Polynomial Formal Verification of Approximate Adders

Martha Schnieber* Saman Froehlich* Rolf Drechsler*†
*Institute of Computer Science, University of Bremen, Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{schnieber, froehlich, drechsler}@uni-bremen.de

***Abstract*—To ensure the functional correctness of digital circuits, formal verification methods have been established, where the circuits are proven to implement the correct function. Several methods exist for the execution of the verification process. However, the verification process can have an exponential time or space complexity, causing the verification to fail. While exponential in general, recently it has been proven that the verification complexity of several circuits is polynomially bounded.**

In this paper, we prove the polynomial verifiability of several state-of-the-art approximate adders using BDDs. These approximate adders include handcrafted approximate adders, which consist of several subadders, as well as automatically generated approximate adders, where regular adders can be arbitrarily altered by removing gates and changing the type of gates. Thus, this paper provides insight into the possible methods for the design of approximate adders, such that the approximate adders remain polynomially verifiable. Here, we give upper bounds for the BDD sizes during the verification process, as well as for the time and space complexity. The upper bounds for the BDD sizes are then experimentally evaluated.

I. Introduction

As digital circuits are a prominent part of modern systems, including safety-critical applications such as airplanes, their functional correctness has to be ensured. Here, it is checked whether the circuit correctly implements the specification. To ensure the correctness, formal verification methods have been established, such as the verification using SAT solvers or decision diagrams like BDDs [1]. Using BDDs, the BDD of the specification and the BDD of the implemented circuit are constructed and compared during the verification process.

However, the construction of the formal representation of the circuit can have an exponential time and space complexity, as the formal representation constructed during the verification process can have an exponential size [2], causing the verification to fail due to time or space constraints. Therefore, recent research has focused on the complexity of several verification processes, more specifically the polynomial verification of circuits. Here, it has been discovered that the verification time and space of several specific circuits scale polynomially. Examples of polynomially verifiable circuits include several adders, such as the *Ripple Carry Adder* (RCA), the *Conditional Sum Adder* (CSA), the *Carry Look Ahead Adder* (CLA) and multiple Prefix Adders (see [3] [4] [5]).

The presented methods for polynomial verification focus on exact functions, however, for some error-tolerant applications, approximate circuits are sufficient, e.g. for media processing or data mining. Here, the circuit approximates the exact function in some cases, while having a lower delay or being more area-efficient [6] [7]. Many approximation techniques exist for several functions, including the adder function. To evaluate the error of an approximation, it can be verified using formal methods such as BDDs or SAT [8]. However, even if the exact circuit is polynomially verifiable, it is not clear whether the bounds still hold for the approximate circuit.

The polynomial verifiability of several adders has already been proven [3] [4] [5], but the polynomial verifiability of approximate adders has not been researched yet. Therefore, in this paper, we prove that the verification process using BDDs of several state-of-the-art approximate adders is guaranteed to have a polynomial time and space complexity, where we give polynomial upper bounds for the BDD sizes during the verification process, as well as for the time and space complexity. Here, we prove the polynomial verifiability for several handcrafted approximate adders, which divide the adder into subadders, including the approximate adders ACAI [9], ACAII [10], GeAr [11], ETAII [12] and GDA [13]. Furthermore, we prove the polynomial verifiability of automatically generated approximate adders, where a set of gates from conventional exact adders is altered [14] [15]. Here, the approximate adders are polynomially verifiable if arbitrary gates are deleted and the type of arbitrary gates is changed, where the RCA, the CSA and the CLA are considered as exact adders. Therefore, the verification process of each approximate adder that results from any approximation technique, which deletes gates and changes the type of gates in the beforementioned exact adders, is guaranteed to run efficiently in polynomial time and space. The results also show that a regular RCA, CSA and CLA can be verified polynomially even if they contain bugs, where the bugs can consist of deleted and changed gates. To evaluate the upper bounds given in this paper, we experimentally evaluate the upper bounds for the BDD sizes during the verification process, as the overall time and space complexities heavily depend on the BDD sizes.

This work was supported by the German Research Foundation (DFG) within the Project *PLiM* (DR 287/35-1) and the Reinhart Koselleck Project *PolyVer* (DR 287/36-1).

II. Preliminaries

A. Adder Function

The adder function adds an incoming carry bit c_{-1} and two input numbers a and b with n bits respectively. Thus, it has $2n + 1$ input bits: $(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c_{-1})$. Its output is the sum of a, b and c_{-1}, which has $n + 1$

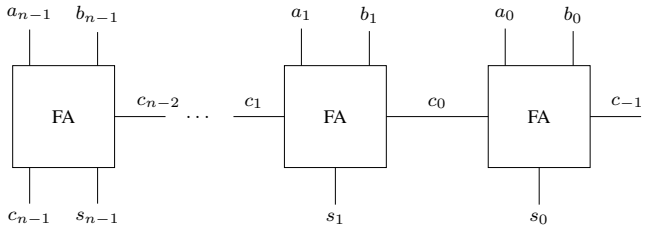

Fig. 1: Ripple Carry Adder

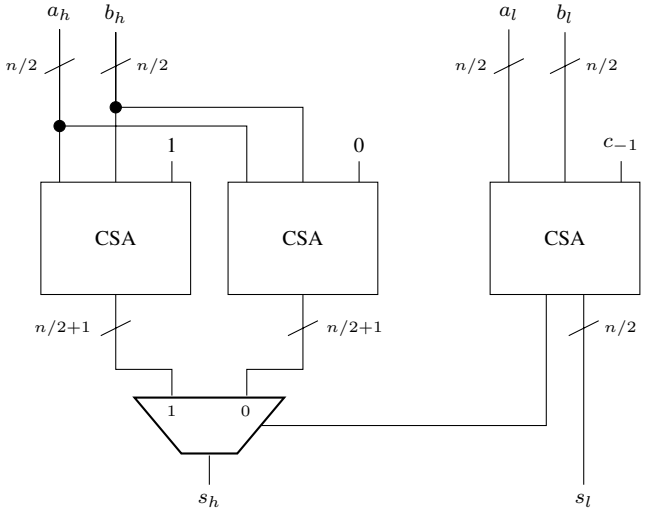

Fig. 2: Conditional Sum Adder

bits: $(c_{n-1}, s_{n-1}, \ldots, s_0)$. Here, c_{n-1} is the carry output bit and $s_{n-1}, \ldots, s_0$ are the sum bits. The sum and carry bits can be calculated as follows:

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$$
$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

Thus, the carry bit c_i, as well as the sum bit s_i can be calculated using a_i, b_i and c_{i-1}.

