EasyChair Preprint

№ 14322

SAT Can Ensure Polynomial Bounds for the Verification of Circuits with Limited Cutwidth

Luca Mueller and Rolf Drechsler

August 6, 2024

SAT can Ensure Polynomial Bounds for the Verification of Circuits with Limited Cutwidth

Luca Müller
German Research Center for Artificial Intelligence
Bremen, Germany
luca.mueller@dfki.de

Rolf Drechsler
University of Bremen
German Research Center for Artificial Intelligence
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—As hardware designs are getting more complex, verification becomes ever more important to prevent producing chips which do not behave according to their specification. This increasing complexity also impacts the verification process, resulting in a longer time-to-market. Ensuring that the verification itself can be conducted efficiently helps facing these challenges. Additionally, taking the efficient verification into consideration during the design phase further enables the optimization of the whole process.**

In this paper, we present a SAT-based verification flow and how it can ensure polynomial bounds for the verification of circuits with limited cutwidth. To demonstrate our approach, the flow is applied to three different adder architectures. Addition is one of the most essential operations in digital computations and the simplicity of its circuit realizations makes it a good starting point to explore their efficient verification using SAT. We provide theoretical proofs that SAT can be used for *Polynomial Formal Verification* (PFV) of circuits with limited cutwidth. We then show that for the considered adder circuits, a linear time complexity of the verification process can be ensured and confirm our findings by experimental evaluation with our own SAT solver.

I. Introduction

Verification is an essential step in today's circuit design. Distributing chips with erroneous behavior can cause serious harm in safety-critical applications and creates extreme costs for manufacturers, which can be avoided by ensuring the functional correctness of a circuit during the design process. Simulation and emulation approaches for verification are usually very time efficient, but are unable to cover the complete search space when looking for errors in a design. Formal verification on the other hand can ensure the correctness completely, at the expense of potentially exponential time complexity [1].

Formal methods benefit from a large research community proposing new and improved approaches for formal verification and its underlying algorithms [2]. As digital systems are becoming more complex, a wider adoption of these approaches in industry tools can be observed [3]. One of the most popular methods for formal verification is SAT [4], with its application in techniques like combinational equivalence checking [5].

SAT is an NP-complete problem in the general case [6], while some subclasses like 2-SAT [7] and Horn-SAT [8] can be solved in polynomial time. Despite the fact that instances of practical circuits are generally not part of these polynomial subclasses, experimental evaluation has shown that SAT can in fact be used to efficiently verify large hardware designs [9]. However, we are not aware of any work providing theoretical bounds on the runtime of SAT solvers for circuit verification, which means that a polynomial behavior of the verification cannot be guaranteed.

Modern SAT solvers [10] [11] rely on a variety of heuristics to accelerate the solving process, which, while improving the runtime of the verification, make it difficult to analyze and make predictions about the solving time of large circuit designs. In order to overcome this limitation, we explore the concept of cutwidth [12] to enable the calculation of polynomial bounds for the verification of combinational circuits using SAT.

Analyses for the complexity of formal verification methods are conducted in the research field of *Polynomial Formal Verification* (PFV) and for BDDs, some circuit classes with polynomial behavior have been found [13] [14] [15]. While a method for exploiting cutwidth for the PFV of combinational circuits using *Answer Set Programming* (ASP) [16] has previously been explored in [17], using SAT can offer several advantages compared to ASP. SAT is one of the core problems of computer science and thus has a large research field with many applications and tools supporting it [18]. Compared to ASP, it is more widely used for verification, as it is generally the easier formalism to understand and use. Without the need to formally specify the circuit behavior with the help of ASP rules, SAT also requires less information about the circuit under verification, as a reference implementation is sufficient for the miter construction employed by our approach (cmp. Section III-A).

In this paper, we propose our own SAT-based verification flow and provide theoretical evidence for its polynomial behavior in the formal verification of circuits with limited cutwidth. We show that the time complexity of SAT is bounded linearly for circuits with constant cutwidth, as demonstrated on the three adder architectures under consideration, namely *Ripple-Carry Adder* (RCA), *Carry-Lookahead Adder* (CLA) and *Carry-Skip Adder* (CSA). We complement our theoretical proofs with practical experimental evaluation to confirm our findings.

The remainder of this paper is structured as follows. In Section II, we provide preliminary information to keep this work self-contained. Section III explains our approach and the general verification flow. The concept of cutwidth and the associated theoretical proofs are introduced in Section IV. Section V provides the main contribution by presenting polynomial bounds for the considered adder architectures. The practical experiments and their results are evaluated in Section VI. Finally, Section VII discusses our findings and gives an outlook on future research directions, before Section VIII concludes our work.

