Polynomial Formal Verification of Sequential Circuits using Weighted-AIGs

Mohamed Nadeem[ψ], Chandan Kumar Jha[ψ], Rolf Drechsler[ψ,†]
University of Bremen, Bremen, Germany[ψ]
DFKI GmbH, Bremen, Germany[†]
mnadeem@uni-bremen.de, chajha@uni-bremen.de, drechsler@uni-bremen.de

***Abstract*—Ensuring the functional correctness of a digital system is achievable through formal verification. Despite the increased complexity of modern systems, formal verification still needs to be done in a reasonable time. Hence, *Polynomial Formal Verification* (PFV) techniques are being explored as they provide a guaranteed upper bound on the run time for verification. Recently, it was shown that combinational circuits characterized by a constant cutwidth can be verified in linear time using *Answer Set Programming* (ASP). However, most of the designs used in digital systems are sequential. Hence, in this paper, we propose a *linear time* formal verification approach using ASP for sequential circuits with constant cutwidth. We achieve this by proposing a new data structure called *Weighted-And Inverter Graph* (W-AIG). Unlike existing formal verification methods, we prove that our approach can verify any sequential circuit with a constant cutwidth in a linear time. Finally, we also implement our approach and experimentally show the results on a variety of sequential circuits like pipelined adders, serial adders, and shift registers to confirm our theoretical findings.**

***Index Terms*—Polynomial Formal Verification, Sequential Circuits, Pipelined Adders, Cutwidth, Answer Set Programming.**

I. Introduction

Ensuring the functional correctness of circuits is one of the most challenging tasks in digital system design. These circuits are mostly sequential in nature [1]–[3], i.e., the output function is determined by both the current and the previous inputs. Given their ubiquitous use in modern systems, ensuring the correctness of sequential circuits becomes an imperative task [4]–[6]. Thus, formal verification methods have gained considerable attention to determine whether these circuits behave as desired [7], [8]. There exist several techniques for achieving efficient verification of sequential circuits using *Binary Decision Diagrams* (BDDs) [9], and *Bounded Model Checking* (BMC) [10]. These methods unroll the sequential circuit up to several clock cycles (also called *Time Frames*) (TF) to guarantee its correctness. In [11], the authors propose a scalable method for sequential verification based on partitioning the Boolean function. The Boolean function is represented as a *Directed Acyclic Graph* (DAG), with its gates arranged in topological order. Verification is then performed inductively on the circuit up to the desired TF. In this context, *And-Inverter Graphs* (AIGs) [12] serve as DAGs to provide an efficient representation of the circuits. However, this approach is limited to acyclic AIGs, as the topological order is not preserved in the cyclic AIGs. In addition to this, there is no guarantee on the runtime of the approach proposed in [11].

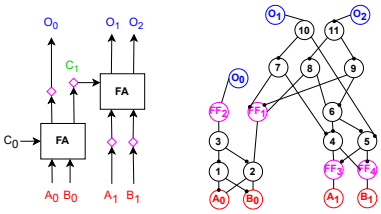
Despite the success of these methods in verifying sequential circuits, they fail to provide any time bounds for the verification process. Thus, *Polynomial Formal Verification* (PFV) [13] has been introduced to guarantee an upper bound on the time required for the verification process. Recently, there have been works that have explored the PFV of sequential circuits [14], [15] using BDD. These methods have demonstrated that the verification of some of the sequential circuits is bounded by polynomial time. However, these works were either limited to specific types of sequential circuits, such as counters in [14], or restricted to a single clock cycle TF in [15].

In this work, we use *Answer Set Programming* (ASP) [16]–[18] for the verification of a large variety of sequential circuits and show stricter than polynomial bound, i.e., linear bound on the runtime. ASP is a declarative programming framework that is well-known in the area of knowledge representation and non-monotonic reasoning [19]. It is used to solve difficult search problems while allowing compact modeling. Recently, it has been demonstrated that the verification of combinational circuits with a constant *Cutwidth* [20] of the AIG (e.g., *Ripple Carry Adder* (RCA), *Carry Look-ahead Adder* (CLA), and *Carry Skip Adder* (CSKA)) can be achieved in linear time using ASP [21]. However, extending this approach to the domain of sequential circuits remains a challenging task due to the limitations of AIG in encoding sequential circuits. This arises from its inability to encode the time frames in the AIG nodes.

We alleviate this limitation of AIG and introduce a new data structure called *Weighted-AIG* (W-AIG) that allows encoding time frames. Unlike [11], we exploit the concept of *Simple Paths* [22] to handle both acyclic as well as cyclic AIGs. We introduce a new method for unrolling the sequential circuit w.r.t. W-AIG and the TF, where TF is defined based on the number of input vectors passed to the circuit. Our approach incorporates the concept of cutwidth in conjunction with W-AIG to partition the circuit into subcircuits, each corresponding to a specific time frame. Subsequently, the ASP solver *Clingo* [23] is employed to verify each subcircuit independently. This approach effectively reduces the overall time complexity to the cutwidth of the circuit. We implemented the proposed methodology and analyzed *Pipelined Adders* having a constant cutwidth. In addition to this, we also show other widely used sequential circuits (i.e., *Shift Registers*, and *Serial Adders*) that are verifiable in polynomial time [14], [15], can now be verified in linear time using our proposed approach.

Contributions: *A) We introduce Weighted-AIG (W-AIG) as a new data structure that enables encoding time frames of acyclic and cyclic AIGs. B) We present an innovative PFV approach for sequential circuits with constant cutwidth using ASP in linear time. C) We conduct an empirical evaluation to show the linear time verifiability for sequential circuits with constant cutwidth using ASP.*

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



(a) Pipeline RCA (b) AIG

Fig. 1: The Pipeline RCA and its corresponding AIG graph.

