PolyEMAC: Polynomial Error Metrics Analysis in Approximate Computing

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

***Abstract*—Approximate Computing (AC) is a design paradigm widely used in error-resilient applications. The goal of AC is to gain benefits over exact designs by trading some accuracy for a reduction in power, delay, and area. Varying error metrics are used to assess the degree of approximation. Several formal methods exist for computing these error metrics. However, these methods often fail to provide theoretical bounds on computational resources and are mostly limited to single error metrics. In this work, we address the limitations of the prior methods by introducing *PolyEMAC*, a novel approach that combines *Cutwidth Partitioning* and *Answer Set Programming (ASP)* to compute several error metrics for approximate circuits with constant cutwidth in polynomial time w.r.t. circuit size and the number of approximations. We evaluate several approximate adders in terms of cutwidth, memory consumption, and computation time to highlight the efficacy of the proposed methodology.**

***Index Terms*—Approximate Computing, Polynomial Error Analysis, Logic Synthesis, Answer Set Programming, Cutwidth.**

I. Introduction

In the context of digital system design, *Approximate Computing (AC)* is an emerging design paradigm, which is widely used in error-resilient applications such as image processing and machine learning [1]. By replacing exact Boolean functions with approximate ones of acceptable quality, AC circuits offer advantages over exact circuits in terms of power, performance, and area [2]. The tolerance to approximation in the error-resilient applications can vary widely, hence one of the major tasks in AC is evaluating the error metrics during the AC circuit design process.

Several error metrics such as *Worst Case Error (WCE)*, *Average Case Error (ACE)*, *Error Rate (ER)*, and *Mean Squared Error (MSE)* have been introduced [3] for the approximate designs. These are obtained by evaluating the difference between the exact outputs and the approximate outputs to check whether the error is below the desired limit. Naive simulation-based methods have exponential complexity and are impractical for computing errors over all input combinations. Hence, in recent years, formal methods have been studied to provide formal error metric evaluation for arithmetic circuits, including *Binary Decision Diagram (BDD)* [4], *Symbolic Compute Algebra (SCA)* [5], and *Boolean Satisfiability (SAT)* [6]. However, these methods face two main challenges. First, as design complexity continuously increases, these approaches fail to establish theoretical upper bounds on the overall error computation time (e.g., the NP-hard SAT problem [7]). Second, they are limited to specific error metrics (e.g., [4] is limited to WCE, ER, and ACE). The approach proposed in [8] introduces the use of SCA to evaluate various error metrics. However, this method also suffers from exponential complexity.

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project *PolyVer* (DR 287/36-1), and by the Data Science Center of the University of Bremen, funded by the State of Bremen.

In prior works, several formal methods have been employed in the area of *Polynomial Formal Verification (PFV)* [9]–[11] to show that approximate circuits can be verified in polynomial time using BDD [12]. Recently, it has been shown that approximate circuits can be verified more efficiently using ASP [13]. More precisely, *Cutwidth Partitioning* has been employed with ASP to show that approximate circuits with constant cutwidth can be verified in linear time [14], [15], where cutwidth is defined as a structural property of *And-Inverter Graph (AIG)* [16], representing to the minimum number of edge-cuts required to partition the circuit into subcircuits. However, there is no prior work that addresses the issue of obtaining the error metrics within a polynomial resource and time bound.

In this work, we address these limitations, and the following are the contributions: *A) We introduce PolyEMAC, a novel approach for computing error metrics using cutwidth partitioning and ASP. B) We prove that error metrics of approximate circuits with constant cutwidth can be computed in polynomial time. C) We conduct experimental evaluations on various approximate adder architectures of different sizes to show the efficacy of our proposed methodology.*

The approach relies on cutwidth to partition the circuit into subcircuits. Each subcircuit is then solved using ASP, and the probability of errors is computed w.r.t. the exact specification functions within each subcircuit. The intermediate nodes between the subcircuits are stored along with their error probabilities and passed to other subcircuits. This ensures that the number of input patterns in each subcircuit is bounded by the cutwidth of the circuit. Consequently, the overall complexity of computing the error metrics is also bounded by the cutwidth of the circuit.

II. Preliminaries

A. Error Metrics

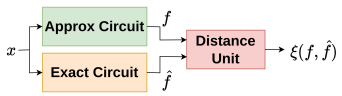


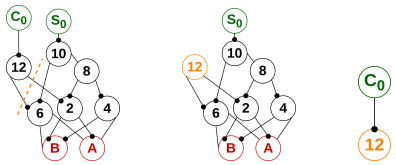
Fig. 1: Error Miter

To compute the error metrics, an *Error Miter* of Fig. 1 is introduced [17] that takes as input an exact circuit with inputs PI and outputs PO and an approximate circuit with inputs PI' and outputs PO', where $PI = PI'$ and $PO = PO'$. Let $f : \mathbb{B}^{PI} \mapsto \mathbb{B}^{PO}$ and $\hat{f} : \mathbb{B}^{PI} \mapsto \mathbb{B}^{PO}$ represent the output functions of the exact and approximate circuits, respectively,

mapping binary primary inputs PI to binary primary outputs PO. The outputs f and $\hat{f}$ are compared, and the distance unit is used to calculate the corresponding error metric $\xi(f, \hat{f})$ (e.g., ER). Let $int(f(x))$ be the integer induced by the binary output function $f(x)$ w.r.t. the input pattern x. We provide a brief overview of the error metrics considered in this work:

1) WCE represents the maximum error difference the approximate design may produce:
$$wce(f, \hat{f}) = \max_{x \in \mathbb{B}^{PI}} \{|int(f(x)) - int(\hat{f}(x))|\}. \quad (1)$$
2) ACE represents the average error in the approximate design:
$$ace(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^{PI}} |int(f(x)) - int(\hat{f}(x))|}{2^{PI}} \quad (2)$$
3) ER represents the probability that the approximate output differs from the exact output:
$$er(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^{PI}} int(f(x)) \neq int(\hat{f}(x))}{2^{PI}} \quad (3)$$
4) MSE represents the average squared error in the approximate design:
$$mse(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^{PI}} (int(f(x)) - int(\hat{f}(x)))^2}{2^{PI}} \quad (4)$$

B. Cutwidth Partitioning



(a) G (b) G_0 (c) G_1

Fig. 2: Nodes highlighted in red and green correspond to inputs and outputs, respectively. Solid circles represent inverter gates.