B. Adders

Several state-of-the-art adder architectures exist, of which we present the three most prominent ones in this section, where two are based on *Full Adders* (FAs). An FA has three inputs a_i, b_i and c_{i-1} and two outputs s_i and c_i, which represent the calculated sum of the three inputs. Here, c_{i-1} is the carry-in signal and c_i is the carry-out signal [16].

1) Ripple Carry Adder: The *Ripple Carry Adder* (RCA) consists of n FAs, which are connected through a carry chain. The general structure of a RCA is shown in Figure 1: The i-th FA has the inputs a_i,b_i and c_{i-1} and the computed carry-out c_i is passed on to the next FA. Furthermore, the i-th FA also computes the sum bit s_i [16] [17].

2) Conditional Sum Adder: The *Conditional Sum Adder* (CSA) is defined recursively as shown in Figure 2. The lower halfs of the inputs are added by a CSA, whereas the higher halfs of the inputs are added by two CSAs in parallel: One CSA has its input carry bit set to 1 whereas the other CSA has its input carry bit set to 0. The higher half of the sum is then determined using a multiplexer and the carry-out

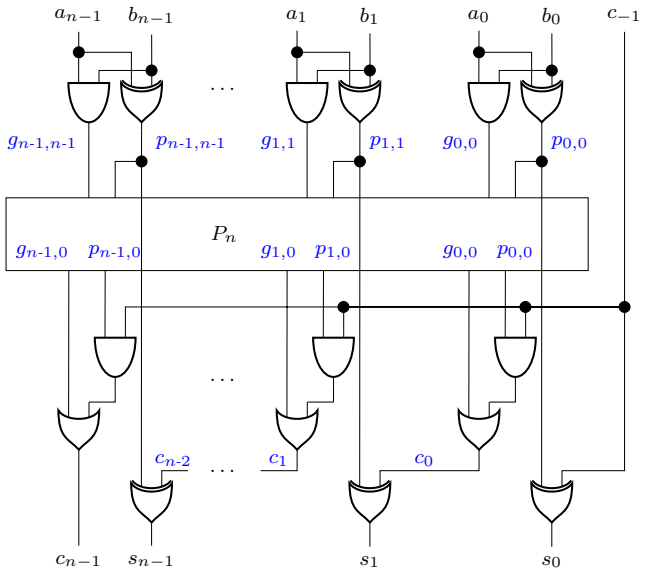

Fig. 3: Carry Look Ahead Adder

of the lower half of the sum. In the last recursive step, the CSAs are 1-bit adders, which are realized as FAs. Thus, a CSA with inputs of bitwidth n consists of one layer of FAs and $\lceil \log n \rceil$ layers of multiplexers [16] [17].

3) Carry Look Ahead Adder: The *Carry Look Ahead Adder* (CLA) first computes all carry bits using a prefix computation. To compute the carry bits, two functions are calculated: the propagate function p and the generate function g. If $0 \leq i \leq k < j < n$, p and g are defined as follows:

$$p_{i,i} = a_i \oplus b_i$$
$$g_{i,i} = a_i \cdot b_i$$
$$p_{j,i} = p_{j,k+1} \cdot p_{k,i}$$
$$g_{j,i} = g_{j,k+1} + p_{j,k+i} \cdot g_{k,i}$$

The carry bit c_i and the sum bit s_i can then be computed with

$$c_i = g_{i,0} + p_{i,0} \cdot c_{-1}$$
$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

The structure of the CLA is shown in Figure 3, where the intermediate results are marked in blue. First, all $g_{i,i}$ and $p_{i,i}$ are computed and then, using the prefix computation, all $g_{i,0}$ and $p_{i,0}$ are computed, from which the carry bits and subsequently the sum bits are computed [16] [17].

C. Approximate Adders

Approximate Computing (AC) is often applied if the exact output of a function is not required, but an approximate output is sufficient, which is beneficial in applications with e.g. hardware or time restrictions [6]. Approximate adders calculate the sum of two numbers with the possibility of some output bits being approximated.

A common method for constructing approximate adders is the division of the adder into several subadders and the cutting of the carry chain between these subadders in order to reduce the depth and therefore the runtime. There exist several

state-of-the-art approximate adders which utilize the described method, e.g. the approximate adders ACAI [9], ACAII [10], GeAr [11], ETAII [12] and GDA [13].

Another possibility for the reduction of area or delay is the alteration of regular adders by removing or changing the type of gates. Here, an algorithm is used to determine a set of gates which are to be altered such that the area or delay is reduced while ensuring a low error [14] [15]. If a gate is removed, the output of this gate is replaced by one of its inputs. If the type of a gate is altered, it is changed from e.g. an AND gate to an OR gate. This alteration can be done with every type of regular adder, e.g. the RCA, CSA or CLA.

D. Binary Decision Diagrams

A *Binary Decision Diagram* (BDD) is a directed acyclic graph representing a Boolean function, which consists of internal nodes as well as terminal nodes. Each internal node is associated with an input variable x_i and has two output edges. Furthermore, the terminal nodes represent the constant functions 0 and 1.

In an *Ordered Binary Decision Diagram* (OBDD), the variables appear in the same order for every path in the BDD, where we call the nodes for a variable x_i the i-th level of the BDD. The order of the variables is given by the variable ordering $(x_1, x_2, \ldots, x_n)$. In addition to being ordered, a *Reduced Ordered Binary Decision Diagram* (ROBDD) consists of a minimal amount of nodes and is canonical for a given variable ordering [1]. A *Quasi Reduced Ordered Binary Decision Diagram* (QROBDD) is canonical as well, given a variable ordering. Here, the BDD is not fully reduced, as the outgoing edges of each node in the i-th level of the BDD lead to nodes in the $(i+1)$-th level [18].