II. Preliminaries

A. Pipelined Adder

Pipelining was introduced to increase the performance of digital circuits and is ubiquitous in modern processors and accelerators [24], [25]. It is achieved by inserting flip flops between combinational circuits, which increases the frequency of operation [26]. We have used an example of a 2-bit RCA, which has two *Full Adders* (FAs), to explain the pipelining methodology. We see from Fig. 1(a), that the flip flops (red diamond) are inserted so that the output of the RCA is obtained after two cycles. In the first cycle, $A_0 + B_0 + C_0$ is done using the first FA, and the outputs (O_0, C_1) are stored in the flip flop at the end of the first cycle. In the second cycle, $A_1 + B_1 + C_1$ are added using the second FA, which generates the remaining outputs (O_1, O_2). Hence, the numbers are added in two cycles.

B. Cutwidth as a Property of AIG

Let A be a circuit design. Then, A can be seen as a directed graph G, which consist of terminal nodes (i.e., inputs PI and outputs PO), and non-terminal nodes (i.e., And, FF, and Inv gates). This can be formulated as follows:

Definition 1 (AIG Graph)**.** *Let $G = (V, E)$ be an AIG of the circuit design A such that:*

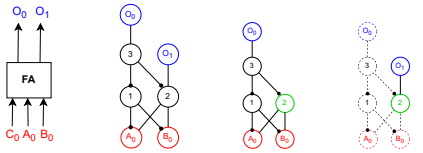
- $V = \{v \mid v$ *is a gate*$\}$.
- $E = \{(v, v') \mid v, v' \in V,\ v$ *is an input of* $v'\}$.

Given the block diagram shown in Fig. 2(a), the AIG graph can be obtained as shown in Fig. 2(b) (C_0 is taken to be 0 hence it does not appear in the AIG).

Intuitively, given an AIG $G = (V, E)$ of size n with $v_0, ..., v_n \in PO$ ordered output gates (i.e., v_i appears before v_{i+1}, where $0 \leq i < n$), the cutwidth of G is the smallest number K of edges required to be removed to partition G into n subgraphs $\sigma = \{G_1, ..., G_n\}$ such that each subgraph G_i has at most K out-going edges.

Each subgraph (G, O_i) can be constructed w.r.t. G by starting from the output gate O_i, and traversing all its reachable nodes. The subgraphs (G, O_0) of Fig. 2(c) and (G, O_1) of Fig. 2(d) are constructed w.r.t. G of Fig. 2(a). Since each subgraph might contain several nodes that appear in more than one subgraph (e.g., node 2 appears in Fig. 2(c) and Fig. 2(d)), it is essential to reduce these subgraphs to ensure that they have disjoint edges. This is achieved by making a copy of that node in each of the required subgraphs. The resulting disjoint subgraphs G_i are called as *Reduced Subgraphs*.

We refer by CO_i and CI_i to the set of out-going and in-going nodes (also called *Non-primary Inputs*) induced by edge-cuts such that CO_i contains the nodes that are stored and passed to other subgraphs, while CI_i contains nodes coming from previous subgraphs. Thus, the subgraph G_i is evaluated by populating the non-primary inputs CI_i together with the primary inputs PI_i appearing in G_i. It is important to highlight that the values of the non-primary inputs CI_i are obtained by concatenating the values of out-going nodes $CO_0, \ldots, CO_{i-1}$. Let $IN_i = PI_i \cup CI_i$ be the inputs of the subgraph G_i.



(a) FA (b) AIG (c) G_0 (d) G_1

Fig. 2: Dotted nodes and edges are removed when the subgraphs are reduced.

The graph G is said to *K-Bounded Graph* if it can be partitioned into $G_0, \ldots, G_n$ reduced subgraphs such that each subgraph G_i has at most K inputs (i.e., $|IN_i| \leq K$, where $0 \leq i \leq n$). Given the reduced subgraphs G_0 and G_1 of Fig. 2(c) and Fig. 2(d), the graph G is a 2-bounded graph. This due to the fact that G_0 has two inputs (i.e., $IN_0 = \{A_0, B_0\}$, where $CI_0 = \emptyset$ and $PI_0 = \{A_0, B_0\}$) and G_1 has one input ($IN_1 = \{2\}$, where $CI_1 = \{2\}$ and $PI_1 = \emptyset$).

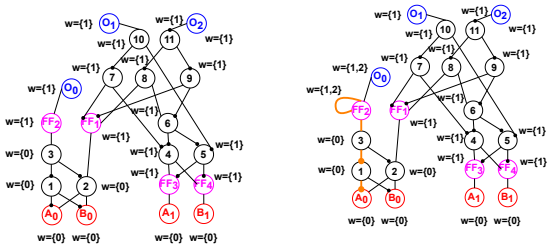
C. Answer Set Programming

ASP is a declarative framework [27], [28] widely used to solve difficult NP-hard search problems [29]. These search problems are reduced for computing *Answer Sets* [30]. We follow the standard definitions of ASP [31], [32]. In the context of circuit design, the basic idea of ASP is to encode an AIG graph G as a logic program Π, together with its specification functions. Then, a query Q (represented by a set of facts) representing values of circuit inputs, is added to the program Π (denoted by Π^Q). The Clingo solver is used to check whether an answer set of Π^Q exists [33]. If the answer set exists, the query Q matches the program Π (i.e., Q is a *Valid Input*). Otherwise, Q is invalid. We refer by $\mathcal{Q}$ to the set of all queries. The graph G is said to be a *Valid Graph* if for every $Q \in \mathcal{Q}$, Q is a valid input.

