PolyAdd: Polynomial Formal Verification of Adder Circuits

Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
drechsler@uni-bremen.de

Abstract

Only by formal verification approaches functional correctness can be ensured. While for many circuits fast verification is possible, in other cases the approaches fail. In general no efficient algorithms can be given, since the underlying verification problem is NP-complete.

In this paper we prove that for different types of adder circuits polynomial verification can be ensured based on BDDs. While it is known that the output functions for addition are polynomially bounded, we show in the following that the entire construction process can be carried out in polynomial time. This is shown for the simple Ripple Carry Adder, but also for fast adders like the Conditional Sum Adder and the Carry Look Ahead Adder. Properties about the adder function are proven and the core principle of polynomial verification is described that can also be extended to other classes of functions and circuit realizations.

I. Introduction

Ensuring the functional correctness of circuits and systems is one of the major challenges in today's circuit and system design. While simulation and emulation approaches reach their limits due to the complexity of the system under verification according to Moore's Law, only formal proof techniques can ensure correctness according to the specification (see e.g. [1], [2]). In these approaches proof engines, like BDD, SAT or SMT, are applied.

In practice these techniques work often well and can handle circuits of several million gates. But it might also happen that the proof fails due to run time or memory constraints. One of the major difficulties is that this can hardly be predicted resulting in non-robust behavior of the tools. For this, a deeper understanding is required which circuits can be handled efficiently and for which ones the formal approach will fail.

In the context of the highly relevant class of arithmetic circuits early studies on BDDs have shown that they are not well-suited to verify multipliers [3], but using dedicated data structures, like *BMDs [4] it was possible to represent the output functions of a multiplier polynomially. In [5] it has been shown that not only the outputs can be represented, but for a specific type of Wallace tree multiplier the complete verification can be carried out polynomially.

In this paper, we consider circuits for addition of two binary numbers. While it is well known that the BDD size for the adder function is only linear in the bit size [6], we show that the complete construction process of the BDD is also bounded polynomially. This is shown for three different adder architectures, namely the *Ripple Carry Adder* (RCA), the *Conditional Sum Adder* (CoSA) and the *Carry Look Ahead Adder* (CLA). Theoretical bounds on the BDD sizes are proven and it is shown that the complete symbolic simulation starting from the inputs to the outputs of the circuit can be carried out polynomially. Furthermore, for specific functions upper bounds on the BDD size are proven.

The paper is structured as follows: In Section II notations and definitions are reviewed to make the paper self-contained. The adder function and BDDs are introduced. For the three adders the circuit realization is reviewed in Section III. in Section IV for the three adder architectures it is proven that formal verification can be done efficiently. Finally, the results are summarized and open problems are addressed.

II. Notation and Definition

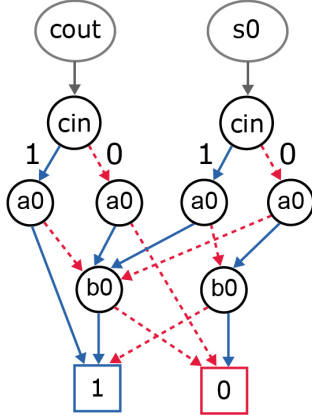Let $f : \mathbf{B}^n \to \mathbf{B}$ be a Boolean function over variable set $X_n = \{x_1, \ldots, x_n\}$.

Fig. 1. BDD for full adder

A. Adder Function

Let a, b and s be three binary numbers of n bits, where s is the sum of a, b and an incoming carry bit c_{-1}. The relation between the sum s and the operands a and b can be described by the following two equations:

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

$$\forall_{i=0}^{n-1} s_i = a_i \oplus b_i \oplus c_{i-1} \tag{2}$$

The variable c_i is called the i-th carry bit. The core cells of many adder architectures are the *Half Adder* (HA) and *Full Adder* (FA) cells realizing a 1-bit addition without or with carry input, respectively. The function table of the HA is shown in following table:

a_i	b_i	ha_1	ha_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

It is easy to see that the function ha_1 can be realized by an AND-gate, while ha_0 is described by an $\oplus$-gate, i.e.:

$$ha_1 = a_i \cdot b_i \qquad ha_0 = a_i \oplus b_i$$

For the FA with inputs a_i, b_i and c_{i-1} it holds:

$$fa_1 = a_i \cdot b_i + c_{i-1} \cdot (a_i + b_i) \qquad fa_0 = a_i \oplus b_i \oplus c_{i-1}$$

B. Binary Decision Diagrams

Reduced ordered *Binary Decision Diagrams* (BDDs) [6], [7] are *Directed Acyclic Graphs* (DAGs) where a Shannon decomposition

$$f = \overline{x}_i f_{\overline{x}_i} + x_i f_{x_i} (1 \leq i \leq n)$$

is carried out in each node.