We denote $|f|$ as the size of the ROBDD of the function f with respect to the number of nodes.

Important operations on BDDs include the compose operation, which replaces a variable x_i in a BDD for a function f with the function g, which we denote as $f|_{x_i=g}$. This operation has a complexity of $\mathcal{O}(|f|^2 \cdot |g|)$ [1].

III. Related Work

In the past years, the polynomial verifiability of several circuits has been proven, including several state-of-the-art adders. The RCA, CSA and CLA have been proven to be polynomially verifiable using BDDs in [3], while the authors of [4] give specific polynomial bounds for the verification time of the CSA. Furthermore, some prefix adders were proven to be polynomially verifiable in [5], including the Serial Prefix Adder, the Ladner-Fischer Adder and the Kogge-Stone Adder.

Additionally, it has been shown that Wallace-tree like multipliers are polynomially verifiable using *BMDs [19] and that integer arithmetic circuits are polynomially verifiable with respect to the circuit size [20]. Furthermore, the verification of tree-like circuits and circuits derived from BDDs have shown to be polynomially bounded [21], as well as circuits for symmetric functions [22].

Thus, the polynomial verifiability of several circuits has been shown. However, the verification complexity of approximate adders has not been researched yet.

IV. Polynomial Verification

In this section, we show the polynomial verifiability of several state-of-the-art approximate adders, including several handcrafted approximate adder architectures that divide the adder into subadders, specifically the approximate adders ACAI, ACAII, GeAr, ETAII and GDA. Furthermore, we examine approximate adders which are generated automatically.

A. Handcrafted Approximate Adders

We inspect the verification complexity with respect to the bitwidth of the inputs for two kinds of handcrafted approximate adders. Firstly, for several handcrafted approximate adders, all outputs are merely computed by one subadder, e.g. for the approximate adders ACAI, ACAII and GeAr.

Theorem 1. *Handcrafted approximate adders are polynomially verifiable with respect to the bitwidth of the inputs, if each output is computed by one subadder, resulting in at most n subadders, where n is the bitwidth of the inputs a and b.*

Proof. As each output is computed by one subadder, the maximum BDD size during the verification of these approximate adders is the maximum BDD size during the verification of each subadder, which we call $|SA|_{max}$.

Let the subadders with inputs of bitwidth w be realized with an adder that is verifiable in time $\mathcal{O}(w^d)$ for some constant d. Then, the verification of each subadder has to be carried out. As the approximate adder consists of at most n subadders, where n is the bitwidth of the inputs to the approximate adder, the resulting time complexity is $\mathcal{O}(n \cdot w^d)$.

Furthermore, if the subadders are verifiable in a space complexity of $\mathcal{O}(w^d)$, the space complexity is bounded by $\mathcal{O}(n \cdot w^d)$ as well. □

Furthermore, in some approximate adders, each output is computed by at most two subadders which are connected through a carry chain, e.g. for the ETAII and GDA approximate adders. Here, the carry generator computes a carry signal, which is the carry-in signal of the sum generator, which then computes the outputs.

Theorem 2. *Handcrafted approximate adders are polynomially verifiable with respect to the bitwidth of the inputs, if each output is computed by two subadders, which have disjoint input variables and are connected through a carry chain.*

Proof. For all outputs which only depend on one subadder, the results from Theorem 1 can be applied. For all outputs which depend on two subadders, meaning a carry generator and a sum generator, the subadders can first be verified individually, where the carry-in of the sum generator is set to a variable c and is represented by the topmost node in the BDD. Then, the BDDs for the outputs can be computed using the compose operator on the BDDs, where the variable c is replaced with the BDD for the carry signal, computed by the carry generator. As

the BDDs generated from the carry and sum generators have disjoint variables, the resulting BDD after the application of the compose operator consists of the BDD computed for the sum generator, where the node for the variable c is replaced with the BDD computed for the carry generator. Let $|SG|_{max}$ be the maximum BDD size during the verification process of the sum generator and let $|CG|_{max}$ be the maximum BDD size during the verification process of the carry generator. Then, the maximum BDD size during the verification process of the approximate adder is bounded by $|SG|_{max} + |CG|_{max} - 3$, where 3 is substracted because of the replaced carry-in node and the terminal nodes of the carry generator.

Let $|SG|$ and $|CG|$ be the BDD sizes of the sum generator and carry generator respectively, before the application of the compose operator. Each compose operation requires $|SG|^2 \cdot |CG|$ steps. As $|SG|$ and $|CG|$ are sum and carry bits and therefore linear [3], each compose operation has a time complexity of $\mathcal{O}(n^3)$. Let the sum generators with inputs of bitwidth w_1 be realized with an adder that is verifiable in time $\mathcal{O}(w_1^{d_1})$ for some constant d_1 and let the carry generators with inputs of bitwidth w_2 be realized with an adder that is verifiable in time $\mathcal{O}(w_2^{d_2})$ for some constant d_2. Then, the time complexity for the verification of the approximate adder is bounded by $\mathcal{O}(n \cdot w_1^{d_1} + n \cdot w_2^{d_2} + n^4)$, as at most n carry generators and sum generators have to be verified respectively, followed by at most n compose operations with a time complexity of $\mathcal{O}(n^3)$.

Furthermore, let the sum generators with inputs of bitwidth w_1 be realized with an adder that is verifiable in a space complexity of $\mathcal{O}(w_1^{d_1})$ and let the carry generators with inputs of bitwidth w_2 be realized with an adder that is verifiable in a space complexity of $\mathcal{O}(w_2^{d_2})$. Then, the space complexity is bounded by $\mathcal{O}(n \cdot w_1^{d_1} + n \cdot w_2^{d_2} + n^4)$ as well. □

B. Altered Ripple Carry Adder

As the regular RCA is polynomially verifiable [3], we show the polynomial verifiability of a circuit which results from a RCA which is approximated by the alteration of gates, such as in [14] [15]. Here the altered RCA is polynomially verifiable with respect to the bitwidth of the inputs a and b if an arbitrary amount of gates is removed and the type of an arbitrary amount of gates is changed, regardless of the type it is changed into. Note that we first analyze the amount of internal nodes of the BDD and add the terminal nodes at the very end.