II. Preliminaries

A. SAT

The *Boolean Satisfiability Problem* (SAT) is defined as follows.

Definition 1. Given a Boolean function ϕ over n variables. Does a mapping α of variables in ϕ to the Boolean truth values $\{0, 1\}$ exist, such that $\phi(\alpha) = 1$?

SAT is an NP-complete problem as proven by Cook in 1971 [6]. This means that in the general case, no efficient algorithm can exist such that an arbitrary SAT instance is solvable in polynomial time, unless $P = NP$. Over the years, many SAT solvers have been developed with techniques like implication learning, activity tracking and restarts to combat this problem and improve the solving times of SAT [10] [11] [19]. However, these heuristic approaches have no effect on the asymptotic complexity of SAT itself.

The formula is provided to most SAT solvers as a Boolean expression in conjunctive normal form.

Definition 2. A Boolean expression ψ is in *Conjunctive Normal Form* (CNF) if

$$\psi = \bigwedge_j \bigvee_i l_{i,j},$$

where l is a literal over the variables of ψ.

The advantage of CNF is that every boolean expression ψ can be transformed to an *equisatisfiable* CNF χ in linear time. This means that χ is satisfiable, if and only if ψ is satisfiable.

B. Binary Adders

Binary addition can be realized on the gate level with the help of two components, *Half Adders* (HA) and *Full Adders* (FA).

The *Half Adder* has two input bits, a and b and two output bits s (sum) and c (carry). The function of a HA is described as follows

$$s = a \oplus b$$

$$c = a \cdot b$$

The *Full Adder* has an additional input bit, c_{in} and its function can be described as

$$s = a \oplus b \oplus c_{in}$$

$$c = ab + ac_{in} + bc_{in}$$

In a more general sense, given two binary numbers a and b of n bits, the sum of the two numbers s can be expressed by the equation

$$\forall_{i=0}^{n-1} s_i = a_i \oplus b_i \oplus c_{i-1}$$

where

$$\forall_{i=0}^{n-1} c_i = a_i b_i + a_i c_{i-1} + a c_{i-1}$$

and

$$c_{-1} = 0$$

Several realizations of the addition function are possible on the gate level, varying in cost (total number of gates) and depth (number of gates on longest path through the circuit).

1) Ripple-Carry Adder: The *Ripple-Carry Adder* (RCA) is the simplest adder realization with the lowest cost but the largest depth. It consists of a simple sequence of $n - 1$ FAs and one HA for the least significant bits without a carry input. Both cost and depth are linear in the number of inputs.

2) Carry-Lookahead Adder: The *Carry-Lookahead Adder* (CLA) is a fast adder, reducing the depth by determining the carry bits first. The equation for the calculation of sum bits in binary addition shows that once the carry bits are calculated, the rest of the addition can be carried out in parallel. For this reason, the CLA makes use of the generation and propagation properties of binary addition

$$\text{For } 0 \leq i < n : p_{i,i} = a_i \oplus b_i, g_{i,i} = a_i \cdot b_i$$

$$\text{For } i \leq k < j : p_{j,i} = p_{k,i} \cdot p_{j,k+1}, g_{j,i} = g_{j,k+1} + (g_{k,i} \cdot p_{j,k+1})$$

If a carry bit c_i is set, it is either generated at the current index or propagated from a lower index.

$$\text{For } 0 \leq i < n : c_i = g_{i,0} + p_{i,0} \cdot c_{i-1}$$

The CLA has a linear size and a logarithmic depth in the number of inputs.

3) Carry-Skip Adder: The *Carry-Skip Adder* (CSA) has the same goal as the CLA of reducing depth in favor of cost. It divides the addition operation into multiple blocks, where each block has its own local carry signal. If this local carry signal has no effect on the final sum, this block can then be skipped when propagating the global carry signal through the circuit, reducing its worst-case depth. Like the CLA, the CSA has linear size and logarithmic depth in the number of inputs.