Example 1. *The BDD for the FA is shown in Figure 1.*

An important property of BDDs is that the synthesis operations, like AND, OR or composition, can be carried out in polynomial time and space. This can be described by the operator *if-then-else* (ite) [6], [8][1]. A sketch of the algorithm is as follows, where *Rh* and *Rl* denote the high- and low-successors, respectively, and e.g. *F1i* is the cofactor to 1 with respect to variable i:

```
ite(F,G,H) {
 if (terminal case OR
     (F,G,H) in computed-table) {
  return result;
} else {
 let xi be the top variable of (F,G,H);
 Rh = ite(F1i,G1i,H1i);
 Rl = ite(F0i,G0i,H0i);
 if (Rh = Rl) return Rh;
 R = find_or_add_unique_table(v,Rl,Rh);
 insert_computed_table(F,G,H,R);
 return R;
 }
}
```

The *ite*-operator has a polynomial worst case behavior, i.e. for graphs F, G and H the result is bounded by $O(|F| \cdot |G| \cdot |H|)$. This bound holds under the assumption of an optimal hashing in $O(1)$. But also in the case of a worst case behavior of the hashing function, *ite* remains polynomial (see [9]).

C. Symbolic Simulation

To build the BDDs for the output signals of a circuit, the circuit is traversed in a topological order starting from the inputs. For the inputs signals the corresponding BDDs are initially generated. Then, for each gate in the circuit the corresponding synthesis operation based on *ite* is carried out. This process is called *symbolic simulation* in the following.

Example 2. *The symbolic simulation for a circuit consisting of a single AND gate is shown in Figure 2.*

III. Circuit Realization

In this section different realizations for adder circuits are briefly reviewed. Only the basic principles are reviewed as far as it is needed for making the paper self-contained. For more details see [10].

---

[1]Notice that in the following for the discussion and the proofs BDDs without complemented edges are considered.
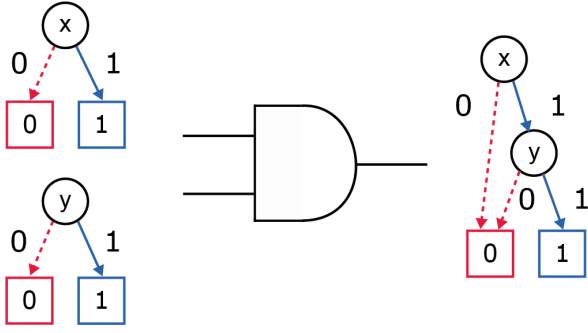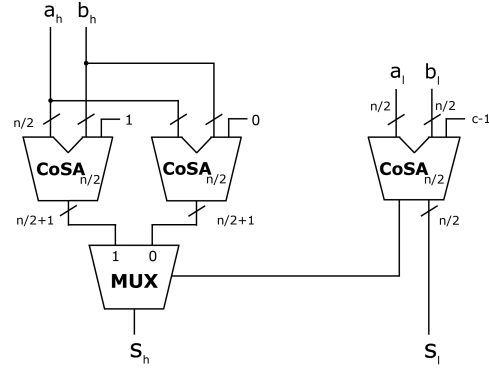
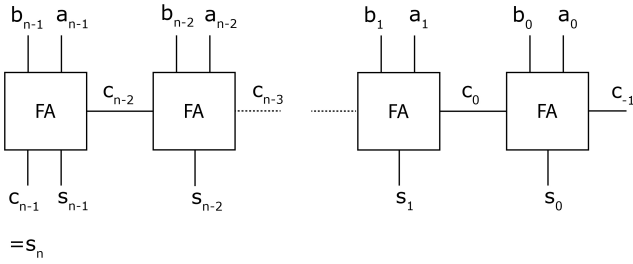**Fig. 2. Symbolic simulation for AND gate**

Fig. 3. Ripple Carry Adder

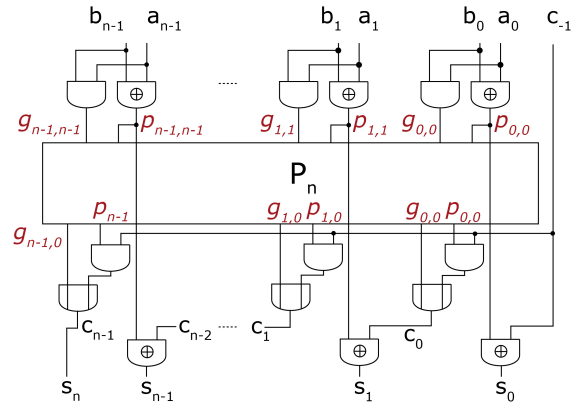
Fig. 4. Conditional Sum Adder

Fig. 5. Carry Look Ahead Adder

A. Ripple Carry Adder

The *Ripple Carry Adder* (RCA) simply consists of a sequence on n full adders. The cells are connected via the carry chain (see Figure 3).