III. W-AIG Data Structure

In this section, we introduce *W-AIG* as a new data structure that extends AIG by labelling each gate with weights that represent the time frames $1, \ldots, TF$ in which the value of the gate has to be verified. The basic idea is to assign a set of weights (representing time frames) to each gate. Intuitively, the set of weights for a gate is computed from inputs PI to outputs PO. For any input node $v \in PI$, we have that the set of weights $w(v) = \{0\}$ (indicating that the value appears in the same time frame in which it is passed). For a non-terminal node n, the set of weights is either incremented by 1, if $v \in FF$, or it remains unchanged. Hence, if the values are passed to input gates at time frame $t = 1$, then all gates appearing before $v \in FF$ will appear in the same time frame, and those appearing after $v \in FF$ will appear at $t = 2$, similar to the actual operation of the sequential circuits.

We start by constructing all possible paths from an input $v \in PI$ to any other non-input node $v' \notin PI$. Given an



(a) G (b) G'

Fig. 3: The W-AIG G is then constructed w.r.t. the AIG of Fig. 1(b) and the modified (W-AIG) AIG G' of Fig. 3(a). Nodes highlighted in red, blue, and magenta correspond to inputs, outputs, and latches, respectively. The edges highlighted in orange correspond to the infinite path appearing in G'.

AIG $G = (V, E)$, and a node $v \in PI$, ρ is a *Path* such that $\rho := \{v, v_1, ..., v_n \mid (v, v_1), ..., (v_{n-1}, v_n) \in E\}$. Let $V(\rho)$ and $E(\rho)$ be the set of nodes and edges appearing in the path ρ. Since sequential AIG can be cyclic, there may exist an infinite path (a path with an infinite number of nodes). Consider the modified AIG as shown in Fig. 3(b), there exists an infinite path $A_0, 1, 3, FF_2, FF_2, ...$ (i.e., the path will keep looping to FF_2) obtained from the input A_0. To overcome this problem, it is required to restrict the path to be a simple path. A path ρ is said to be a *Simple Path* if every two edges $e, e' \in E(\rho)$, it holds that $e \neq e'$. By the previous definition, the simple path ρ contains only distinct edges and, consequently, has no loops.

This allows us to define the weight function w of a node $v \in V(\rho)$ w.r.t. the simple path ρ (denoted by $w(v, \rho)$) as follows:

$$w(v,\rho) = \begin{cases} w(v',\rho) + 1, & \text{if } v \in FF, (v',v) \in E(\rho), v' \in V(\rho). \\ w(v',\rho), & \text{if } v \notin FF, (v',v) \in E(\rho), v' \in V(\rho). \\ 0, & \text{if } v \in PI. \end{cases} \quad (1)$$

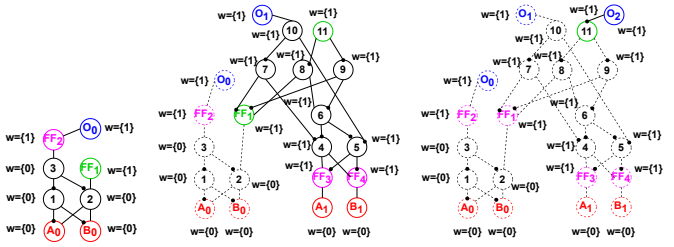
Consider the AIG of Fig. 3(a). It has five simple paths. For instance, the path $\rho = A_0, 1, 3, FF_2, O_0$ can be constructed starting from the input node A_0. Similarly, consider the modified AIG of Fig. 3(b). It also has five simple paths. For instance, the path $\rho' = A_0, 1, 3, FF_2, FF_2, O_0$ can be constructed starting from the input node A_0. The simple path requires all edges to be disjoint, but a node might appear twice in the simple path (e.g.,$\rho' = A_0, 1, 3, FF_2, FF_2, O_0$ in Fig. 3(b)).

Considering the path $\rho = A_0, 1, 3, FF_2, O_0$ obtained from the AIG of Fig. 3(a). It holds that $w(O_0, \rho) = 1$, as this path has only one flip-flop FF_2 appearing before O_0, while $w(A_0, \rho) = 0$. Thus, the weights allow us to track the time frame at which a gate needs to be checked. This means that if a value is passed at time frame i on node A_0, it will appear on node O_0 in time frame $i + w(O_0, \rho)$.

Now, let ϱ be the set of all simple paths, obtained from any input $v \in V \cap PI$. Since there may exist several weights for the same node, let $w(v)$ be the set of all weights of the node v obtained from any path $\rho \in \varrho$. This allows us to define the *Weighted-AIG* (W-AIG) w.r.t. ϱ as follows:

Definition 2 (W-AIG)**.** *Let $G' = (V', E')$ be an AIG graph. Then, the W-AIG $G = (V, E, \mathcal{W})$ is a weighted directed graph of $G' = (V', E')$ such that:*

- $V = V'$.
- $E = E'$.
- $\mathcal{W} = \{(v, w(v)) \mid v \in V\}$.



(a) G_0 (b) G_1 (c) G_2

Fig. 4: The nodes highlighted in red, blue, magenta, and green correspond to inputs, outputs, flip flops, and out-going (in-going) nodes, respectively. Dotted nodes and edges are removed when the subgraphs are reduced.

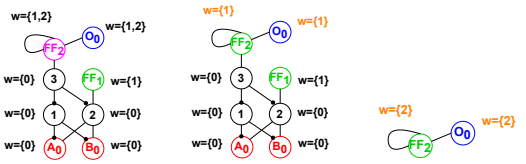
The W-AIG G and G' are constructed w.r.t. the simple paths as shown in Fig. 3(a) and Fig. 3(b), respectively. The set of weights of a node $v \in V$ guarantees that no two values are passed to node v in the same time frame t. Considering the modified W-AIG G' of Fig. 3(b), and suppose all values are passed to inputs at time frame 1. Then, the resulting value of O_0 will appear at time frames 2 and 3, i.e., it needs to be checked for both time frames. The functional correctness is established by verifying whether the output value at time frame t matches its specification for the passed input values at time frame t'. Here $t' \leq t \leq TF+1$, where TF is the number of time frames in which the values of IN are passed.