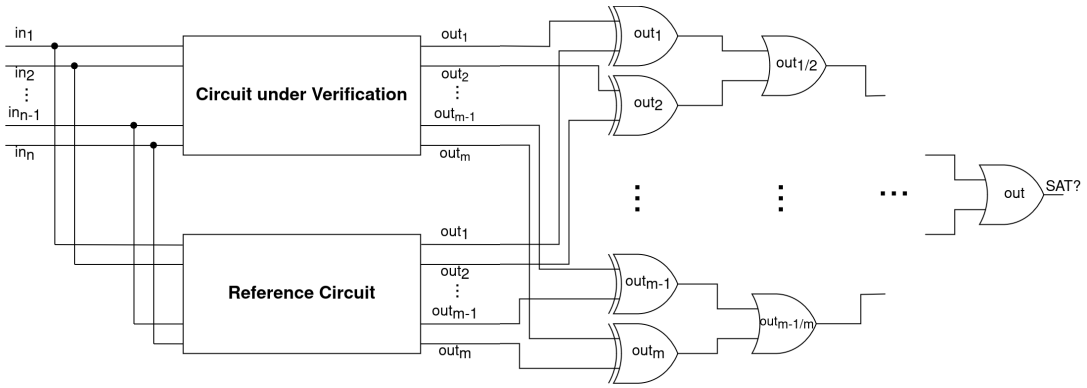


Fig. 1: A miter circuit

III. Verification Flow

A. Approach

One of the approaches for circuit verification using SAT is combinational equivalence checking, where the circuit under verification is compared against a reference circuit. To this end, a miter is created between the two circuits and checked for satisfiability. Figure 1 shows how the miter circuit functions. It connects the corresponding inputs of the two circuits to have the same value and compares their respective outputs with XOR gates, which can only have their outputs set to 1 if the two circuits produce different outputs for the same input values. These XOR gates are in turn connected via a tree-like structure of OR gates to detect any difference in the outputs. The complete miter circuit can now be tested for satisfiability by a SAT solver. If there is any combination of inputs for which the miter produces the output 1, the two circuits are not combinationally equivalent.

In order to be able to solve the miter instance with a SAT solver, it needs to be transformed to a *Circuit-CNF* formula. This can be done via Tseitin transformation [20]. Tseitin transformation for circuits operates on a gate-by-gate basis, introducing an auxiliary variable for each gate wire and adding the gate's CNF to the set of clauses. The advantage of Tseitin-transformation is that each gate can be translated separately by introducing one auxiliary variable for each gate output. This means that the transformation can be conducted in linear time and introduces a linear number of variables in the number of gates in the underlying circuit.

Example 1. An XOR gate can be expressed in CNF via Tseitin Transformation as follows:

$$\{\{-1,-2,-3\},\{1,2,-3\},\{1,-2,3\},\{-1,2,3\}\}$$

The variables 1 and 2 represent the two inputs to the XOR gate, while the auxiliary variable 3 is introduced to denote its output. This auxiliary variable can then be used as the input for another gate to represent a connection between two gates. For the XOR gate, each clause corresponds to one of the four input combinations with its corresponding output as specified by the XOR function. While this is the maximum number of clauses for a two input gate, other gates like the AND gate and the OR gate require only three clauses for their CNF representation.

The advantage of Tseitin transformation over different representations like *And-Inverter-Graphs* (AIG) is that no structural information of the circuit is lost during translation. This allows for the identification of the complete circuit structure in the resulting CNF, as each variable can be mapped to either a primary input, primary output, or an intermediate wire. In order to minimize the amount of auxiliary variables which are introduced, the circuit may be considered at the level of components instead of strictly on the level of gates. For example, a full adder can be translated as a component with three inputs and two outputs with its respective operations, instead of being translated as five separate gates, reducing the number of variables in the resulting Circuit-CNF formula.

B. SAT Solver

SAT solvers have gradually improved over the years and new techniques have decreased their runtime and memory consumption. These predominantly heuristic methods make it difficult to make exact predictions of their runtime behavior. This becomes especially relevant when considering the verification of circuits with an increasing number of inputs. If the verification of a specific circuit is feasible for n inputs, there is no guarantee that a heuristic solver can provide a result for $n + m$ (given $m > 0$) inputs within a certain time. Some SAT solvers introduce non-determinism through techniques like random restarts, which means that even for the same SAT instance, the runtime may vary.