For a circuit design A, an AIG is a gate-level representation consisting of two-input AND gates AND (encoded as even literals $l \geq 2$), inverters INV, primary inputs PI, and primary outputs PO. A gate (also referred to as a node) can have at most two incoming edges. The primary inputs PI have no incoming edges, while the primary outputs PO have no outgoing edges. The AIG graph G can be defined as follows:

Definition 1 (AIG): Given a circuit A, the AIG $G = (V, E)$ is constructed such that:

- $V = AND \cup INV \cup PI \cup PO$.
- $E = \{(v, v') \mid v, v' \in V, v \text{ is an input of } v'\}$.

The cutwidth partitioning of G with outputs $O_0, \ldots, O_n \in PO$ corresponds to the minimum number of edges required to be removed to partition G into disjoint subgraphs $G_0 = (V_0, E_0), \ldots, G_n = (V_n, E_n)$ such that each subgraph G_i contains the output O_i and its dependency. It starts from an output O_i and recursively traverses all nodes that do not appear in other previous subgraphs $O_0, \ldots, O_{i-1}$ to inputs, where $0 \leq i \leq n$. To ensure the number of removed edges is minimum, the subgraph G_i goes from the inputs to the output O_i to add any node v such that its inputs appear in G_i. Given the AIG of Fig. 2(a) with the outputs $S_0, C_0 \in PO$, the subgraphs G_0 and G_1 are constructed as shown in Fig. 2(b) and Fig. 2(c), respectively.

We refer by CO_i and CI_i to the set of out-going (also called non-primary outputs) and in-going nodes (also called non-primary Inputs) such that CO_i contains nodes in G_i that are passed to other subgraphs $G_{i+1}, \ldots G_n$, and CI_i contains nodes that are evaluated in one of the previous subgraphs $G_0, \ldots, G_{i-1}$. We refer by $IN_i = PI_i \cup CI_i$ to the input nodes appearing in G_i, consisting of the primary inputs PI_i and the in-going nodes CI_i. For the subgraph G_0 of Fig. 2(b), we have $CO_0 = \{12\}$, $CI_0 = \emptyset$, and $PI_0 = A, B$, while for the subgraph G_1 of Fig. 2(c), we have $CO_1 = \emptyset$, $CI_1 = \{12\}$, and $PI_1 = \emptyset$ (for further details, see [14], [18]).

Algorithm 1: PolyEMAC Approach

Input : Approx AIG G^A and exact AIG G^E of the same size.
Output: Error Metrics ER, WCE, ACE, and MSE.

Miter Graph $G = (V, E) \leftarrow$ constructMiter(G^A, G^E)
Partition G into $G_0, \ldots, G_n$ w.r.t. cutwidth partitioning
for $i \leftarrow 0$ **to** n **do**
  if $CI_i \neq \emptyset$ **then**
    $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1}) \leftarrow (\mathcal{X}_{i-1}, P_{i-1}, D_{i-1}) \bowtie \ldots \bowtie (\mathcal{X}_0, P_0, D_0)$
    $Q(IN_i) \leftarrow Q(PI_i) \bowtie \mathcal{X}(CI_i)$
  else
    $Q(IN_i) \leftarrow Q(PI_i)$
  end if
  $\Pi_i \leftarrow$ ASP encoding w.r.t. $G_i = (V_i, E_i)$ and $Q(IN_i)$
  $\mathcal{X}_i \leftarrow$ Validate Π_i using *Clingo* solver
  if $i = 0$ **then**
    $P_i \leftarrow \frac{|\mathcal{X}_i|}{2^{|IN_i|}}$; // Unconditional Probability
  else
    $P_i \leftarrow \frac{|\mathcal{X}_i \cap |\mathcal{X}(CI_i)|}{|\mathcal{X}(CI_i)|}$; // Conditional Probability
  end if
end for
ER, WCE, ACE, MSE $\leftarrow$ ComputeMetrics($(\mathcal{X}_n, P_n, D_n)$)
return *ER, WCE, ACE, MSE*

C. Answer Set Programming

ASP is a widely used declarative programming framework well-suited for solving combinatorial problems. It is primarily used for tackling NP-hard search problems by reducing them to the task of computing *answer sets* [19] (for more details, see [20]).

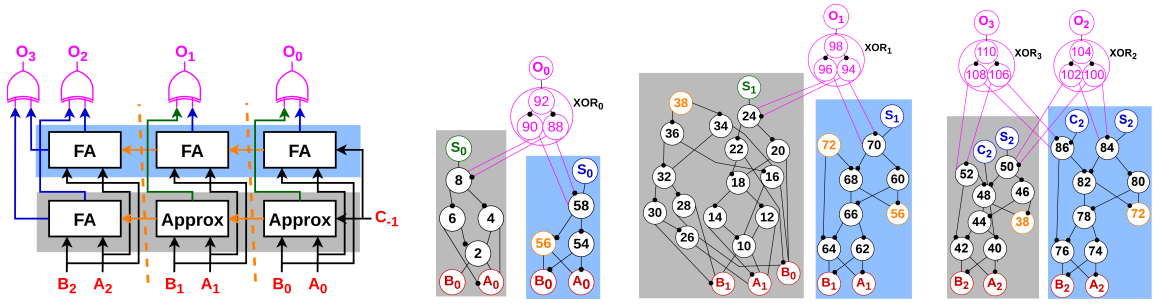
In the context of *Electronic Design Automation (EDA)*, the basic idea of ASP is to encode the graph G as a logic program Π, together with its specification functions. Then, a query x (represented by a set of facts) that defines an input pattern is added to the program Π (resulting in Π^x). The *Clingo* [21] solver is used to check whether an answer set of Π^x exists. If an answer set exists, then the graph G matches the specification functions under the input pattern x (i.e., x is a *valid input*). Otherwise, the input x is invalid, and consequently, the distance $\xi(f, \hat{f})$ is computed to derive the corresponding error metric.