Theorem 3. *The BDD for each output of an FA with 3 variables as inputs has at most 5 internal nodes after the deletion or change of an arbitrary amount of gates.*

Proof. As the FA has three inputs and the deletion or change of gates cannot increase the amount of inputs, an alteration of the FA by deleting or changing the type of one or multiple gates can only lead to an unreduced BDD with at most 7 internal nodes for the sum bit and the carry bit respectively. The lowest level of an unreduced BDD with 3 inputs has at most 4 nodes, however, the reduction of the BDD reduces the possible number of nodes in the lowest level to 2. Thus, a reduced BDD with 3 inputs and therefore also the BDD for each output of an altered FA, has at most 5 internal nodes: one in the top level, two in the second level and two in the third level. □

Using Theorem 3, the polynomial verifiability of the altered RCA, as well as upper bounds for the BDD sizes, the time complexity and the space complexity can be shown.

Theorem 4. *An altered RCA is polynomially verifiable with respect to the bitwidth n.*

Proof. As shown in Theorem 3, despite the alteration of the FAs, the BDDs of the altered FAs have a constant size of at most 5 internal nodes. From the BDDs of the altered FAs, the BDD for the whole altered RCA can be constructed using the compose operator: Let f_i be the function of the carry output bit of the i-th FA with the inputs c_{i-1}, a_i, b_i which outputs the carry bit c_i and let f_{i+1} be the function of either the carry or sum output bit of the $(i+1)$-th FA with the inputs c_i, a_{i+1}, b_{i+1}. The BDD for the combination of both FAs can then be computed with the compose operator: $f_{i+1}|_{c_i = f_i}$.

As the variables of both FAs are disjoint, the variable for c_i in the BDD of f_{i+1} is simply replaced by the BDD of f_i and the resulting BDD has a size of $|f_i| + |f_{i+1}| - 1$.

We start the composition with $f_i = f_0$ and $f_{i+1} = f_1$, meaning the first two FAs. Then, the resulting BDD is composed into the BDD for the third FA, followed by the fourth, etc., until all FAs are combined. Here, the compose operator is applied $n-1$ times until all FAs are combined. If an FA is fully deleted, it is skipped and not combined with the other FAs. Therefore, each BDD of s_i and c_i for every i has a maximum of $4(i+1)+1$ internal nodes, as all nodes for the carry inputs except the very first one, meaning the carry-in of the whole adder, are replaced during the composition process and every pair of a_i and b_i therefore adds at most 4 nodes to the BDD. Thus, the final BDD has at most $4n+1$ internal nodes, resulting in $4n+3$ nodes including the terminal nodes.

The time complexity for the verification of the altered RCA is $\mathcal{O}(n + n \cdot n) = \mathcal{O}(n^2)$, as the BDDs for n altered FAs with a constant size have to be computed and the compose operator has to be carried out a linear amount of times, once for every carry and sum bit. Here, each compose operator requires at most $|f_{i+1}|^2 \cdot |f_i| = \mathcal{O}(n)$ steps, as f_{i+1} is always an altered FA, which has a constant size as shown in Theorem 3, whereas $|f_i|$ is always linear. Thus, the time complexity is $\mathcal{O}(n + n \cdot n) = \mathcal{O}(n^2)$.

For every carry and sum bit, only a constant BDD for an FA and a linear BDD for the previously computed carry-bits have to be kept in the memory during the verification process, resulting in a space complexity of $\mathcal{O}(n^2)$. □

C. Altered Conditional Sum Adder

Like the RCA, the CSA has proven to be polynomially verifiable [3] [4]. Since the CSA consists of FAs and multiplexers, we show that it is polynomially verifiable if arbitrary gates in the FAs and in the multiplexers are deleted or changed. For the proof, we only use the structure of the CSA and not the specific

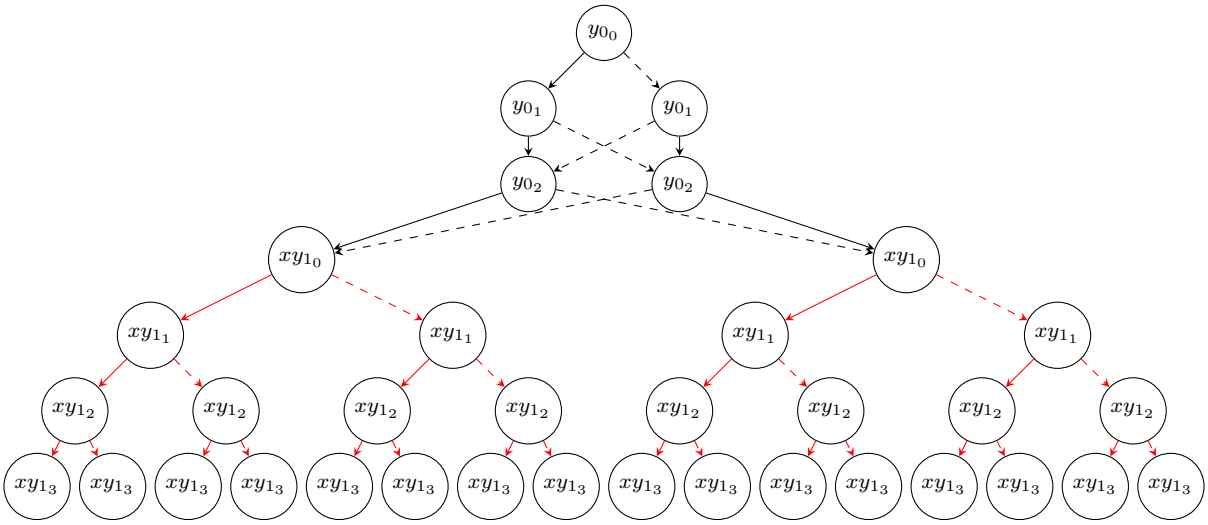

Fig. 4: Example BDD after the application of Theorem 5

gates and therefore, we prove the polynomial verifiability after arbitrary alterations. Again, we first focus on the internal nodes and add the terminal nodes at the end. Before the proof of the polynomial verifiability of the altered CSA, we first introduce a necessary theorem.