As evident from the argument above, a deterministic SAT algorithm is required to conduct a theoretical analysis of its runtime behavior for verification. For this purpose, we use a caching-based SAT solver. The general algorithm of *CacheSAT* is outlined in Figure 2. Variables are assigned according to a static ordering, which is given to the solver. For each variable, the value 0 is applied first, then the value 1. The *CacheSAT* procedure is called recursively, until a conflict occurs. The conflict formula is then inserted into the cache and a lookup is done before going deeper in the recursion tree to avoid

```
procedure SAT(Formula, Cache)
    zero ← CACHESAT(Formula, 0, Cache)
    one ← CACHESAT(Formula, 1, Cache)
    if zero = UNSAT and one = UNSAT then
        return UNSAT
    else
        return SAT
    end if
end procedure

procedure CACHESAT(Formula, Value, Cache)
    Literal ← first variable in ordering with value value
    res ← APPLYFORMULA(Formula, Literal)
    if res = SAT then
        return SAT
    else if res = UNSAT then
        INSERTCACHE(Cache, Formula)
    end if

    if CACHELOOKUP(Cache, Formula) = true then
        return UNSAT
    end if

    zero ← CACHESAT(Formula, 0, Cache)
    one ← CACHESAT(Formula, 1, Cache)
    if zero = UNSAT and one = UNSAT then
        INSERTCACHE(Cache, Formula)
        return UNSAT
    end if
end procedure
```

Fig. 2: CacheSAT algorithm

testing a formula multiple times. For the cache lookup, a hash value of the formula is created in $O(1)$ time, ensuring that it can always take place in constant time.

It is important to note that the *CacheSAT* algorithm is expected to have larger run-times than modern SAT solvers, as it is missing features like random restarts and decision heuristics. However, the absence of non-deterministic techniques allows for an analysis of the algorithm's behavior and runtime, which is of relevance in the following section.

IV. Polynomial Verification

A. Implied Clauses

An important property of combinational circuits which can be taken advantage of during functional verification is that the values of certain signals depend on other signals that come before them. Since the structure of the circuit is preserved during the CNF translation, the search space of the SAT solver can be limited.

Example 2. Consider a set of clauses

$$\{\{-1,-2,\text{-}5\},\{1,2,\text{-}5\},\{1,-2,5\},\{-1,2,5\},$$
$$\{-1,-2,\text{-}9\},\{1,2,\text{-}9\},\{1,-2,9\},\{-1,2,9\},$$
$$\{\textbf{-5, -9, -13,}\},\{\textbf{5, 9, -13,}\},\{\textbf{5, -9, 13,}\},\{\textbf{-5, 9, 13}\}\}$$

The first eight clauses correspond to two XOR gates with 1 and 2 as input wires and 5 and 9 as output wires respectively. The remaining four clauses represent an XOR gate of the miter logic, taking variables 5 and 9 as inputs and introducing variable 13 as the gate's output.

Once the assignments of the two variables 5 and 9 are fixed by the clauses containing the variables 1 and 2, the remaining four clauses can be *implied*, as they will either be resolved or just contain one variable, namely 13. In a more general sense, the last four clauses can be implied by resolution of the first eight clauses.

Definition 3 Let ϕ be a Circuit-CNF formula over variables V_ϕ with clauses C_ϕ. Given a set $A \subseteq V_\phi$, the set $I_A \subseteq C_\phi$ of *implied clauses* contains all clauses of ϕ which can be implied by assigning a value to all variables in A.

The search space of the SAT solver can be constrained to all variables contained in A, if I_A contains all remaining clauses of ϕ which are not resolved by setting the values of the variables in A.

One useful set A^V to consider is the set of all variables representing the circuit under verification. A^V especially contains all primary inputs, which adds all clauses of the reference circuit to I_{A^V}. When all variables for both the circuit under verification and the reference circuit are fixed, the set of clauses representing the miter logic is also contained in I_{A^V}. This means that the search space for our verification approach presented in Section III can be constrained to the variables of the circuit under verification.

B. Cutwidth

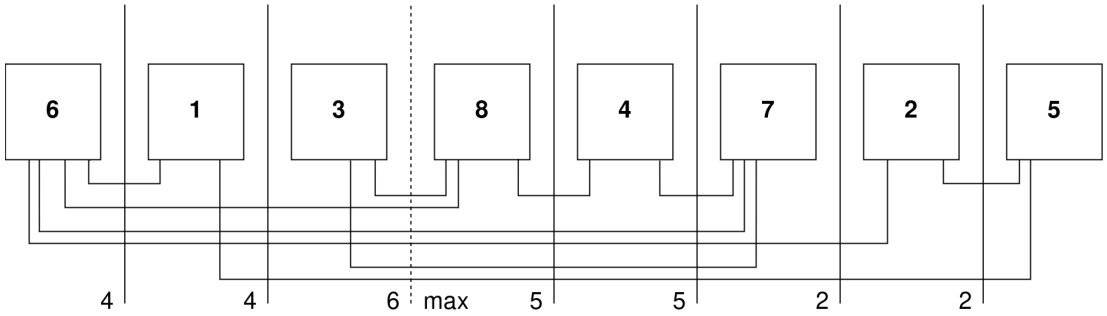
For an analysis on the complexity of SAT for circuit verification, we introduce the concept of cutwidth [12].