IV. PFV of Sequential Circuits

The PFV approach consists of five main steps: **Step 1** begins by extending the AIG to a W-AIG. **Step 2** constructs the output-based subgraphs $G_0, \ldots, G_n$ with respect to the W-AIG. **Step 3** addresses the fact that each subgraph G_i might contain more than a single time frame, which needs to be unrolled. Thus, it is essential to partition each subgraph G_i, allowing only a single time frame. Therefore, each subgraph G_i is further divided into smaller ones, each representing a single weight (time frame) within the subgraph (referred to as a *Weight-based Subgraph*) (Section IV-A). **Step 4** involves unrolling each weight-based subgraph w.r.t. the number of input vectors, VW (Section IV-B). **Step 5** consists of two tasks: **1) Information Passing**: Proposing a method for storing and passing outgoing nodes between the unrolled subgraphs (Section IV-C). **2) Verification**: Encoding the unrolled subgraph as an ASP logic program, as described in [34]. The ASP solver is then employed to verify the unrolled graph. If the verification is valid, the solver determines the values of the outgoing nodes within their specified time frames for use in other unrolled subgraphs (Section IV-D). Finally, **Step 5** is repeated until all unrolled subgraphs are verified.

A. Graph Splitting

Given the W-AIG of Fig. 3(a), the reduced subgraphs G_0, G_1, and G_2 are constructed as shown in Fig. 4(a), Fig. 4(b), and Fig. 4(c). Considering the subgraph G_1, we have $CI_1 = \{FF_1\}$ and $CO_1 = \{11\}$. To ensure that each subgraph G_i contains at most one time frame t, and hence, can be unrolled for only one time frame, it is essential to split each G_i into a set of subgraphs $G_{i,j}$ (called *Weight-Based Subgraph*) such that each subgraph $G_{i,j}$ represent at most one time frame. If the subgraph contains more than one time frame, it implies that



(a) G'_0 (b) $G'_{0,1}$ (c) $G'_{0,2}$

Fig. 5: The subgraph G'_0 defined w.r.t. the modified WAIG G' of Fig. 3(b), and the resulting weight-based subgraphs $G'_{0,1}$ and $G'_{0,2}$. The weights highlighted in orange are the resulting weights after G'_0 is split.

inputs appear in different time frames, thereby increasing the number of inputs in the subgraph which is not desired.

Let $\mathcal{W}(G_i)$ be the sets of all weights appearing in the graph G_i. Similarly, let $\mathcal{W}(IN_i)$ be the set of weights of the inputs IN_i w.r.t. G_i. To split the subgraph G_i into smaller ones, each representing a single time frame t, it is essential to determine the number of time frames appearing in G_i. Let $len(G_i)$ be the difference between the largest and the smallest weight appearing in G_i (i.e., $len(G_i) = max(\mathcal{W}(G_i)) - min(\mathcal{W}(G_i))$). The weight-based graph splitting is defined as follows:

Definition 3 (Weight-based Graph)**.** *Let $G_i = (V_i, E_i, \mathcal{W})$ be a reduced subgraph, and j be an integer such that $0 < j \leq len(G_i)$. Then, the graph $G_{i,j} = (V_{i,j}, E_{i,j}, \mathcal{W}_{i,j})$ is a Weight-based Subgraph of G_i such that:*

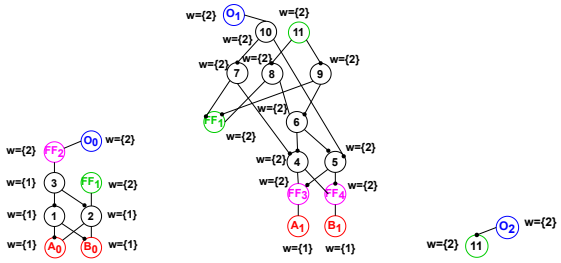
- $V_{i,j} := \{v \in V_i \mid w(v) \cap \{j-1, j\} \neq \emptyset, (v, w(v)) \in \mathcal{W}_i\}$.
- $E_{i,j} := \{(v, v') \in E_i \mid v, v' \in V_{i,j}\}$.
- $\mathcal{W}_{i,j} := \{(v, w'(v)) \mid, w'(v) \subseteq w(v) \cap \{j-1, j\}, (v, w(v)) \in \mathcal{W}_i, v \in V_{i,j}\}$.

By the previous definition, if the subgraph G_i has time frames $0, 1$, and 2, it can be partitioned into two subgraphs $G_{i,1}$ and $G_{i,2}$. For $G_{i,1}$, then $j = 1$. Therefore, all nodes with the weight 0 or 1 are selected with their edges, and the weights of these nodes are restricted to values 0 and 1. Similarly, $G_{i,2}$ contains nodes that have weights 1 or 2, and their weights are restricted to values 1 and 2. This allows us to guarantee that each subgraph $G_{i,j}$ to be unrolled at most one time frame t. Considering the subgraph G_0 of Fig. 4(a). As $len(G_0)$ is exactly 1 (i.e., $len(G_0) = 1 - 0 = 1$), then, G_0 can only be split into one subgraph $G_{0,1}$ that is identical to G_0. The same is true for $G_{1,1}$, and $G_{2,1}$.

To show a case of having more than one weight-based subgraph, consider the modified subgraph G'_0 of Fig. 5(a). Since $len(G'_0) = 2 - 0 = 2$, then G'_0 is split into two subgraphs $G'_{0,1}$ and $G'_{0,2}$ as shown in Fig. 5(b) and Fig. 5(c), respectively. For $G'_{0,1}$, where $j = 1$, only nodes with weights 0 and 1 are selected, restricting all nodes to these weights. The weight-based subgraph $G'_{0,2}$ is defined analogously.