Theorem 5. *Let a QROBDD have $2i+1$ input variables $(y_0, x_1, y_1, \ldots, x_i, y_i)$, where the sum of all nodes for every x_j is u. Furthermore, let every input variable be replaced with a function with at most three input variables such that the input variables of the functions which replace y_0 and each pair (x_j, y_j) are disjoint, meaning the input variables of the functions which replace a pair (x_j, y_j) are disjoint from all other input variables. Let $\hat{u}$ be the number of internal nodes in the QROBDD for the function which replaces y_0. Additionally, let each pair (x_j, y_j) be replaced by functions such that both functions together have a total of at most 4 input variables. Then, the resulting QROBDD has at most $\hat{u}+15u$ internal nodes.*

Proof. During the composition of the top variable y_0, the respective node of the BDD is replaced by the QROBDD for the function which the variable is composed with, as its input variables are disjoint from all other input variables. Thus, the topmost node of the BDD is replaced with $\hat{u}$ nodes.

Now let (x_j, y_j) with $1 \leq j \leq i$ be a pair of input variables which are to be replaced. As is given, the pair (x_j, y_j) has a total of at most 4 input variables. Furthermore, they do not share any input variables with any other pair $x_{j'}, y_{j'}$ with $1 \leq j' \leq i$ and thus, the composition process does not affect any other parts of the BDD.

First, we compose x_j with its respective function. Here, the nodes for x_j are simply replaced by the BDDs for the respective function, as their input variables are not yet present in the current BDD. Afterwards, we compose y_j with its respective function. As it may share several input variables with the function of x_j, the nodes cannot be simply replaced with the BDDs for the respective function, as the variables may already be included in the BDD and a simple replacement would result in multiple BDD levels with the same variable. However, as the pair (x_j, y_j) has a total of at most 4 input variables, each node for x_j becomes the root of at most a full binary tree with 4 inputs, which has 15 nodes. As the input BDD is a QROBDD, each node for y_j is the direct child of a node for x_j and therefore, the binary trees contain each node for y_j after the composition process. Furthermore, the variable ordering of the 4 input variables can be chosen freely, as every ordering results in at most a full binary tree.

Thus, the BDD has a maximum of $\hat{u}+15u$ internal nodes during and after all compositions, as the top node of the BDD is replaced with $\hat{u}$ nodes and each pair (x_j, y_j) is replaced with at most $15u_j$ nodes, where u_j is the number of nodes in the BDD level for the variable x_j. The resulting BDD can be reduced to a QROBDD, as y_0 is replaced by a QROBDD and the full binary trees can be reduced to QROBDDs. □

Figure 4 shows an example for a BDD after the composition of all variables. y_0 is replaced with a function on the variables $(y_{0_0}, y_{0_1}, y_{0_2})$ which has a QROBDD with 5 internal nodes, resulting in $\hat{u}=5$. The pair x_1, y_1 is replaced by functions with the input variables $(xy_{1_0}, xy_{1_1}, xy_{1_2}, xy_{1_3})$. Here, the exact functions with which x_1 and y_1 are composed, as well as the terminal nodes, are omitted for clarity and to highlight the binary tree structure. As can be seen, the two nodes which were originally the nodes for x_1 are now the root of full binary trees with 4 inputs $(xy_{1_0}, xy_{1_1}, xy_{1_2}, xy_{1_3})$, both of which are marked with red edges. Thus, the total size of the BDD, excluding the terminal nodes, is $\hat{u}+15u = 5+15\cdot 2 = 35$ and the BDD can be reduced to a QROBDD.

We now show that the altered CSA is polynomially verifiable by showing that Theorem 5 can be applied multiple times for the verification and that the resulting BDDs are always polynomial.

Theorem 6. *An altered CSA is polynomially verifiable with respect to the bitwidth n.*

Proof. A CSA consists of a layer of FAs and $\lceil \log n \rceil$ layers

of multiplexers. We now show that the outputs of the t-th multiplexer have a BDD with at most a polynomial amount of nodes, despite any alterations. Furthermore, we show that this bound is not exceeded during the computation.

Firstly, there are $2n-1$ FAs, one for (a_0, b_0) with a carry-in variable and two for every $0 < i < n$. As it has three inputs, the BDD for the first altered FA with the carry-in variable has a size of at most 5 internal nodes, as shown in Theorem 3. As the carry-in is a constant for all other altered FAs, the BDD for both the sum bit and the carry bit of each altered FA has only two variables and therefore at most 3 internal nodes.

For the first layer of multiplexers, we compute the BDD for each output with the input variables set to the select bit and the two respective inputs of the multiplexer. As it has three input variables, its QROBDD has a size of at most 7 internal nodes with 2 nodes in the second level. Afterwards, we compose the inputs with their respective functions, meaning with the outputs of the layer of FAs. The select bit can be composed with the output of an FA with at most 3 inputs, whereas both inputs to the multiplexer are the outputs of an FA, one of them having their carry-in set to 1 and the other having their carry-in set to 0. As both of these FAs have the same inputs (a_j, b_j), together both variables in the BDD are replaced by a total of 2 variables. Thus, Theorem 5 can be applied, where $\hat{u} = 7$ and where $u^0 = 2$ is the value of u before the application and conclude, that the QROBDD of each multiplexer output has a size of at most $7 + 15 \cdot u^0 = 37$ internal nodes which is not exceeded during the computation. Including the terminal nodes, this evaluates to a maximum BDD size of $7 + 15 \cdot u^0 + 2 = 39$.