Definition 4. Given an undirected graph $G = \{V, E\}$ and a bijective ordering function $h : V \mapsto \{1, 2, ..., |V|\}$. The *cutwidth* of G under ordering h is defined as

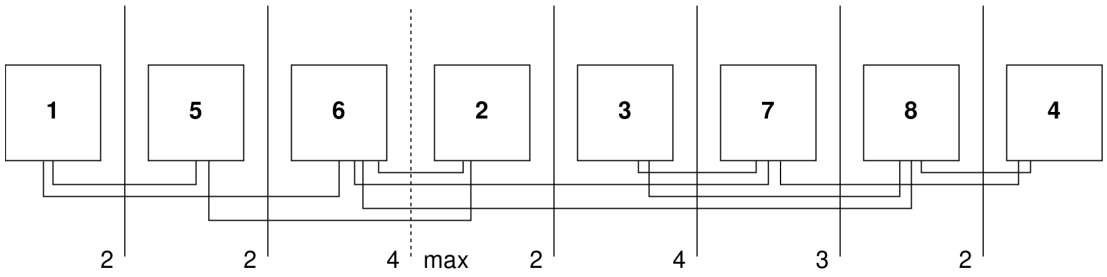
$$W(G,h) = \max_{i \in \{1,2,...,|V|\}} |\{\{u,v\} \in E : h(u) \leq i < h(v)\}|$$

The cutwidth $W(G)$ of a graph G refers to the minimum cutwidth $\min W(G,h)$ over all possible ordering functions h.

For a Circuit-CNF formula ϕ, the circuit graph $G_\phi = \{V, E\}$ is constructed such that V is equal to the set of variables V_ϕ occurring in ϕ and E contains an edge $\{u, v\}$, if a clause $c \in C_\phi$ contains variable u as a gate input and variable v as a gate output.



(a) Suboptimal ordering with *cutwidth* of 6



(b) Optimal ordering with *cutwidth* of 4

Fig. 3: Cutwidth for a 2-bit Ripple-Carry Adder with two different orderings.

Example 3. Figure 3 gives an intuitive view on the concept of cutwidth. It shows two different variable orderings for a 2-bit Ripple Carry Adder with its corresponding cutwidth. The variables $1, 2, 3$ and 4 represent the two inputs, where the odd numbers correspond to input a and the even numbers correspond to input b. Variables 5 and 6 are the two outputs for the first half adder, while 7 and 8 represent the first full adder component.
Comparing the two orderings, it can be observed that keeping variables of components which are connected locally close in the ordering decreases its cutwidth. For the given example, the second ordering is optimal, meaning that the cutwidth for the 2-bit RCA as a whole is equal to 4.

C. Polynomial Bounds

In general, the complexity of SAT is bounded exponentially by the number of variables n. The complete search tree for a SAT instance has 2^n paths, which serves as an upper bound for the time complexity of SAT. Since for arithmetic circuits, increasing the number of input bits also increases the number of variables in the corresponding Circuit-CNF at least linearly, this would result in an exponential complexity in the number of input bits.

As Prasad et al. presented in [21], it can be shown that for a SAT algorithm with static variable ordering, the complexity can be bounded exponentially in the *cutwidth* of the formula instead of the number of variables. For this, the concept of *Distinct Consistent Sub-Formulas* (DCSF) is introduced.

Definition 5. Given a Circuit-CNF formula ϕ, a *sub-formula* $\phi_{A_{V'}}$ is obtained by setting the values of variables $V' \subseteq V$ to a value $a \in \{0, 1\}$ determined by the solver. A sub-formula is called *consistent* if it does not contain empty clauses, as this would immediately lead to backtracking. As a *consistent sub-formula* $\phi_{A_{V'}}$ may occur in multiple positions in the search tree, which can be detected by the caching-based solver, we are only interested in *distinct consistent sub-formulas*.

From the Cache-SAT algorithm it is obvious that the search tree is bounded by the number of DCSFs δ, as all other paths are not explored further by the solver. δ in turn depends on the cutwidth of the formula.