B. Weight-based Graph Unrolling

Intuitively, the circuit is unrolled w.r.t. the number of input vectors passed to it. Consequently, the number of time frames can be computed accordingly, and the circuit is unrolled up to the computed number of time frames. We refer by *Vector Width* (VW) to the number of input vectors. The number of time frames can be calculated w.r.t. the vector width VW and the WAIG $G = (V, E, \mathcal{W})$ as shown in the following corollary.



(a) $G^1_{0,1}$ (b) $G^1_{1,1}$ (c) $G^2_{2,1}$

Fig. 6: The nodes highlighted in red, blue, magenta, and green correspond to inputs, outputs, flip flops, and out-going (in-going) nodes, respectively.

Corollary IV.1. *(Number of Time Frames) Let VW be the vector width, and $max(\mathcal{W}(G))$ be the maximum weight of $G = (V, E, \mathcal{W})$. Then, the number of time frames $TF = VW * max(\mathcal{W}(G))$.*

Given a time frame t, each weight-based subgraph $G_{i,j}$ is unrolled. We refer by $G^t_{i,j}$ to the unrolled subgraph of $G_{i,j}$ under time frame t. The graph unrolling is defined as follows:

Definition 4 (Unrolled Graph)**.** *Let $G_{i,j} = (V, E, \mathcal{W})$ be a weight-based subgraph, t be a positive integer such that $1 \leq t \leq VW$. Then, a graph $G^t_{i,j} = (V^t, E^t, \mathcal{W}^t)$ is the unrolled graph of $G_{i,j}$ w.r.t. t such that:*

- $V^t = V$.
- $E^t = E$.
- $\mathcal{W}^t = \{(v, w(v) + min(\mathcal{W}(IN)) + t) \mid (v, w(v)) \in \mathcal{W}\}$.

Intuitively, every gate will either appear at the same time frame as the inputs or appear in the next time frame. Therefore, it is essential to compute the least time frame of the inputs $min(\mathcal{W}(IN))$. Then, based on the number of input vectors, the time frame t is computed and added to $min(\mathcal{W}(IN))$. Finally, if there is no flip flop between the node v and the inputs, the weight $w(v)$ of v is equal to $\min(\mathcal{W}(IN))$, and thus, they appear in the same time frame. Otherwise, the node v will appear in the next time frame.

For example, consider $G_{1,1}$ of Fig. 4(a). We refer by $w^t(v)$ to the set of weights of node v appearing in the unrolled graph $G^t_{i,j}$. Since $G_{1,1}$ has three inputs $IN_{1,1}$ (i.e., $PI_{1,1} = \{A_1, B_1\}$ and $CI_{1,1} = \{FF_1\}$) with different weights (e.g., $w(A_1) = \{0\}$ and $w(FF_1) = \{1\}$), then $min(\mathcal{W}(IN_{1,1})) = \{0\}$. Suppose that $t = 1$, then the values of A_0 and B_0 will appear at time frame 1 (e.g., $w(A_0) = \{0\}$, and $w^t(A_0) = 0 + 0 + 1 = \{1\}$), and the value of FF_1 will appear at time frame 2 (e.g., $w(FF_1) = \{1\}$, and $w^t(FF_1) = 1 + 0 + 1 = \{2\}$). It is important to notice that unrolling does not increase the number of inputs. Consider $G_{0,1}$ of Fig. 4(a), and suppose that $VW = 1$. Then, the unrolled subgraphs $G^1_{0,1}$, $G^1_{1,1}$, and $G^2_{2,1}$ are constructed as shown in Fig. 6(a), Fig. 6(b), and Fig. 6(b), respectively (recall Definition 4).

C. Information Passing w.r.t. Unrolled Weight-based Subgraphs

Intuitively, the set of out-going nodes $CO^{t'}_{i,j}$ is evaluated in the unrolled subgraph $G^t_{i,j}$ to be used in other subgraphs $G^{t'}_{i',j'}$, where $i' \geq i$, $j' \geq j$, and $t' \geq t$. Hence, the values of $CO^{t'}_{i,j}$ must be stored w.r.t. the time frame t', in which they appear. These values cannot be stored as a function of the primary

inputs $PI^t_{i,j}$ because doing so would require passing $PI^t_{i,j}$ to other subgraphs, resulting in an exponential expansion of the search space. Instead, we store the values of $CO^{t'}_{i,j}$ w.r.t the carry function $carry_i$ under the same time frame t'. This allows us to utilize the carry in the specification of the output O_i.

Therefore, we introduce two mapping functions f and g. We start by defining a mapping function f that maps each set of input values $s \in IN_{i,j}$ to a set of values $COUT_{i,j}$ of out-going nodes under time frame $t+1 \in TF$ of the unrolled graph $G^t_{i,j}$.

$$f\colon IN^t_{i,j} \mapsto COUT^{t+1}_{i,j} \tag{2}$$

The function g maps each set of values $s' \in COUT^{t+1}_{i,j}$ to the value of the carry function $carry_i$ at the same time frame $t+1$.

$$g\colon COUT^{t+1}_{i,j} \mapsto [0,1]^{t+1}. \tag{3}$$

To illustrate the previous Eq. (2), let us consider $G^1_{0,1}$ of Fig. 6(a). Then, $IN^1_{0,1} = \{A_0, B_0\}$ has four possible combinations that need to be evaluated, while it has only one out-going node $CO^2_{0,1} = \{FF_1\}$. By Eq. (2), the relation between the inputs and the out-going nodes is defined such that $f(0,0) = \{0\}$, $f(1,1) = \{1\}$, $f(0,1) = \{0\}$, and $f(1,0) = \{1\}$. Then, Eq. (3) is used to define the relation between the out-going nodes and the carry function $carry_i$ such that $g(0) = \{0\}$, and $g(1) = \{1\}$. Hence, it is sufficient to store only the relation between functions f and g, instead of storing their values w.r.t. the primary inputs $IN^1_{0,1}$. To store these values, we construct a hash table $\mathcal{X}^{t+1}_{i,j}$ to store the values as follows:

$$\mathcal{X}^{t+1}_{i,j} = \{(f(s), g(f(s))) \mid s \in IN^t_{i,j}\}. \tag{4}$$

To illustrate the previous equation, the inputs $IN^t_{i,j}$ are passed at time frame t, while the values of out-going nodes appear at time frame $t+1$. E.g., the values of $IN^1_{0,1} = \{A_0, B_0\}$ are passed at time frame 1, and these values appear on the out-going nodes $CO^2_{0,1} = \{FF_1\}$ at time frame 2.