For the outputs of the t-th layer of multiplexers, the BDD can be generated similarly. Again, the QROBDD for the multiplexer with its variables set to the select bit and the two inputs is computed first, which has at most 7 internal nodes with 2 nodes in its second level. As specified in the architecture of the CSA, the select bit results from a multiplexer in the $(t-1)$-th layer with the inputs $(c_i, c_{j|c_i=1}, c_{j|c_i=0})$, one of the other inputs results from a multiplexer with the inputs $(c_{k|c_j=0}, s_{l|c_k=1}, s_{l|c_k=0})$ and the final input results from a multiplexer with the inputs $(c_{k|c_j=1}, s_{l|c_k=1}, s_{l|c_k=0})$. Thus, the inputs of the select bit are disjoint from all other inputs and the other two inputs have a total of 4 variables. Thus, Theorem 5 can be applied and the intermediate result has $7 + 15 \cdot u^0 = 37$ internal nodes, where the variable ordering $(c_i, c_{j|c_i=1}, c_{j|c_i=0}, c_{k|c_j=0}, c_{k|c_j=1}, s_{l|c_k=1}, s_{l|c_k=0})$ can be chosen. Afterwards, the resulting BDD is reduced to a QROBDD. Now, Theorem 5 can be applied again with $u^1 = 2 + u^0 \cdot 5$, where u^1 is the value of u after one application of Theorem 5, as the composition of the select input has added a pair of variables where the first variable of this pair has 2 nodes and furthermore, every binary tree of 15 nodes has two pairs of variables of which the top ones have 1 node and 4 nodes respectively. Theorem 5 is applied until the last layer of multiplexers is reached and the outputs of the FAs are composed. Here, $u^{t'} = 2 + u^{t'-1} \cdot 5$ is the value of u after t' applications of Theorem 5. For the outputs of the t-th layer of multiplexers, Theorem 5 has to be applied t times until the full BDDs are computed. The final BDD size including the terminal nodes is polynomial and is not exceeded during the computation process:

$$\begin{aligned} & 7 + 15 \cdot u^{t-1} + 2 \\ =& 9 + 15 \cdot (2 + 2 \cdot 5 + 2 \cdot 5^2 \cdots + 2 \cdot 5^{t-1}) \\ \leq& 9 + 30 \cdot t \cdot 5^{t-1} \end{aligned}$$

Here, $7 + 15 \cdot u^{t-1}$ is the number of internal nodes of the BDD after the last application of Theorem 5, whereas $u^{t-1} = (2 + 2 \cdot 5 + 2 \cdot 5^2 \cdots + 2 \cdot 5^{t-1})$ is the result for u after the $t-1$ previous applications of Theorem 5. Furthermore, the two terminal nodes are added.

As the altered CSA has $\lceil \log n \rceil$ layers of multiplexers, we can conclude that all altered final sum and carry bits have a BDD with at most a polynomial amount of nodes:

$$\begin{aligned} & 9 + 30 \cdot \lceil \log n \rceil \cdot 5^{\lceil \log n \rceil - 1} \\ \leq& 9 + 30 \cdot \lceil \log n \rceil \cdot 2^{\log n \cdot 3} \\ =& 9 + 30 \cdot \lceil \log n \rceil \cdot n^3 \end{aligned}$$

Thus, an upper bound for the BDD size during the verification process of the altered CSA is $9 + 30 \cdot \lceil \log n \rceil \cdot n^3$.

The time complexity of the verification process can be calculated as follows: The BDDs for a linear amount of FAs and a linear amount of multiplexers per multiplexer layer have to be generated, where each of these BDDs has a constant size, resulting in a time complexity of $\mathcal{O}(n \cdot \log n \cdot n) = \mathcal{O}(\log n \cdot n^2)$ for this step. Furthermore, the BDDs for a linear amount of outputs have to be generated. For every multiplexer layer, a linear amount of variables have to be replaced using the compose operator. Thus, a total of $\mathcal{O}(n \cdot \log n \cdot n) = \mathcal{O}(\log n \cdot n^2)$ compose operations have to be conducted. Using the compose operator, each composition requires at most $(9 + 30 \cdot \lceil \log n \rceil \cdot n^3) \cdot (9 + 30 \cdot \lceil \log n \rceil \cdot n^3) \cdot 7 = \mathcal{O}(\log^2 n \cdot n^6)$ steps. Thus, the overall time complexity of the verification is $\mathcal{O}(\log n \cdot n^2 + \log n \cdot n^2 \cdot \log^2 n \cdot n^6) = \mathcal{O}(\log^3 n \cdot n^8)$.

The space complexity of the verification process is equal to the complexity of the compose operators and is therefore $\mathcal{O}(\log^2 n \cdot n^6)$, as the compose operations have the largest upper bound for the space complexity during the verification process of the altered CSA. □

D. Altered Carry Look Ahead Adder

The regular CLA is polynomially verifiable as shown in [3]. We now show that it is still polynomially verifiable if an arbitrary amount of gates is deleted or changed with the constraint that the computation of each carry bit has a logarithmic depth in the CLA. The proof again relies merely on the structure of the CLA and uses Theorem 5.

Theorem 7. *An altered CLA is polynomially verifiable with respect to the bitwidth n if the calculation of each $c_{i,0}$ has a logarithmic depth.*

Proof. For an unaltered CLA, the carry bits are computed as follows: $c_i = g_{i,0} + (p_{i,0} \cdot c_{-1})$. Thus, we compute the QROBDD for each c_i with the input variables $(c_{-1}, p_{i,0}, g_{i,0})$,

which consists of at most 7 internal nodes. Here, $p_{i,0}$ consists of two variables, as it is defined as $p_{i,0} = p_{i,k+1} \cdot p_{k,0}$ for the unaltered CLA, whereas $g_{i,0}$ consists of 3 variables, as it is defined as $g_{i,0} = g_{i,k+1} + (p_{i,k+1} \cdot g_{k,0})$. Thus, $p_{i,0}$ and $g_{i,0}$ have a total of 4 variables and therefore, Theorem 5 can be applied for the composition of $p_{i,0}$ and $g_{i,0}$, where $u^0 = 2$ and $\hat{u} = 1$, as the variable c_{-1} does not have to be replaced. This results in a QROBDD with a maximum of $1 + 15 \cdot u^0$ internal nodes, where the variable ordering $(c_{-1}, p_{k,0}, g_{k,0}, p_{i,k+1}, g_{i,k+1})$ is chosen. In this variable ordering, each pair again depends on a total of 4 variables and each pair has disjoint variables. Thus, Theorem 5 can be applied again with $u^1 = u^0 \cdot 5$.