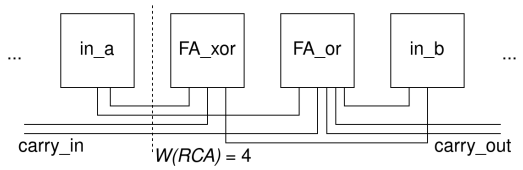
Lemma 1. Given a Circuit-SAT formula ϕ and a cut of size c between two adjacent variables in the formula's graph G_ϕ, the number of distinct consistent sub-formulas d_c which can be obtained by assigning their values is bounded by

$$d_c \leq 2^{2k_{fo}c}$$

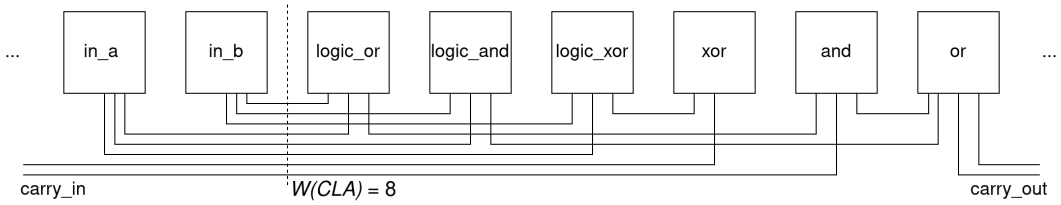
where k_{fo} is the maximum fan-out of the underlying circuit.

For further insight on the proof of Lemma 1, the reader is referred to [21].

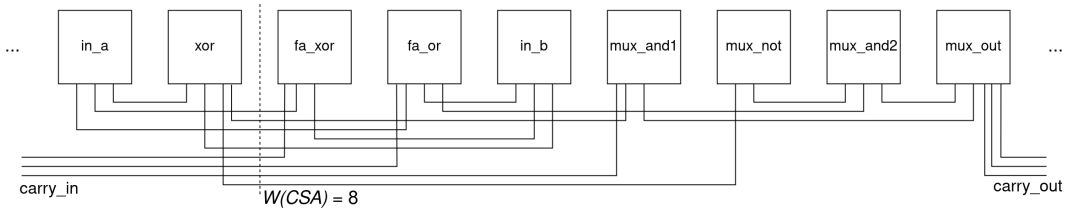
With this result at hand, an observation on the time complexity of the SAT procedure can be made. As pointed out before, the complexity is bounded by the total number of DCSFs δ, which can be derived from Lemma 1.



(a) Ordering for Ripple-Carrry Adder: $W(RCA) = 4$



(b) Ordering for Carry-Lookahead Adder: $W(CLA) = 8$



(c) Ordering for Carry-Skip Adder: $W(CSA) = 8$

Fig. 4: Cutwidth for different adder architectures

Theorem 1. Given a Circuit-CNF formula ϕ and a corresponding circuit graph G_ϕ with cuts c, a caching-based SAT solver can solve the instance ϕ in time $O(n * 2^{2k_{fo}W(G_\phi)})$, where n is the number of variables $|V|$ in ϕ.

Proof.

$$\begin{aligned}\delta &\leq \sum_c d_c \\ &\leq n * \max_c d_c \\ &\leq n * \max_c 2^{2k_{fo}c} \\ &= n * 2^{2k_{fo}W(G_\phi)}\end{aligned}$$

□

With this result, we can directly relate the time complexity for solving a Circuit-CNF formula with a caching-based SAT solver with its cutwidth, since it is the only quantity remaining in the exponent, along the constant maximum fan-out.
As can be derived from the time complexity $O(n*2^{2k_{fo}W(G)})$, a circuit with constant cutwidth in the number of the circuit's inputs results in a linear complexity, while a logarithmic cutwidth results in a polynomial complexity. For *Polynomial Formal Verification*, these circuit classes are of particular interest.

V. Cutwidth of Adder Circuits

We can now calculate the cutwidth of different adder circuits to obtain the time complexity of their verification. As was shown in Section IV-A, it is sufficient to consider the cutwidth of the circuit under verification, as the remaining clauses can be implied after fixing its variables. This means that a constant cutwidth of the circuit under verification results in a linear time complexity for the whole verification process. Note that the cutwidth calculation is done manually for the considered circuits, meaning no additional complexity is introduced into the verification flow. Figure 4 presents an overview of the cutwidth of the three adder architectures which are discussed in the following sections.

A. Ripple-Carry Adder

The ordering for the *Ripple-Carry Adder* is constructed by grouping together the adder cells between their connected primary inputs. As shown in Figure 4a, the maximum cut occurs between the full adder and its respective primary input. This results in a constant cutwidth of 4.