D. Unrolled Subgraph Verification

Two main tasks must be performed for each unrolled subgraph $G^t_{i,j}$. The first task is to check whether $G^t_{i,j}$ is a valid graph (recall Section II-C). The second task is to construct the table $\mathcal{X}^{t+1}_{i,j}$ to be used in $CI^{t+1}_{i',j'}$ of the unrolled graph $G^{t+1}_{i',j'}$. This is due to the fact that $IN^{t+1}_{i',j'}$ may contain primary inputs $PI^t_{i',j'}$ at time frame t, and non-primary inputs $CI^{t+1}_{i',j'}$ at time frame $t+1$ (e.g., $CI^2_{1,1} = \{FF_1\}$, and $PI^1_{1,1} = \{A_1, B_1\}$ w.r.t. $G^1_{1,1}$ of Fig. 6(b)).

As the values of $CI^{t+1}_{i',j'}$ may be a concatenation of multiple tables (in case the values of $CI^{t+1}_{i',j'}$ are obtained from several unrolled subgraphs at time frame t), we define a relation $\bowtie$ that allows us to concatenate different tables, behaving as an inner join if the tables have common nodes, and as a cross join otherwise. Let $\mathcal{X}^{t+1}_{i',j'}(CI^{t+1}_{i',j'})$ be the resulting table. Hence, the resulting table is populated with the values of $PI^t_{i',j'}$ (e.g., table $\mathcal{X}^2_{1,1}(CI^2_{1,1})$ is populated with $PI^1_{1,1} = \{A_1, B_1\}$).

Through this approach, the search space for each subgraph $G^t_{i,j}$ drops to $2^{IN^t_{i,j}}$ Consequently, the overall complexity for the verification is reduced to $\mathcal{O}(N \cdot 2^K)$, where N is the number of all unrolled subgraphs, and K represents the maximum number of inputs obtained from all weight-based subgraphs $G_{i,j}$. This is in contrast to the complexity 2^{TF*n} illustrated in Section III.

V. Proofs and Experimental Results

A. Time Complexity

We refer by $\Pi(G^t_{i,j})$ to the logic program constructed w.r.t. the unrolled subgraph $G^t_{i,j}$ and time frame t, and by $\Pi(G)$ to the logic program constructed w.r.t. the WAIG graph G. Then, checking the graph validity of $\Pi(G^t_{i,j})$ depends on the number of its input nodes $IN^t_{i,j}$ (recall Section II-C). Thus, the verification time of the unrolled graph $G^t_{i,j}$ is characterized in the following theorem.

Theorem V.1. *Let G^t be an unrolled subgraph under a time-frame t. Then, $\Pi(G^t)$ can be verified in time $\mathcal{O}(2^{|IN|})$, where IN is the number of inputs appearing in G^t.*

Proof: Let G^t be the unrolled subgraph w.r.t. time-frame t. Then, G^t is a valid graph iff for each input value $q \in Q$, it holds that q is a valid input, where Q is the set representing all possible input values. Hence, the $\Pi(G^t)$ is bounded by the size of Q. The size of Q depends on the number of inputs of G. Let IN be the set of inputs representing the primary inputs PI and the in-going nodes CI appearing in G^t. Consequently, it holds that $|Q| = 2^{IN}$. Hence, $\Pi(G^t)$ is bounded by 2^{IN}, and consequently, $\Pi(G^t)$ is verified in time $\mathcal{O}(2^{|IN|})$. ■

To analyze the overall complexity of our approach, it is essential to compute the complexity required to construct the WAIG $G = (V, E, \mathcal{W})$ from the AIG graph $G' = (V', E')$. For the computation of weights, which includes constructing simple paths in G', we employ a *Depth First Search* (DFS) approach. It has been shown that the complexity of DFS is $\mathcal{O}(|V'| + |E'|)$ [35].

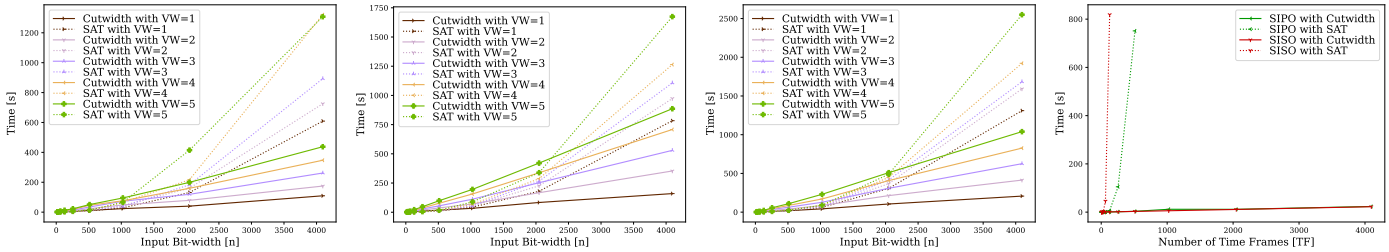
Since the unrolled subgraph $G^t_{i,j}$ may contain in-going nodes $CI^t_{i,j}$, these values are obtained from $\mathcal{X}^t_{i,j}(CI^t_{i,j})$ (recall Section IV-C). We assume that $\mathcal{X}^t_{i,j}(CI^t_{i,j})$ can be computed in constant time. This is because the search operation of a hash table may take a linear time in the worst case [36].