Example 1: Given the subgraph G_0 of Fig. 2(b), the program Π is constructed such that $\Pi = \{and2 \leftarrow (A\hat{}1) \wedge (B\hat{}1); and6 \leftarrow A \wedge B; \ldots\}$. It is important to highlight that $A\hat{}1$ represents the complement of A. Considering the query $x = \{A \mapsto 0; B \mapsto 1\}$, we have that the exact output $f(x) = 1$ represents the exact sum function and the approximate output $\hat{f}(x) = 0$ (i.e., S_0). For the error metric WCE, we have $\xi(f, \hat{f}) = |int(f(x)) - int(\hat{f}(x))| = |0 - 1| = 1$.

III. PolyEMAC Approach

A. Algorithm

The PolyEMAC approach is summarized in Algorithm 1, which proposes a polynomial approach for calculating error metrics of circuits with constant cutwidth. The approach takes an approximate circuit A and an exact circuit E of the same size (i.e., $|PO^A| = |PO^E|$). Their corresponding AIGs G^A and G^E are constructed using *Yosys* tool [22].



(a) Miter Circuit (b) G_0 (c) G_1 (d) G_2

Fig. 3: The miter circuit (Fig. 3(a)) includes a 3-bit approximate circuit (gray, with the two least significant bits approximated), an exact 3-bit *Ripple Carry Adder (RCA)* (light blue), and subgraphs G_0, G_1, and G_2. Nodes in orange, green, blue, and magenta correspond to outgoing nodes, approximate outputs, exact outputs, and XOR gates, respectively.

The miter Graph G is constructed by XOR gates such that each XOR gate XOR_i connects an approximate output O_i^A and its corresponding exact output O_i^E (line 1). Given the approximate adder of size 3 with the two least significant bits approximated (labeled as "Approx"), and the exact RCA adder (i.e., consisting of three *Full Adder (FA)* blocks), the miter circuit is constructed as shown in Fig. 3(a). The cutwidth partitioning (recall Section II-B) is then used to partition the miter graph G into three subgraphs G_0, G_1, and G_2 as shown in Fig. 3(b), Fig. 3(c), and Fig. 3(d), respectively (line 2), such that each subgraph G_i contains one sum output S_i and its dependencies.

In each subgraph G_i, the input patterns $Q(IN_i)$ are constructed such that the in-going table $\mathcal{X}(CI_i)$ is populated with the primary inputs PI_i (lines 4-8). For the first subgraph G_0, we have $CI_0 = \emptyset$ and consequently, the input patterns $Q(IN_0)$ consist only of the primary inputs PI_0 (line 8). The subgraph G_i and the input patterns are then encoded as an ASP program Π_i (line 10), and the out-going table $\mathcal{X}_i$ is computed (line 11). To enable the computation of error metrics, the probability P_i and the distance unit D_i are computed w.r.t. the table $\mathcal{X}_i$, and the results are stored and passed to the next subgraph G_{i+1} (lines 12-16). For the first subgraph G_0, we have that $\mathcal{X}(CI_0) = \emptyset$. Therefore, an unconditional probability is computed for the out-going table $\mathcal{X}_0$ (line 13). For subgraphs G_i, where $n \geq i > 0$, a conditional probability is computed for $\mathcal{X}_i$ w.r.t. the in-going probability table $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$ (line 15). After evaluating all subgraphs, the error metrics are computed w.r.t. the probability of the last out-going probability table $(\mathcal{X}_n, P_n, D_n)$ (line 18).

B. Information Passing

TABLE I: $\mathcal{X}_0$

x		$g(x)$		
A_0	B_0	56	S_0^A	S_0^E
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	0	0

TABLE II: $(\mathcal{X}_0, P_0, D_0)$

B_0	56	S_0^A	S_0^E	D_0	P_0
0	0	0	0	0	1/4
1	0	0	1	1	1/4
0	0	1	1	0	1/4
1	1	0	0	0	1/4

To enable independent evaluation of each subgraph G_i, a mapping function $g : Q(IN_i) \mapsto COUT_i$ is introduced, which maps every input pattern $x \in Q(IN_i)$ to the values $COUT_i$ of the outgoing nodes CO_i. Since error metrics are computed w.r.t. the values of the approximate and exact outputs, the outputs of both the approximate and exact circuits within the subgraph G_i are included in CO_i.

