each computation is simply delegated to a separate execution thread, where the maximum number of threads that can run in parallel is limited to a user-definable t.

n	CW SAT		CW SAT Thread	
	Verify	Total	Verify	Total
1024	1487.79	1619.79	361.95	392.79
2048	3165.30	3302.86	732.05	862.81
3072	T.O.	T.O.	1115.89	1425.26
4096	T.O.	T.O.	1468.01	2005.50
5120	T.O.	T.O.	1864.49	2746.38

TABLE VI: Improvement of CSKA ALU verification by applying threading to SAT

Table VI shows how this parallelization technique can improve the runtimes of the proposed verification approach in practice. For this small case study, the CSKA ALU is considered for the higher values of n, showing an upside of SAT, which can easily be parallelized by creating different instances for each valuation. The value of t is set to 32. Comparing the total wall clock time with results previously obtained without threading, it can be seen that a speedup around the factor 4 can be achieved. For values of $n = 3072$ through $n = 5120$, verification runs which previously reached the timeout of one hour are now feasible.

VI. Conclusion

This work explored how the verification complexity of arithmetic circuits can be tamed using SAT and ASP. A verification approach was presented which can employ both of these formal verification techniques, making use of the cutwidth decomposition of the Circuit-AIG as a shared basis. With the established approach, it was proven that for circuits with a constant cutwidth, linear-time verification can be ensured, regardless of the verification method. It was demonstrated that this theoretical finding is practically applicable to arithmetic circuits, as a linear verification time could be observed for three different ALUs, improving on polynomial bounds derived in previous work and outperforming a standard CEC flow for circuits with a high input-width. Future research may be directed toward investing further engineering efforts which provide improvements of runtimes of the presented verification approach observed in practice. Furthermore, other circuit types which could potentially exhibit a constant cutwidth could be explored, taking considerations for BDDs [48] as a starting point.

References

[1] G. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, p. 82–85, Jan. 1998.

[2] W. Wolf, *Modern VLSI design: system-on-chip design*. Pearson Education, 2002.

[3] A. Newton, "Computer-aided design of VLSI circuits," *Proceedings of the IEEE*, vol. 69, no. 10, pp. 1189–1199, 1981.

[4] R. Drechsler, *Advanced formal verification*. Boston: Kluwer Academic Publishers, 2004.

[5] A. Prout, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, *et al.*, "Measuring the impact of spectre and meltdown," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–5, IEEE, 2018.

[6] S.-M. Edgar and M.-V. David, "State of the Art in the Research of Formal Verification," *Ingeniería, Investigación y Tecnología*, vol. 15, no. 4, p. 615–623, 2014.

[7] R. Brinkmann and D. Kelf, "Formal Verification—The Industrial Perspective," in *Formal System Verification: State-of the-Art and Future Trends* (R. Drechsler, ed.), p. 155–182, Cham: Springer International Publishing, 2018.

[8] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free Verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, vol. 97, 2013.

[9] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications, IOS Press, 2021.

[10] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C–35, no. 8, p. 677–691, 1986.

[11] Bollig, B. and Wegener, I., "Improving the variable ordering of obdds is np-complete," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996.

[12] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, (Shaker Heights, Ohio, United States), p. 151–158, ACM Press, 1971.

[13] R. Drechsler, "Fast and Exact is Doable: Polynomial Algorithms in Test and Verification," in *2022 IEEE 23rd Latin American Test Symposium (LATS)*, (Montevideo, Uruguay), p. 1–2, IEEE, 2022.

[14] R. Drechsler, "PolyAdd: Polynomial Formal Verification of Adder Circuits," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, (Vienna, Austria), p. 99–104, IEEE, 2021.

[15] A. Mahzoon and R. Drechsler, "Polynomial Formal Verification of Prefix Adders," in *2021 IEEE 30th Asian Test Symposium (ATS)*, (Matsuyama, Ehime, Japan), p. 85–90, IEEE, 2021.

[16] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Polynomial Formal Verification of Floating Point Adders," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (Antwerp, Belgium), p. 1–2, IEEE, 2023.

[17] R. Drechsler, A. Mahzoon, and L. Weingarten, *Polynomial Formal Verification of Arithmetic Circuits*, vol. 99 of *Lecture Notes on Data Engineering and Communications Technologies*, p. 457–470. Singapore: Springer Nature Singapore, 2022.

[18] M. Gelfond and V. Lifschitz, "The Stable Model Semantics For Logic Programming," *Logic Programming*, vol. 2, 12 2000.