Theorem 1. The *Ripple-Carry Adder* can be verified with linear time complexity.

n	RCA (with implied clauses)		RCA		CLA		CSA	
	# DCSFs	Runtime in sec	# DCSFs	Runtime in sec	# DCSFs	Runtime in sec	# DCSFs	Runtime in sec
32	438518	230	316	0.30	378	0.42	378	0.68
64	T.O.	T.O.	636	1.11	762	1.66	762	2.65
128	T.O.	T.O.	1276	4.27	1530	6.52	1530	10.87
256	T.O.	T.O.	2556	17.31	3066	26.64	3066	45.49
512	T.O.	T.O.	5116	70.42	6138	107.05	6138	179.33
1024	T.O.	T.O.	10236	287.07	12282	441.17	12282	733.63

TABLE I: Experimental results for the three adder architectures

B. Carry-Lookahead Adder

The time complexity for the *Carry-Lookahead Adder* depends on the size of the propagate/generate logic block size. For the most simple implementation, we consider a block size of 1. The logic cells are grouped together with their connected primary inputs, so the cutwidth remains local within the respective block. The cutwidth for a single block is depicted in Figure 4b. The maximum cut occurs between the primary inputs and the carry-lookahead logic, resulting in a cutwidth of 8. While increasing the block size also increases the cutwidth of the CLA, for a constant block size, the cutwidth will remain constant, as it remains local within the given block.

Theorem 2. The *Carry-Lookahead Adder* can be verified with linear time complexity.

C. Carry-Skip Adder

Similarly to the CLA, the time complexity for the *Carry-Skip Adder* depends on the block size of the bypass logic. Once again, a block size of 1 is considered. The bypass logic cells and related multiplexers are grouped together with the connected inputs, with the maximum cut occurring between the bypass logic cells, as can be seen in Figure 4c. The cutwidth remains local within the bypass logic block at a constant of 8 and for a constant block size, the resulting cutwidth remains constant.

Theorem 3. The *Carry-Skip Adder* can be verified with linear time complexity.

VI. Experimental Results

To experimentally confirm our findings, we implemented our own SAT solver following the Cache-SAT algorithm outlined in Figure 2. As touched on in Section III-B, the algorithm is not expected to keep up with current state of the art solvers in terms of runtime, with the benefit of ensuring deterministic behavior. Thus, a comparison with competing methods like BDDs or ASP is omitted, as the purpose of these experiments is to show that the theoretical bounds can be observed in practice, not that our proposed approach outperforms other methods.

All miter circuits are generated using the ArithsGen tool [22], with a Ripple-Carry Adder as the reference implementation. The conversion of the circuits into CNF formulas is conducted using a custom script and the variable orderings are derived from the cutwidth calculations in Section V.

Table I shows the number of distinct consistent sub-formulas considered by the solver and the runtime in seconds for the different adder architectures with increasing input sizes. The first column illustrates the effectiveness of restricting the search space with the help of implied clauses. If the ordering includes variables of the miter logic in the search space, the number of DCSFs increases exponentially, making the verification infeasible for only 64 input bits.

When considering the optimal ordering explored in Section V, the linear behavior can be observed for all three adder architectures, Ripple-Carry Adder, Carry-Lookahead Adder and Carry-Skip Adder. The number of distinct consistent sub-formulas roughly doubles when the number of inputs is doubled, indicating a linear behavior in the number of inputs. For the RCA, the lower cutwidth results in a lower number of DCSFs across all input sizes. Since the CLA and CSA have the same cutwidth, the number of DCSFs is also the same for all input sizes.

The runtime of our SAT solver remains feasible for input sizes up to 1024 for all adder architectures. While the solving time increases with the number of inputs, there are no exponential increases in the runtime when the input size is doubled. Between the adders, the verification of the RCA is faster than for the other two because of the lower cutwidth. The runtime is larger for the CSA compared to CLA because of the greater number of variables, as each block contains 9 instead of 8 variables.

VII. Discussion & Future Work

Although our approach shows promising results in theory and practice, there are some improvements which can be investigated, especially in the SAT solver itself. Namely, two techniques were explored which are commonly employed to speed up modern implementations of SAT algorithms.

Early solvers like Chaff [23] saved a lot of time during *Boolean Constraint Propagation* (BCP) by replacing the simple unit propagation algorithm with an optimized version using the two watched literal scheme. Though this limits the number of clauses to be explored during BCP, during our evaluation only minor improvements could be observed in the largest instances, while smaller instances even produced longer runtimes due to the increased overhead of tracking the watched literals. We conjecture that the clauses in the instances we consider contain too few literals, so that the benefit of the two watched literal scheme is outweighed by many operations to update the watched literals.