For the subgraph G_0, we have that $CO_0 = \{12, B_0, S_0^A, S_0^E\}$ and $CI_0 = \emptyset$. Notably, B_0 is also considered as out-going node, since it appears as input for a gate that does not appear in G_0 (recall Section II-B). The ASP program Π_0 is constructed w.r.t. G_0 and $Q(IN_0)$ (i.e., $IN_0 = PI_0 \cup CI_0 = \{A_0, B_0\}$). The out-going table $\mathcal{X}_i = \{g(x) \mid x \in Q(IN_i)\}$ is constructed such that it contains the resulting values under all possible input patterns $Q(IN_0)$ as shown in Table I. To enable computing error metrics within the subgraph G_i, it is essential to compute the probability of the error P_i, and the distance unit D_i is computed such that it contains the integer differences between the exact and approximate outputs appearing in subgraph G_i. The distance unit D_i is computed as follows:

$$D_i(\mathcal{X}_i = g(x)) = D_{i-1}(\mathcal{X}(CI_i) = x) + \left((S_i^E - S_i^A) \cdot 2^i\right) + \left(1 \text{ if } O_{n+1}^E \neq O_{n+1}^A, \text{ else } 0, O_{n+1}^E, O_{n+1}^A \in G_i\right), \text{with } x \in Q(IN_i) \quad (5)$$

To illustrate this, the distance unit D_i adds the current integer difference of the outputs in G_i to the accumulated difference from G_{i-1}. Since the last subgraph G_n includes a carry C_n (i.e., represented by O_{n+1}), we add 1 if $O_{n+1}^A \neq O_{n+1}^E$ and 0 otherwise. For the first subgraph G_0, we have that $D_{-1}(\mathcal{X}(CI_0) = x) = 0$. This is because $Q(IN_0)$ contains only input patterns from the primary inputs PI_0. Also, as $CI_0 = \emptyset$, the unconditional probability is computed as follows:

$$P_i(\mathcal{X}_i = g(x)) = \frac{\#\{\text{rows with } \mathcal{X}_i = g(x)\}}{|Q(IN_i)|}, \text{where } x \in Q(IN_i) \quad (6)$$

Let $(\mathcal{X}_i, P_i, D_i)$ be the resulting table (also called the out-going probability table). Given the out-going table $\mathcal{X}_0$ of Table I, the probability table $(\mathcal{X}_0, P_0, D_0)$ is constructed as shown in Table II. Since every row occurred only once, we have that the probability per row is $P_0(\mathcal{X}_0 = x) = 1/4 = 0.25$, for every row $x \in \mathcal{X}_0$. The probability table $(\mathcal{X}_0, P_0, D_0)$ is passed to the next subgraph G_1.

In subgraph G_1 of Fig. 3(c), we have that $CI_1 = \{56\}$. To ensure the probabilities are computed correctly, we add the previous outputs S_{i-1}^A and S_{i-1}^E to the in-going nodes CI_i. Hence, $CI_1 = \{56, S_0^A, S_0^E\}$. To compute the values CI_i, the out-going probability tables are joined such that $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1}) = (\mathcal{X}_{i-1}, P_{i-1}, D_{i-1}) \bowtie \ldots \bowtie (\mathcal{X}_0, P_0, D_0)$, where $\bowtie$ denotes a relational join between the tables. We refer by $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$ to the in-going probability table. Given the subgraph G_1 with $CI_1 = \{56, S_0^A, S_0^E\}$, the in-going probability table $(\mathcal{X}(CI_1), P_0, D_0) = (\mathcal{X}_0, P_0, D_0)$ is constructed as shown

in Table II. The input patterns $Q(IN_1)$ is constructed by populating the primary inputs $PI_1 = \{A_1, B_1\}$ with the in-going table $(\mathcal{X}(CI_1), P_0, D_0)$. The subgraph G_1 and the input patterns $Q(IN_1)$ are encoded into an ASP program Π_1. The out-going table $\mathcal{X}_1$ is constructed as shown in Table III. The rows highlighted in blue represent the errors induced by the first output bit S_0^A and S_0^E, while the rows highlighted in red denote the errors induced by the current output bit S_1^A and S_1^E.

TABLE III: $\mathcal{X}_1$ and $(\mathcal{X}_1, P_1, D_1)$

x				$g(x)$				Distance Unit		
B_0	56	S_0^A	S_0^E	38	72	S_1^A	S_1^E	D_1	P_0	P_1
1	0	0	1	0	1	1	0	-1	1/4	1/16
1	0	0	1	1	1	1	1	1	1/4	1/16
1	0	0	1	1	1	0	1	3	1/4	1/16
1	0	0	1	1	0	1	0	-1	1/4	1/16
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The next step is to compute the probability P_1 and the distance unit D_1 w.r.t. $\mathcal{X}_1$. Since $CI_1 \neq \emptyset$, the conditional probability is computed for $\mathcal{X}_1$ w.r.t. the in-going probabiltiy table $(\mathcal{X}(CI_1), P_0, D_0)$. Given a out-going table $\mathcal{X}_i$ and an in-going probability table $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$, the unconditional probability is computed as follows:

$$P_i(\mathcal{X}_i = g(x) \mid \mathcal{X}(CI_i) = x) = \left(\frac{\#\{\text{rows with } \mathcal{X}_i = g(x) \cap \mathcal{X}(CI_i) = x\}}{\#\{\text{rows with } \mathcal{X}(CI_i) = x\}}\right) \cdot P_{i-1}(\mathcal{X}(CI_i) = x), \text{where } x \in Q(IN_i). \quad (7)$$

To illustrate this, the incorrect input pattern $A_0 \mapsto 0$ and $B_0 \mapsto 1$ is the only incorrect pattern w.r.t. the subgraph G_0. Therefore, its probability is $P_0(B_0 \mapsto 1, 56 \mapsto 0, S_0^A \mapsto 0, S_0^E \mapsto 1) = 1/4$. However, as it is populated with new inputs A_1 and B_1 of subgraph G_1, the probability becomes $4/16$ (i.e., the row $B_0 \mapsto 1, 56 \mapsto 0, S_0^A \mapsto 0, S_0^E \mapsto 1$ is populated with 2^{PI_1} in the subgraph G_1, where $PI_1 = \{A_1, B_1\}$). By doing so, it is ensured that the probability is computed correctly over the subgraph $G_0, \ldots, G_n$. To illustrate the distance unit computation, we have that $D_0(B_0 \mapsto 1, 56 \mapsto 0, S_0^A \mapsto 0, S_0^E \mapsto 1) = 1$, and $D_1(38 \mapsto 1, 72 \mapsto 1, S_1^A \mapsto 0, S_1^E \mapsto 1) = D_0 + (1-0) \cdot 2^1 + 0 = 1 + 2 + 0 = 3$.