The application of Theorem 5 can be repeated until the inputs to the BDD are $p_{j,j}$ and $g_{j,j}$ for $j \leq i$. Then, all these inputs can be composed using Theorem 5, resulting in the BDD for c_i. Here, $u^{t'} = u^{t'-1} \cdot 5$ is the value of u after t' applications of Theorem 5. Let t be the number of applications of Theorem 5 which are needed until the BDD for c_i is computed. Then, the final BDD has a size of

$$\begin{aligned} & 1 + 15 \cdot u^{t-1} + 2 \\ =& 3 + 15 \cdot 2 \cdot 5^{t-1} \\ =& 3 + 30 \cdot 5^{t-1} \end{aligned}$$

Here, $1 + 15 \cdot u^{t-1}$ is the final application of Theorem 5, where $u^{t-1} = 2 \cdot 5^{t-1}$. Furthermore, the two terminal nodes are added.

The exact amount of applications needed for the calculation depends on the value of k. However, typically the CLA has a logarithmic depth and a logarithmic amount of applications are needed, until the BDD has the inputs $p_{j,j}$ and $g_{j,j}$ for $j \leq i$, resulting in $\lceil \log n \rceil + 1$ applications of Theorem 5 and a maximum BDD size of

$$\begin{aligned} & 3 + 30 \cdot 5^{\lceil \log n \rceil + 1 - 1} \\ \leq& 3 + 150 \cdot 2^{3 \cdot \log n} \\ =& 3 + 150 \cdot n^3 \end{aligned}$$

In the unaltered CLA, the sum outputs are computed with $s_i = a_i \oplus b_i \oplus c_{i-1}$. The BDD for the altered $a_i \oplus b_i \oplus c_{i-1}$ has at most 5 internal nodes if c_{i-1} is a variable, as it only has 3 inputs. Since the variables are disjoint, the BDD for the altered c_{i-1} can simply replace the variable and therefore, the final BDD, including the two terminal nodes, has a size of at most $4 + 3 + 150 \cdot n^3 = 7 + 150 \cdot n^3$, which is not exceeded during the verification process.

The time complexity for the verification process of the altered CLA can be calculated as follows: Each c_i has to be generated, as well as each s_i. For each c_i, a logarithmic amount of applications of Theorem 5 are needed, where a linear amount of compositions have to be conducted for each application. Here, each of these compositions requires a total of at most $(7 + 150 \cdot n^3) \cdot (7 + 150 \cdot n^3) \cdot 5 = \mathcal{O}(n^6)$ steps. Thus, the computation of all c_i requires a total of at most $\mathcal{O}(n \cdot \log n \cdot n \cdot n^6) = \mathcal{O}(\log n \cdot n^8)$ steps. Finally, for the computation of each s_i, one composition is required with a time complexity of $5 \cdot 5 \cdot (5 + 150 \cdot n^3) = \mathcal{O}(n^3)$, resulting in a time complexity of $\mathcal{O}(n \cdot n^3) = \mathcal{O}(n^4)$ for all s_i. Overall, the verification process of the altered CLA has a time complexity of $\mathcal{O}(\log n \cdot n^8 + n^4) = \mathcal{O}(\log n \cdot n^8)$.

Again, the compose operations have the highest space complexity during the verification process of the altered CLA and therefore, the space complexity of the verification process is $\mathcal{O}(n^6)$. □

V. Experiments

To evaluate the upper bounds for the verification process of the approximate adders given in Section IV, we have implemented the computation of the BDDs for all presented approximate adders in CUDD 3.0.0 [23]. Here, we evaluate the upper bounds for the sizes of the BDDs during the verification process, as the time and space complexity heavily depend on the BDD sizes.

A. Handcrafted Approximate Adders

For the approximate adders ACAI, ACAII, GeAr, ETAII and GDA, we evaluate two examples per approximate adder. As bitwidth we use $n = 16$ and we use two different exemplary values for the other parameters. For details on the parameters, see [9] [10] [11] [12] [13]. Here, the subadders of ACAI, ACAII and GeAr are realized with RCAs, as well as the sum generators of ETAII and GDA, whereas the carry generators of ETAII and GDA are realized with CLAs. We have implemented these approximate adders without an incoming carry bit, as they are defined as such.

For the evaluation of the computed upper bounds, the maximum sizes of the BDDs during the verification process of all subadders are needed. For a RCA with n inputs, the maximum BDD size is $3n + 4$ [3] with an incoming carry-bit, which is reduced to $3n + 2$ without an incoming carry-bit. Furthermore, CLAs are used as carry generators in the ETAII and GDA architectures. Without an incoming carry bit, the BDDs during the calculation of the carry bit have a maximum size of $3n + 1$ for n inputs [3].

Table I shows the results for the ACAI, ACAII, GeAr, ETAII and GDA. Here, the respective parameters for the adders are given, as well as the respective calculated upper bounds and the maximum BDD sizes during the verification process. The upper bound is computed using the maximum BDD sizes during the verification of the RCA and CLA. E.g., for the ETAII, the upper bound is computed as $|SG|_{max} + |CG|_{max} - 3 = (3\frac{n}{m} + 4) + (3\frac{n}{m} + 1) - 3$ and

TABLE I: Calculated upper bound and maximum BDD size for handcrafted approximate adders

Adder	Parameters	Upper Bound	Maximum BDD Size
ACAI	n=16, q=4	14	14
ACAI	n=16, q=8	26	26
ACAII	n=16, k=2	14	14
ACAII	n=16, k=4	26	26
GeAr	n=16, r=2, p=4	20	20
GeAr	n=16, r=6, p=4	32	32
ETAII	n=16, m=4	26	26
ETAII	n=16, m=2	50	50
GDA	n=16, m=4, p=4	26	26
GDA	n=16, m=4, p=8	38	38

TABLE II: Upper bound and maximum BDD size for automatically generated approximate adders with $n = 16$

Adder	Upper Bound	Maximum BDD size of unaltered adders	Maximum BDD size of approximate adders					
			Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
RCA	67	52	52	51	55	55	53	54
CSA	491,529	52	81	79	88	108	85	90
CLA	614,407	52	75	87	152	222	148	189

is therefore 26 for $n = 16, m = 4$ and 50 for $n = 16, m = 2$. From our results, it is apparent that the calculated upper bound coincides with the actual maximum BDD size for all test cases and all shown handcrafted approximate adders.