Finally, the overall complexity of our PFV approach is characterized as follows.

Theorem V.2. *Let $G = (V, E, \mathcal{W})$ be a WAIG constructed w.r.t. the AIG G'. Then, G can be verified in time $\mathcal{O}(N \cdot 2^K)$, where N is the number of all unrolled subgraphs and K is the maximum size of input nodes over all unrolled subgraphs.*

Proof: Let $G = (V, E, \mathcal{W})$ be a W-AIG of G', where I is the number of outputs. Then, l the output-based subgraphs are constructed from G. As each subgraph $i \in I$ may contain more than one weight, then J weight-based subgraphs have to be constructed from each subgraph $i \in I$ (recall Definition 3). This results in $G_{0,1}, ..., G_{I,J}$ weight-based subgraphs. Finally, each $G_{i,j}$ is unrolled w.r.t. time frames TF (recall Definition 4), where TF is computed as shown in Corollary IV.1. Let $G^t_{i,j}$ be the unrolled subgraph at time-frame t such that $1 \leq t \leq TF$. Also, let $IN^t_{i,j}$ be the set of inputs appearing $G^t_{i,j}$. Hence, $\Pi(G^t_{i,j})$ can be verified in time $\mathcal{O}((2^{|IN^t_{i,j}|})$ by Theorem V.1.

As the graph unrolling does not change the number of inputs of the weighted subgraph $G_{i,j}$ (recall Definition 4), then, we have $IN^t_{i,j} = IN_{i,j}$, where $1 \leq t \leq TF$.



(a) PRCA (b) PCSKA (c) PCLA (d) Shift Registers

Fig. 7: Verification time per circuit type. In Fig. 7(a), Fig. 7(b), and Fig. 7(c), the x-axis represents the input bit-width, while in Fig. 7(d), it corresponds to the number of time frames. The y-axis shows the verification time, sorted in ascending order. The solid and dotted lines indicate the verification time obtained from the Cutwidth and SAT approaches, respectively.

Let U be the number of unrolled subgraphs $G^t_{i,j}$ w.r.t the weight-based subgraph $G_{i,j}$. The overall time complexity for verifying the WAIG G can be calculated as follows:

$$\mathit{Complexity}(G) := \sum_{i=0}^{I}\sum_{j=0}^{J}\sum_{t=1}^{U}\mathcal{O}(2^{|IN^t_{i,j}|}) \tag{5}$$

Moreover, G is said to be a K-bounded graph, iff for every unrolled subgraphs $G^t_{i,j}$ has at most K inputs, where $0 \leq i \leq I$, $1 \leq j \leq J$, and $1 \leq t \leq U$ (recall Section II-B).

As the graph unrolling does not change the number of inputs of the weight-based subgraph $G_{i,j}$ (recall Definition 4), then, it always holds that $IN^t_{i,j} = IN_{i,j}$, where $1 \leq t \leq TF$.

Let K be the maximum number of inputs over all weight-based subgraphs $G_{i,j}$, where $0 \leq i \leq I$ and $1 \leq j \leq J$. Also, let N be the overall number of all unrolled subgraphs. Hence, the WAIG G can be verified in time $\mathcal{O}(N \cdot 2^K)$. If the constant K remains unchanged regardless of the circuit size n, it scales linearly with time. This is due to the fact that increasing the circuit size n will only increase the number of subgraphs N. ■

B. Empirical Evaluation

To show the feasibility of our approach, we have implemented the ASP framework and the W-AIG data structure in Python, where Clingo is used as the ASP solver. The framework takes input circuits in the standard AIGER format [37].

1) Experimental Setup: We mainly compare our approach (labeled as Cutwidth), and *Yosys* SAT-based BMC approach [38] (labeled as SAT) in terms of the overall time (referred to as verification time). In the SAT approach, a miter circuit is constructed and unrolled up to the vector width VW, and the resulting circuit is then verified. Notably, we are comparing the Cutwidth approach with the SAT approach, since the approach [14] is limited to counters, and the approach [15] is restricted to a single clock cycle. We evaluate *pipelined RCA* (PRCA), *pipelined CSKA* (PCSKA), and *pipelined CLA* (PCLA) of different sizes n up to 4096 bitwidth, and under different vector widths VW in terms of verification time, cutwidth (CW), and upper bound K. These types of pipelined adders have been selected, as these adders have a constant cutwidth as shown in [21]. The RCA, CSKA, and CLA adders are constructed using the ArithsGen tool [39] and synthesized using *Yosys*. Then, the pipelined adders are constructed by introducing flip flops between two adder blocks, each of size $n/2$, as shown in Fig. 1(a). To show that our approach is not limited to pipelined adders, we evaluate *Serial RCA* (SRCA), *Serial CSKA* (SCSKA), *Serial CLA* (SCLA), *Serial-In-Parallel-Out* (SIPO), and *Serial-In-Serial-Out* (SISO) of different sizes in terms of upper bound K. Consequently, they can also be verified in linear time.

TABLE I: Calculated cutwidth (CW), upper bound (K) for different circuits.