The approach continues to compute the distance unit D_i and the probability G_i until it reaches the last subgraph G_n.

C. Error Metrics Computation

To compute the error metrics WCE, ACE, ER, and MSE, we rely on the distance unit D_n and P_n. It is important to highlight that every out-going probability table $(\mathcal{X}_i, P_i, D_i)$ contains the distance unit D_i and the error probability P_i from subgraph G_0 to G_i, where $0 \leq i \leq n$. The error metrics can be computed w.r.t. the last out-going probability table $(\mathcal{X}_n, P_n, D_n)$ as follows:

$$wce(\mathcal{X}_n, P_n, D_n) = \max_{x \in \mathcal{X}_n}(|D_n(x)|) \quad (8)$$

$$ace(\mathcal{X}_n, P_n, D_n) = \sum_{x \in \mathcal{X}_n}(D_n(x) \cdot P_n(x)) \quad (9)$$

$$mse(\mathcal{X}_n, P_n, D_n) = \sum_{x \in \mathcal{X}_n}\left((D_n(x))^2 \cdot P_n(x)\right) \quad (10)$$

$$er(\mathcal{X}_n, P_n, D_n) = \sum_{x \in \mathcal{X}_n}(e(x) \cdot P_n(x)),$$

$$\text{where } e(x) = \begin{cases} 1, & \text{if } D_n(x) > 0 \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

Given the miter graph G of Fig. 3, we have $wce = 7$, $ace = 1.84$, $mse = 7.25$, and $er = 0.6875$ w.r.t. the table $(\mathcal{X}_3, P_3, D_3)$. We refer by $cw_{in} = \max(IN_0, \ldots, IN_n)$ to the in-going cutwidth, indicating the maximum number of input patterns that need to be checked in any subgraph $G_0, \ldots, G_n$. Similarly, $cw_{out} = \max(CO_0, \ldots, CO_n)$ to the out-going cutwidth, indicating the maximum number of values that need to be stored and passed to other subgraphs $G_0, \ldots, G_n$. We can see that the overall complexity is reduced from 2^{PI} to be a function of the in-going cutwidth cw_{in} and the out-going cutwidth cw_{out}.

IV. Complexity Analysis

Let G be a miter graph constructed w.r.t. an approximate AIG $G^A = (V^A, E^A)$ and an exact AIG $G^E = (V^E, E^E)$. Let G_i be a subgraph of G constructed using cutwidth partitioning (recall Section II-B). The time complexity of validating G_i and computing its probability P_i and distance unit D_i can be characterized as follows:

Lemma 4.1: Let $G_i = (V_i, E_i)$ be a subgraph of the miter graph G. Then, the evaluation of the program Π_i w.r.t. G_i has a computational complexity of $\mathcal{O}(2^{|IN_i|} \cdot (2^{|CO_i|} \cdot |(\mathcal{X}(CI)_i, P_{i-1}, D_{i-1})|)$.

Proof: Let $G_i = (V_i, E_i)$ be a subgraph of G with in-going nodes CI_i and out-going nodes CO_i (recall Section II-B). Then, the *Horn* ASP program Π_i is constructed w.r.t. G_i [14]. Let $Q(IN_i)$ be the input patterns constructed w.r.t. $\mathcal{X}(CI_i)$ and the primary inputs Π_i. For every input pattern $x \in Q(IN_i)$, the program Π_i is evaluated under x and the values $g(x)$ of the out-going nodes CO_i are obtained. Since Π_i is a horn program, then evaluating one input pattern $x \in Q(IN_i)$ can be achieved in linear time w.r.t. the program size [23] (recall Section II-C). Since the size of $Q(IN_i)$ is bounded by $2^{|IN_i|}$ (i.e., $|Q(IN_i)| \leq 2^{|IN_i|}$), evaluating Π_i under all possible input patterns is achieved in $\mathcal{O}(2^{|IN_i|})$. Consequently, $\mathcal{X}_i$ can be obtained w.r.t. all input patterns.

To enable the computation of error metrics, constructing the out-going probability table $(\mathcal{X}_i, P_i, D_i)$ is essential. Computing the probability P_i requires joining the in-going probability table $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$ with the out-going table $\mathcal{X}_i$ (recall Equation (III-B)). Since the in-going probability table is bounded by $|(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})|$ and the out-going table $\mathcal{X}_i$ is bounded by $2^{|CO_i|}$, the worst case requires performing a cross-join between the two tables (i.e., checking every row in one table with every row in the other table). Hence, computing the out-going probability table $(\mathcal{X}_i, P_i, D_i)$ can be achieved in $\mathcal{O}(2^{|CO_i|} \cdot |(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})|)$. For the subgraph G_0, we have that $CI_0 = \emptyset$. Therefore, unconditional probability is computed (recall Equation (6)). Therefore, evaluating the subgraph G_i has a computational complexity of $\mathcal{O}(2^{|CO_i|} \cdot |(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})|)$. ■

Since the out-going probability table $(\mathcal{X}_i, P_i, D_i)$ contains binary values representing the out-going nodes CO_i and integer values representing the integer distance D_i between the exact outputs and the approximate outputs appearing in G_i, the size of the table can be larger than $2^{|CO_i|}$. In fact, the size of the table is bounded by $2^{|CO_i|} \cdot S_i$, where S_i is the size of the unique values of D_i w.r.t. G_i (i.e., $S_i = |\{D_i\}|$). The size S_i is bounded by the index of the output bit (i.e., $-2^{i+1}+1 \leq D_i \leq 2^{i+1}-1$). This is because the distance D_i between the exact output and the approximate output can be positive or negative, appearing in G_i. For instance, the subgraph G_1 of Fig. 3(c), the distance D_1 can have any value between -3 and 3 (i.e.,

TABLE IV: Overall computation time for approximate circuits w.r.t. different approaches.