[19] M. Nadeem, J. Kleinekathofer, and R. Drechsler, "Polynomial Formal Verification exploiting Constant Cutwidth," *34th International Workshop on Rapid System Prototyping (RSP)*.

[20] D. M. Thilikos, M. Serna, and H. L. Bodlaender, "Cutwidth i: A linear time fixed parameter algorithm," *Journal of Algorithms*, vol. 56, no. 1, pp. 1–24, 2005.

[21] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *2006 43rd ACM/IEEE Design Automation Conference*, (San Francisco, CA, USA), pp. 532–535, 2006.

[22] L. Müller and R. Drechsler, "SAT can Ensure Polynomial Bounds for the Verification of Circuits with Limited Cutwidth," in *2024 27th Euromicro Conference on Digital System Design (DSD)*, pp. 57–64, 2024.

[23] E. Goldberg, M. Prasad, and R. Brayton, "Using SAT for combinational equivalence checking," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, (Munich, Germany), p. 114–121, IEEE Comput. Soc, 2001.

[24] D. Brand, "Verification of large synthesized designs," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, (Santa Clara, CA, USA), p. 534–537, IEEE Comput. Soc. Press, 1993.

[25] C. Berge, *The theory of graphs*. Courier Corporation, 2001.

[26] D. Marx, G. S. Sankar, and P. Schepper, "Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth," in *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)* (N. Bansal, E. Merelli, and J. Worrell, eds.), vol. 198 of *Leibniz International Proceedings in*

Informatics (LIPIcs), (Dagstuhl, Germany), pp. 95:1–95:20, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
[27] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*, p. 466–483. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983.
[28] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," *Information Processing Letters*, vol. 8, no. 3, p. 121–123, 1979.
[29] W. F. Dowling and J. H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional horn formulae," *The Journal of Logic Programming*, vol. 1, no. 3, p. 267–284, 1984.
[30] V. W. Marek and M. Truszczynski, "Stable models and an alternative logic programming paradigm," *A Computing Research Repository*, 1998.
[31] I. Niemelä, "Logic programs with stable model semantics as a constraint programming paradigm," *Annals of Mathematics and Artificial Intelligence*, vol. 25, no. 3, pp. 241–273, 1999.
[32] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
[33] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, pp. 365–385, 1991.
[34] T. Sato, "Completed logic programs and their consistency," *The Journal of Logic Programming*, vol. 9, no. 1, pp. 33–44, 1990.
[35] D. A. Devi and L. S. Sugun, "Design, implementation and verification of 32-Bit ALU with VIO," in *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pp. 495–499, 2018.
[36] W. Wong, "Formal verification of VIPER's ALU," Tech. Rep. UCAM-CL-TR-300, University of Cambridge, Computer Laboratory, Apr. 1993.
[37] P. Revesz, "Introduction to databases," *Texts in Computer Science*, 2010.
[38] M. Hannula, Z. Zhang, B.-K. Song, and S. Link, "Discovery of Cross Joins," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6839–6851, 2023.
[39] J. K. Fichte and S. Szeider, "Backdoors to tractable answer set programming," *Artificial Intelligence*, vol. 220, pp. 64–103, 2015.
[40] J. Klhufek and V. Mrazek, "ArithsGen: Arithmetic Circuit Generator for Hardware Accelerators," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 44–47, 2022.
[41] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
[42] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, *CaDiCaL 2.0*, vol. 14681 of *Lecture Notes in Computer Science*, p. 133–152. Cham: Springer Nature Switzerland, 2024.
[43] M. Gebser, R. Kaminski, A. König, and T. Schaub, "Advances in gringo series 3," in *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR'11, (Berlin, Heidelberg), p. 345–351, Springer-Verlag, 2011.
[44] P. Choudhary, L. Bhargava, M. Fujita, and V. Singh, "Synthesis of LUT based approximating adder circuits with formal error guarantees," in *International Symposium on VLSI Design and Test*, pp. 435–449, Springer, 2022.
[45] X. Dai, M. Cheng, W. Fan, and Y. Tai, "An Automatic Extraction and Verification Method of Security Properties for Integrated Circuit Design," in *2024 9th International Conference on Integrated Circuits and Microsystems (ICICM)*, pp. 553–559, 2024.
[46] A. Mishchenko, "ABC. A System for Sequential Synthesis and Verification." https://people.eecs.berkeley.edu/~alanmi/abc/. accessed 17-09-25.
[47] A. M. R. Brayton and A. Mishchenko, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, pp. 15–22, 2006.
[48] R. Drechsler, A. Mahzoon, and M. Goli, "Towards Polynomial Formal Verification of Complex Arithmetic Circuits," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 1–6, 2022.