	Circuit Type							
	SIPO	**SISO**	**SRCA**	**SCSKA**	**SCLA**	**PRCA**	**PCSKA**	**PCLA**
CW	1	1	1	3	7	1	3	7
K	1	1	3	8	11	3	8	11

2) Experimental Results: Table I shows the upper bound K of inputs of different sequential circuits of different sizes up to 4096 (i.e., n =4096). We can see that these circuits have a constant width, and consequently, they can be verified in linear time using our approach. Fig. 7 shows the verification time of Cutwidth and SAT approaches for PRCA (Fig. 7(a)), PCSKA (Fig. 7(b)), and PCLA (Fig. 7(c)) w.r.t. the input bitwidth n and the vector width VW. The results demonstrate that the Cutwidth approach exhibits better scalability in terms of the verification time for adder circuits with a constant cutwidth compared to the SAT approach. Specifically, the curve is linear for the Cutwidth approach but non-linear for the SAT approach. Since these pipelined adders have a constant cutwidth, the verification time scales linearly w.r.t. n and VW. Fig. 7(d) shows the verification time of Cutwidth and SAT approaches for SIPO and SISO w.r.t. the number of time frames TF, where $VW = 1$. The results again demonstrate that the Cutwidth approach provides better scalability in verification time for both SIPO and SISO circuits compared to the SAT approach, with a linear curve for Cutwidth and a non-linear curve for SAT. This confirms the theoretical findings that we have proven in Section V-A.

VI. Conclusion

In this paper, we have introduced the W-AIG as a novel data structure that extends AIG representation, enabling efficient tracking of gate values across different time frames in sequential AIGs. Additionally, we have introduced a novel PFV approach that combines the cutwidth of W-AIG to partition the circuit into subcircuits, where the ASP solver was used to verify each subcircuit independently, and reason about the outgoing nodes of the subcircuit. Moreover, we have demonstrated that sequential circuits with a constant cutwidth K can be verified in a linear time w.r.t. the circuit size n and the time frames TF. Furthermore, we have conducted experiments to show that several sequential circuits exploit a constant cutwidth. Finally, our experiments have shown that several types of pipelined adders can be verified in linear time.

REFERENCES

[1] D. Lewin and D. Protheroe, *Sequential circuits*, pp. 200–251. 1992.

[2] F. S. Stanciulescu, "Sequential logic and its application to the synthesis of finite automata," *IEEE Trans. Electron. Comput.*, 1965.

[3] M. K. Stojcev, "Joseph cavanagh, sequential logic: Analysis and synthesis," *Microelectron. Reliab.*, pp. 1108–1109, 2008.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.*, p. 244–263, 1986.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[6] A. Ghosh, S. Devadas, and A. R. Newton, *Verification of Sequential Circuits*, pp. 123–151. 1992.

[7] R. Drechsler, ed., *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[8] R. Drechsler, *Formal System Verification: State-of the-Art and Future Trends*. Springer Publishing Company, Incorporated, 1st ed., 2017.

[9] K. L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, 1992.

[10] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using sat procedures instead of bdds," in *Proceedings 1999 Design Automation Conference*, pp. 317–320, 1999.

[11] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008.

[12] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *DAC*, pp. 532–535, 2006.

[13] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *DDECS*, pp. 99–104, 2021.

[14] C. Dominik and R. Drechsler, "Polynomial formal verification of sequential circuits," in *Proceedings of 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[15] L. Weingarten, K. Datta, A. Kole, and R. Drechsler, "Complete and efficient verification for a RISC-V processor using formal verification," in *Proceedings of 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[16] G. Brewka, T. Eiter, and M. Truszczyński, "Answer set programming at a glance," *ACM*, vol. 54, no. 12, p. 92–103, 2011.

[17] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2012.

[18] I. Niemelä, "Logic programs with stable model semantics as a constraint programming paradigm," *Annals of Mathematics and Artificial Intelligence*, vol. 25, pp. 241–273, 1999.

[19] F. Harmelen, V. Lifschitz, and B. Porter, "The handbook of knowledge representation," p. 1034, 2007.

[20] F. R. K. Chung, "On the cutwidth and the topological bandwidth of a tree," *SIDMA*, vol. 6, no. 2, pp. 268–277, 1985.

[21] M. Nadeem, J. Kleinekathofer, and R. Drechsler, "Polynomial formal verification exploiting constant cutwidth," in *Proceedings of the 34th International Workshop on Rapid System Prototyping*, RSP '23, Association for Computing Machinery, 2024.

[22] R. Diestel, *Graph Theory*. Springer Publishing Company, Incorporated, 5th ed., 2017.

[23] M. Gebser, R. Kaminski, A. König, and T. Schaub, "Advances in gringo series 3," in *LPNMR*, pp. 345–351, 2011.

[24] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[25] H. Peng, S. Huang, S. Chen, B. Li, T. Geng, A. Li, W. Jiang, W. Wen, J. Bi, H. Liu, *et al.*, "A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1135–1140, 2022.

[26] C. V. Ramamoorthy and H. F. Li, "Pipeline architecture," *ACM Computing Surveys (CSUR)*, vol. 9, no. 1, pp. 61–102, 1977.

[27] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[28] M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-driven answer set solving: From theory to practice," *Artificial Intelligence*, 2012.

[29] W. Marek and M. Truszczyński, "Autoepistemic logic," *Journal of ACM*, vol. 38, p. 587–618, 1991.

[30] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, pp. 365–385, 1991.

[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] V. W. Marek and M. Truszczynski, "Stable models and an alternative logic programming paradigm," *A Computing Research Repository*, 1998.

[33] T. Sato, "Completed logic programs and their consistency," *The Journal of Logic Programming*, vol. 9, no. 1, pp. 33–44, 1990.

[34] F. Wotawa and D. Kaufmann, "Model-based reasoning using answer set programming," *Applied Intelligence*, vol. 52, pp. 1–19, 2022.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[36] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition*. Addison-Wesley, 2011.

[37] A. Biere, "The AIGER And-Inverter Graph (AIG) format version 20071012," Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2007.

[38] C. Wolf, "Yosys open synthesis suite." https://yosyshq.net/yosys/.

[39] J. Klhufek and V. Mrazek, "Arithsgen: Arithmetic circuit generator for hardware accelerators," in *DDECS*, pp. 44–47, 2022.