Type	Size	#Approx	#Gates	#Levels	cw_{in}	cw_{out}	ER	WCE [log]	ACE	MSE [log]	PolyEMAC Memory	PolyEMAC Wall Time	Simulation
AHA1	8	4	170	27	6	4	68.36	2.41	6.67	2.89	1.93	1.01	2.01
AHA2	8	4	158	23	6	4	100.0	1.18	5.38	1.70	2.48	0.97	2.01
AHA3	8	4	194	27	6	4	100.0	2.40	10.80	2.91	2.49	1.08	2.00
MA1	8	4	146	27	6	4	73.44	2.41	5.47	2.90	2.49	1.05	2.00
MA2	8	4	158	23	6	4	68.36	1.18	3.62	1.45	1.94	1.00	2.00
MA3	8	4	158	27	6	4	86.72	2.41	7.44	2.90	2.49	1.08	2.02
MA4	8	4	146	24	6	4	86.72	2.41	7.89	3.00	2.49	1.13	2.01
SESA2	8	4	107	23	6	4	93.75	1.18	7.50	1.89	2.49	0.98	2.00
SESA3	8	4	206	23	6	4	93.75	1.18	7.50	1.89	2.49	0.99	2.02
AHA1	16	8	338	55	6	4	89.99	4.82	106.73	6.45	2.51	1.85	T.O.
AHA2	16	8	314	47	6	4	100.0	2.41	75.41	4.04	2.50	1.94	T.O.
AHA3	16	8	386	55	6	4	100.0	4.82	167.27	6.45	2.53	2.27	T.O.
MA1	16	8	290	55	6	4	85.91	4.82	77.65	6.45	2.51	1.90	T.O.
MA2	16	8	314	47	6	4	89.99	2.41	59.69	3.86	2.51	1.79	T.O.
MA3	16	8	314	55	6	4	97.56	4.82	114.35	6.45	2.51	2.07	T.O.
MA4	16	8	290	48	6	4	97.56	4.82	130.06	6.62	2.52	2.13	T.O.
SESA2	16	8	211	47	6	4	99.61	2.41	127.50	4.34	2.50	1.48	T.O.
SESA3	16	8	410	47	6	4	99.61	2.41	127.50	4.34	2.50	1.54	T.O.
AHA1	32	16	674	111	6	4	99.00	9.63	27306.74	13.67	10.62	275.59	T.O.
AHA2	32	16	626	95	6	4	100.00	4.82	19115.42	8.85	6.94	156.42	T.O.
AHA3	32	16	770	111	6	4	100.00	9.63	42715.84	13.67	18.05	550.43	T.O.
MA1	32	16	578	111	6	4	96.03	9.63	19734.68	13.67	11.88	333.38	T.O.
MA2	32	16	626	95	6	4	99.00	4.82	15291.73	8.68	10.73	306.33	T.O.
MA3	32	16	626	111	6	4	99.92	9.63	29190.85	13.67	13.28	422.84	T.O.
MA4	32	16	578	96	6	4	99.92	9.63	33367.14	13.85	13.43	412.66	T.O.
SESA2	32	16	419	95	6	4	100.00	4.82	32767.50	9.16	7.00	155.66	T.O.
SESA3	32	16	818	95	6	4	100.00	4.82	32767.50	9.16	6.81	156.61	T.O.

$-2^2+1 \leq D_i \leq 2^2-1$). Hence, $S_i \leq 2 \cdot (2^{i+1}-1)+1$. The upper bound for S_i keeps increasing based on the number of approximate bits appearing in the approximate circuit C^A. We rely on the upper bound of S_i in computing the complexity of the PolyEMAC approach. The overall complexity can be characterized in the following theorem:

Theorem 4.2: Let $G=(V,E)$ be a miter graph constructed w.r.t. the approximate AIG $G^A=(V^A,E^A)$ with A approximate bits and the exact AIG $G^E=(V^E,E^E)$. Then, the PolyEMAC approach for computing error metrics has a time complexity of $\mathcal{O}(n \cdot 2^{|cw_{in}|} \cdot (2^{|cw_{out}|} \cdot S))$, where $cw_{in} = \max(|IN_0|,\ldots,|IN_n|)$, $cw_{out} = \max(|CO_0|,\ldots,|CO_n|)$, $S = 2 \cdot (2^{A+1}-1)+1$, and n is the number of subgraphs.

Proof: Let $G=(V,E)$ be a miter graph with $n+1$ outputs. The subgraphs $G_0,\ldots,G_n$ are constructed using the cutwidth partitioning w.r.t. G (recall Section II-B). For each subgraph G_i, two tasks are performed. First is to compute the in-going probability table $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$. We assume that $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$ can be computed in constant time. This is because the search operation of a table may take linear time in the worst case [24]. Second is to encode G_i together with the input patterns $Q(IN_i)$ as the horn ASP program Π_i and compute the out-going table $\mathcal{X}_i$. Since evaluating one input pattern $x \in Q(IN_i)$ can be achieved in linear time for a horn ASP program Π_i [23], the complexity for evaluating Π_i under all possible patterns $x \in Q(IN_i)$ is bounded by $\mathcal{O}(2^{|Q(IN_i)|})$. As the size of $Q(IN_i)$ is bounded by the in-going nodes $2^{|CI_i|}$, evaluating Π_i can be achieved in time $\mathcal{O}(2^{|CI_i|})$.