B. Automatically Generated Approximate Adders

For the evaluation of the upper bounds for the automatically generated approximate adders, we have implemented the regular adders RCA, CSA and CLA with 16-bit inputs and we delete and change the type of random gates. If the type is changed, it is changed into either an AND gate, an OR gate, an XOR gate, a NAND gate, a NOR gate or an XNOR gate. For the RCA, CSA and CLA, we evaluate the results for 600 approximate adders respectively, wich are divided into 6 cases:

1) 5% of gates are deleted
2) 10% of gates are deleted
3) 5% of gates are changed
4) 10% of gates are changed
5) 5% of gates are deleted and 5% of gates are changed
6) 10% of gates are deleted and 10% of gates are changed

For each case, 100 test cases are generated automatically, resulting in 600 test cases in total.

Table II shows the results for the automatically generated approximate adders where the RCA, CSA and CLA are approximated. For each adder, the calculated upper bound for the BDD size is shown, as well as the maximum BDD sizes during the verification process of the unaltered adder and the maximum BDD sizes during the verification process of the 100 approximate adders for each of the six cases. Here, Cases 1 and 2 represent approximate adders where gates are merely deleted, whereas gates are merely changed in the Cases 3 and 4. Finally, in the Cases 5 and 6, gates are both deleted and changed. It is apparent that the calculated upper bounds are not exceeded by any example.

For the RCA, the upper bound evaluates to $4n + 3 = 67$, whereas it evaluates to $9 + 30 \cdot \log 16 \cdot 16^3 = 491,529$ for the CSA and to $7 + 150 \cdot 16^3 = 614,407$ for the CLA. As can be seen, for the RCA, the upper bound overestimates the maximum BDD sizes during the verification process of all examples only slightly with the overall maximum BDD size being 55, whereas the upper bounds for the CSA and CLA overestimate the BDD sizes more significantly with the overall maximum BDD sizes being 108 and 222 respectively. This is due to the repeated application of Theorem 5, where the intermediate BDDs resulting from its application are not reduced in the calculation of the upper bound.

Furthermore, it can be seen that the mere deletion of gates can increase the BDD sizes during the verification process, but the change of gates increases the BDD sizes more significantly. In general, Case 4 results in the largest BDDs, which is the case where the type of 10% of gates is changed.

VI. Conclusion

In this paper, we have proven that the verification process of several approximate adders is guaranteed to have a polynomial time and space complexity with respect to the bitwidth of the inputs. These polynomially verifiable approximate adders include approximate adders that consist of several subadders, such as ACAI or ETAII, as well as approximate adders where the regular adders RCA, CSA and CLA are altered by the deletion or change of gates. For all these approximate adders, we have given polynomial upper bounds for the BDD sizes during the verification process, as well as for the time and space complexity.

In addition to the proof of the polynomial bounds given in this paper, we have experimentally evaluated these bounds. We have specifically evaluated the bounds for the BDD sizes during the verification process, as the time and space complexity heavily depend on them. In our experimental evaluation, we have exemplarily shown the correctness of our given bounds, as no test case has exceeded them.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[2] R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 205–213, 1991.

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

[4] A. Mahzoon and R. Drechsler, "Late breaking results: Polynomial formal verification of fast adders," in *DAC*, 2021, pp. 1376–1377.

[5] ——, "Polynomial formal verification of prefix adders," in *ATS*, 2021, pp. 85–90.

[6] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *ETS*, 2013, pp. 1–6.

[7] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *ICCAD*, 2011, pp. 667–673.

[8] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "ADAC: Automated design of approximate circuits," in *Computer Aided Verification*. Cham: Springer International Publishing, 2018, pp. 612–620.

[9] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *DATE*, 2008, pp. 1250–1255.

[10] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, 2012, pp. 820–825.

[11] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *DAC*, 2015, pp. 1–6.

[12] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *ISOCC*, 2010, pp. 323–327.

[13] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *ICCAD*, 2013, pp. 48–54.

[14] S. Froehlich, D. Große, and R. Drechsler, "Approximate hardware generation using symbolic computer algebra employing Groebner basis," in *DATE*, 2018, pp. 889–892.

[15] R. Hrbacek, V. Mrazek, and Z. Vasicek, "Automatic design of approximate circuits by means of multi-objective evolutionary algorithms," in *DTIS*, 2016, pp. 1–6.

[16] R. Zimmermann, "Binary adder architectures for cell-based VLSI and their synthesis," Ph.D. dissertation, Swiss Federal Institute of Technology, 1997.

[17] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. A.K. Peters, Ltd., 2002.

[18] N. Ishiura, "Synthesis of multilevel logic circuits from binary decision diagrams," *IEICE Transactions on Information and Systems*, vol. 76, pp. 1085–1092, 1993.

[19] M. Keim, R. Drechsler, B. Becker, M. Martin, and P. Molitor, "Polynomial formal verification of multipliers," *Formal Methods Syst. Des.*, vol. 22, no. 1, pp. 39–58, 2003.

[20] M. Barhoush, A. Mahzoon, and R. Drechsler, "Polynomial word-level verification of arithmetic circuits," in *MEMOCODE*, 2021, pp. 1–9.

[21] R. Drechsler, "Polynomial circuit verification using bdds," in *ICEECCOT*, 2021, pp. 49–52.

[22] R. Drechsler and C. Dominik, "Edge verification: Ensuring correctness under resource constraints," in *SBCCI*, 2021, pp. 1–6.

[23] "CUDD 3.0.0," https://github.com/ivmai/cudd.