The idea of parallelization was explored by pre-assigning some of the variables and solving the resulting sub-formulas in different execution threads. In our experiments, this yielded no improvements on the runtime. An explanation for this behavior lies in the polynomial verification approach itself. Each sub-instance created for a given thread contains fewer variables, but the cutwidth of the sub-instance itself does not change for our examples with constant cutwidth. Consequently, each execution thread will take a similar time to solve its sub-instance compared to the complete formula.

These evaluations confirm that the main factor influencing the runtime of our *CacheSAT* algorithm is indeed the cutwidth of the formula. Engineering the most efficient SAT solver with a static variable ordering for our cutwidth approach is not the main focus of this work, but may be explored in the future.

Adder circuits are considered as a case study for our approach, but the established polynomial bounds utilize a property which is inherent to every circuit. An interesting direction to be explored in future work in the context of PFV is the identification of further circuit classes with constant cutwidth. For BDDs, some circuit classes apart from integer adders were already considered [24] [25]. The authors conjecture that polynomial bounds for these circuit classes could also be established for SAT. Additionally, circuits which are difficult to verify using BDDs are of special interest.

VIII. Conclusion

In this paper, we have shown that SAT can ensure polynomial bounds on the time complexity for the verification of circuits with constant or logarithmic cutwidth. As the theoretical analysis and practical evaluation of all three considered adder architectures, Ripple-Carry Adder, Carry-Lookahead Adder and Carry-Skip Adder, have shown, the structure of the circuit has a direct impact on its cutwidth. If a circuit can be divided into separate logic blocks with few connections between these blocks, the cutwidth will remain local within the logic blocks, resulting in a constant cutwidth. With this characteristic in mind, future circuit classes with similar behavior may be identified, which are likely to exhibit a constant cutwidth as well. The field of *Polynomial Formal Verification* using SAT provides a large variety of directions to be explored in future work.

Acknowledgements

This work was supported by the German Research Foundation (DFG) within the Project PolyVer (DR 287/36-1).

References

[1] R. Drechsler *et al.*, *Advanced Formal Verification*. Springer, 2004.

[2] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: a survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 4, p. 123–193, Apr. 1999.

[3] 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.

[4] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Theory in practice for system design and verification," *ACM SIGLOG News*, vol. 2, p. 46–51, jan 2015.

[5] 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*, pp. 114–121, 2001.

[6] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), p. 151–158, Association for Computing Machinery, 1971.

[7] 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.

[8] 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.

[9] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to Combinational Equivalence Checking," in *2006 IEEE/ACM International Conference on Computer Aided Design*, (Double Tree Hotel, San Jose, CA,USA), p. 836–843, IEEE, Nov. 2006.

[10] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *International Conference on Theory and Applications of Satisfiability Testing*, 2003.

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

[12] F. R. K. Chung, "On the Cutwidth and the Topological Bandwidth of a Tree," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, p. 268–277, Apr. 1985.

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

[14] R. Drechsler, A. Mahzoon, and L. Weingarten, "Polynomial Formal Verification of Arithmetic Circuits," in *Proceedings of International Conference on Computational Intelligence and Data Engineering* (N. Chaki, N. Devarakonda, A. Cortesi, and H. Seetha, eds.), (Singapore), pp. 457–470, Springer Nature Singapore, 2022.

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

[16] M. Gelfond and V. Lifschitz, "The Stable Model Semantics for Logic Programming," in *Proceedings of International Logic Programming Conference and Symposium* (R. Kowalski, Bowen, and Kenneth, eds.), pp. 1070–1080, MIT Press, 1988.

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

[18] J. Gu, P. Purdom, J. Franco, and B. Wah, *Algorithms for Satisfiability (SAT) problem: a survey*, vol. 35, pp. 19–152. 12 1997.

[19] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT Solvers to Cryptographic Problems," in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings* (O. Kullmann, ed.), vol. 5584 of *Lecture Notes in Computer Science*, pp. 244–257, Springer, 2009.

[20] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*, pp. 466–483. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983.

[21] M. Prasad, P. Chong, and K. Keutzer, "Why is Combinational ATPG Efficiently Solvable for Practical VLSI Circuits?," *J. Electronic Testing*, vol. 17, pp. 509–527, 12 2001.

[22] 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.

[23] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pp. 530–535, 2001.

[24] 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, Apr. 2023.

[25] M. Schnieber and R. Drechsler, "Polynomial Formal Verification of KFDD Circuits," in *Proceedings of the 21st ACM-IEEE International Conference on Formal Methods and Models for System Design*, (Hamburg Germany), p. 82–89, ACM, Sept. 2023.