Finally, the out-going probability table $(\mathcal{X}_i, P_i, D_i)$ has to be constructed for $\mathcal{X}$ w.r.t. the in-going probability table $(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})$. It can be achieved in time $\mathcal{O}(2^{|CO_i|} \cdot |(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})|)$ (recall Lemma 4.1). As $|(\mathcal{X}(CI_i), P_{i-1}, D_{i-1})| = 2^{|CI_i|} \cdot S_i$, where $S_i = 2 \cdot (2^{i+1}-1)+1$ denotes the size of all possible values of the distance unit D_i, the worst case complexity for computing $(\mathcal{X}_i, P_i, D_i)$ is $\mathcal{O}(2^{|CO_i|} \cdot (2^{|CI_i|} \cdot S_i))$. The overall complexity for evaluating all subgraphs $G_0,\ldots,G_n$ can be characterized as follows:

$$\textit{Complexity}(G) = \mathcal{O}\left(\sum_{i=0}^{n} 2^{|IN_i|} + \left(2^{|CO_i|} \cdot (2^{|CI_i|} \cdot S_i)\right)\right) \quad (12)$$

Let $cw_{in} = \max(|IN_0|,\ldots,|IN_n|)$, $cw_{out} = \max(|CO_0|,\ldots,|CO_n|)$, and $S = \max(S_0,\ldots,S_n)$ be the in-going cutwidth, the out-going cutwidth, and the maximum size of all possible values any distance unit $D_0,\ldots,D_n$ can have. Hence, the PolyEMAC approach can run in time $\mathcal{O}(2^{|cw_{in}|} \cdot (2^{|cw_{out}|} \cdot S))$. It is important to highlight that the range of the values D_i can only increase up to the number of approximate bits A w.r.t. $G=(V^A,E^A)$. Hence, the value of S is bounded by $S = 2 \cdot (2^{A+1}-1)+1$ (i.e., $-2^A+1 \leq S \leq 2^A-1$). For the non-approximate subgraphs $G_{A+1},\ldots,G_n$, the distance unit D_i does not change. Hence, for a subgraph G_j, we have $S_j \leq S_i$, where $0 \leq i \leq A < j \leq n$. We conclude that PolyEMAC runs in time $\mathcal{O}(2^{|cw_{in}|} \cdot (2^{|cw_{out}|} \cdot (2 \cdot (2^{A+1}-1)+1)))$. ■

By the previous theorem, for approximate circuits with constant cutwidth (i.e., cw_{in} and cw_{out} do not change over different circuit size) with a fixed number of approximate bits A, it holds that PolyEMAC scales polynomially w.r.t. the circuit size.

V. Experimental Work

To check the feasibility of the PolyEMAC approach, we have implemented the ASP framework in Python. The framework takes an input approximate circuit and an exact circuit in the standard AIGER format [16]. We evaluate *Approximate Hybrid Adders* (i.e., *AHA1*, *AHA2*, and *AHA3*) [25], *Approximate Mirror Adders* (i.e., *MA1*, *MA2*, *MA3*, and *MA4*) [26], and *Single Exact Single Approximate Adders* (i.e., *SESA2*, and *SESA3*) [27] of different sizes n up to 64, and different numbers of approximate bits between 1 and $n-1$. It has been shown that these circuits exhibit constant cutwidth [14]. We compare the PolyEMAC approach with the simulation approach in terms of overall wall time (seconds). Due to the page limit, we only show the comparison for circuits with different sizes n up to 64 with $n/2$ approximate bits. All experiments were performed on an Intel(R) Core(TM) i7-11370 with 3.30 GHz, where the timeout and available RAM per instance are set to 1800 seconds and 16 GB, respectively.

A. Experimental Results

Table IV shows the results for approximate circuits with $n/2$ approximate bits, comparing the PolyEMAC and sim-

TABLE V: Overall computation time of PolyEMAC for 64-bit approximate circuits under different approximation bits.

Type	Size	#Approx	#Gates	#Levels	ER	WCE [log]	ACE	MSE [log]	Memory	Wall Time
AHA1	64	4	842	139	68.36	2.71	3.75	3.40	2.50	3.34
MA2	64	4	830	135	68.36	2.71	3.62	3.33	2.50	2.96
MA4	64	4	782	130	48.44	44.36	0.75	43.67	2.50	2.90
SESA3	64	4	878	135	93.75	2.71	7.50	4.35	2.50	2.88
AHA1	64	8	715	151	53.28	44.36	38.30	46.22	2.56	6.64
MA2	64	8	717	143	82.66	5.54	29.85	8.20	2.55	6.58
MA4	64	8	694	144	90.46	5.54	33.15	8.27	2.58	8.53
SESA3	64	8	746	143	99.61	5.54	127.5	9.99	2.53	4.83
AHA1	64	12	759	163	96.83	8.32	1023.75	14.56	3.48	56.92
MA2	64	12	761	151	96.83	8.32	955.72	14.44	3.53	63.82
MA4	64	12	726	152	96.10	8.32	530.71	13.82	3.77	85.52
SESA3	64	12	806	151	50.12	8.32	1023.75	14.84	3.03	33.64

ulation approaches. It shows the approximate circuit type, size, number of approximate bits (#Approx), number of gates (#Gates), circuit depth (#Levels), in-going cutwidth (cw_{in}), out-going cutwidth (cw_{out}), error rate (ER), worst-case error [log] (WCE), average-case error (ACE), mean squared error [log] (MSE), memory consumption (Memory) for PolyEMAC, computation time (Wall Time) for PolyEMAC, and computation time for the simulation approach (Simulation). T.O. denotes a timeout (i.e., the approach exceeded the time limit of 1800 seconds). We observe that the PolyEMAC approach outperforms the simulation approach across all circuit sizes and under different approximate bits. Also, the simulation approach exceeded the time limit starting from approximate circuits of sizes 16, while the PolyEMAC approach was able to solve all instances. Additionally, memory consumption is primarily influenced by the number of approximate bits. For instance, for *AHA1* with size 8 and 4 approximate bits, the memory consumption is 1.93, whereas for size 16 and 8 approximate bits, it increases to 2.51. This aligns with our theoretical findings that PolyEMAC runs in time $\mathcal{O}(n \cdot 2^{|cw_{in}|} \cdot (2^{|cw_{out}|} \cdot S))$, where S is bounded by the approximate bits $A = n/2$ (recall Theorem 4.2). As these circuits exhibit constant cutwidth, the PolyEMAC computes the error metrics in polynomial time.

To demonstrate that our approach scales to large circuit sizes, Table V presents the results for *AHA1*, *MA2*, *MA4*, and *SESA3* of size 64 under 4, 8, and 12 approximate bits. We observe that varying the number of approximate bits for the same circuit size has a significant impact on both wall time and memory consumption. For *MA2*, we observe that the memory consumption is 2.50 and the wall time is 2.96 with 4 approximate bits. In contrast, with 12 approximate bits, the memory consumption increases to 3.48 and the wall time to 63.82. This demonstrates that, for circuits with constant cutwidth, the number of approximate bits has a significant practical impact on both memory usage and overall computation time. This is because $S = 2 \cdot (2^{A+1} - 1) + 1$ (recall Theorem 4.2), meaning that increasing the number of approximate bits directly increases the value of S.

VI. Conclusion

In this paper, we introduced the PolyEMAC approach for computing error metrics in approximate circuits with constant cutwidth. Moreover, we proved that these metrics can be computed in polynomial time w.r.t. both the circuit size and the number of approximate bits. Finally, our experimental evaluation demonstrated that PolyEMAC outperforms the simulation-based approach across various circuit sizes and different approximate bits. In future work, we aim to extend our approach to compute additional error metrics in polynomial time. We also plan to investigate different approximate circuit architectures, especially those with non-constant cutwidth.

References

[1] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013.
[2] W. Liu *et al.*, "A retrospective and prospective view of approximate computing," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.
[3] V. Mrazek *et al.*, "Evoapproxlib: Extended library of approximate arithmetic circuits," in *Proc. Workshop Open-Source EDA Technol.(WOSET)*, 2019.
[4] M. Soeken *et al.*, "BDD minimization for approximate computing," in *ASP-DAC*, 2016, pp. 474–479.
[5] S. Froehlich *et al.*, "Approximate hardware generation using symbolic computer algebra employing Gröbner basis," in *DATE*, 2018, pp. 889–892.
[6] C. Meng *et al.*, "Mecals: A maximum error checking technique for approximate logic synthesis," in *2023 DATE*, 2023, pp. 1–6.
[7] S. A. Cook, "The complexity of theorem proving procedures," in *3. ACM Symposium on Theory of Computing*, 1971, pp. 151–158.
[8] S. Froehlich *et al.*, "One method - all error-metrics: A three-stage approach for error-metric evaluation in approximate computing," in *2019 DATE*, 2019, pp. 284–287.
[9] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *DDECS*, 2021, pp. 99–104.
[10] R. Drechsler *et al.*, "Polynomial formal verification: Ensuring correctness under resource constraints," in *ICCAD*, 2022, pp. 70:1–70:9.
[11] M. Schnieber *et al.*, "Polynomial formal verification of approximate functions," in *ISVLSI*, 2022.
[12] M. Schnieber *et al.*, "Polynomial formal verification of approximate adders," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022.
[13] A. Provetti *et al.*, "Answer set programming: Towards efficient and scalable knowledge representation and reasoning," in *Proceedings of the 1st Intl. ASP'01 Workshop*, 2001.
[14] M. Nadeem *et al.*, "Polynomial formal verification of approximate adders with constant cutwidth," in *Proceedings of 2024 IEEE European Test Symposium (ETS)*, 2024.
[15] M. Nadeem *et al.*, "Polynomial formal verification of multi-valued approximate circuits within constant cutwidth," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 3, pp. 1411–1424, 2025.
[16] A. Biere, "The AIGER And-Inverter Graph (AIG) format version 20071012," Institute for Formal Models and Verification, Johannes Kepler University, Tech. Rep., 2007.
[17] R. Venkatesan *et al.*, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM ICCAD*, 2011, pp. 667–673.
[18] M. Nadeem *et al.*, "Polynomial formal verification exploiting constant cutwidth," in *Proceedings of the 34th International Workshop on Rapid System Prototyping*. Association for Computing Machinery, 2024.
[19] T. Sato, "Completed logic programs and their consistency," *The Journal of Logic Programming*, vol. 9, no. 1, pp. 33–44, 1990.
[20] "Logic programs with stable model semantics as a constraint programming paradigm," *Annals of Mathematics and Artificial Intelligence*, 1999.
[21] M. Gebser *et al.*, "Advances in gringo series 3," in *LPNMR*, 2011.
[22] C. Wolf, "Yosys open synthesis suite," https://yosyshq.net/yosys/.
[23] J. K. Fichte *et al.*, "Backdoors to tractable answer set programming," *Artificial Intelligence*, vol. 220, pp. 64–103, 2015.
[24] R. Sedgewick *et al.*, *Algorithms, 4th Edition*. Addison-Wesley, 2011.
[25] C. K. Jha *et al.*, "Energy and error analysis framework for approximate computing in mobile applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 2, pp. 385–389, 2020.
[26] V. Gupta *et al.*, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 124–137, 2013.
[27] C. K. Jha *et al.*, "Single exact single approximate adders and single exact dual approximate adders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 7, pp. 907–916, 2023.