The RCA is very area efficient, since it only requires a linear number of gates. But the RCA is also very slow, since the delay – measured in the number of gates that has to be traversed – is also linear in the number of inputs.

B. Conditional Sum Adder

The *Conditional Sum Adder* (CoSA) can be recursively described. While the lower $n/2$ bits are computed by a CoSA of bit-width $n/2$, for the higher $n/2$ bits the result is computed by two CoSAs in parallel, where one assumes an incoming carry, while the other does not. Thus, the adder makes use of the fact that the higher bits only depend on the incoming carry from the lower half. Both results are pre-computed and the correct result is selected by a multiplexer stage. The computation scheme is shown in Figure 4. For the 1-bit adders, simply full adders can be used.

The CoSA is a fast adder, i.e. it has a depth of $O(log(n))$. The circuit has a gate count of $O(n \cdot log(n))$.

C. Carry Look Ahead Adder

The *Carry Look Ahead Adder* (CLA) makes use of a fast prefix computation in a block P_n (see Figure 5). From Equation (2) it is obvious that it is sufficient to compute the carry bits c_i for all i. This can be done based on parallel prefix computation of the generation and propagation properties for addition. These are described by function g and p, respectively:

1) For $0 \le i < n$: $p_{i,i} = a_i \oplus b_i$, $g_{i,i} = a_i \cdot b_i$
2) For $i \le k < j$: $p_{j,i} = p_{k,i} \cdot p_{j,k+1}$,
   $g_{j,i} = g_{j,k+1} + (g_{k,i} \cdot p_{j,k+1})$,

This means that either a carry bit is generated in the upper part or a carry is generated in the lower part and is propagated through the higher part. Thus, the carry bits can be computed as ($0 \le i < n$):

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

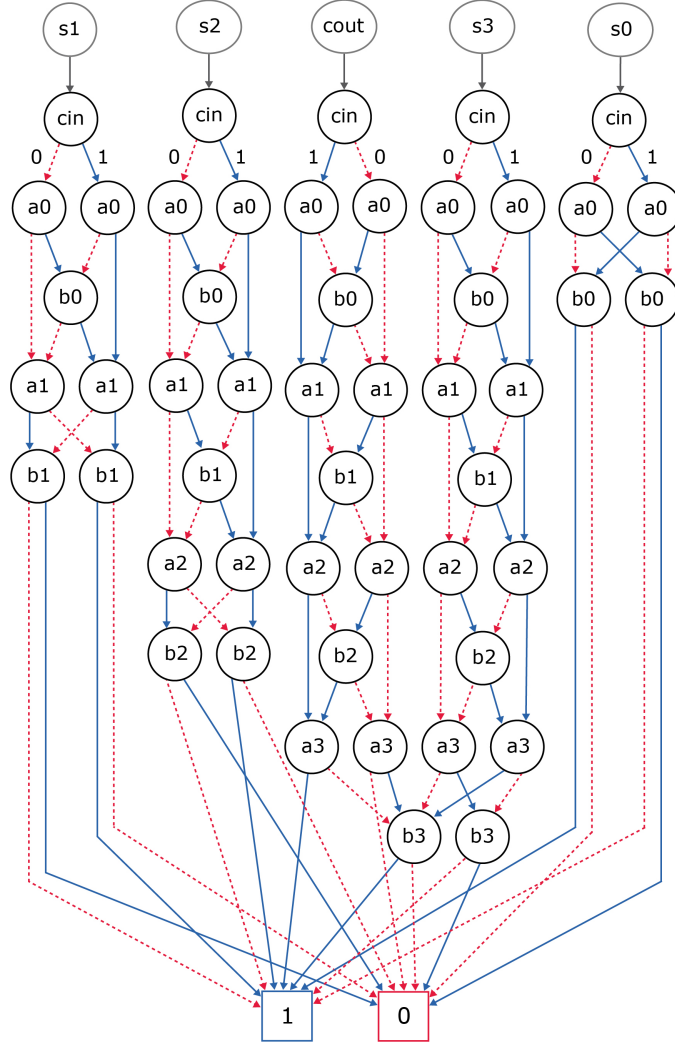The CLA has a logarithmic depth and a size linear in the number of input variables.

Fig. 6. BDD for 4-bit adder function

IV. Polynomial Verification

It is well known that the size of BDDs for the adder function is dependent on the variable ordering. It has also been proven that the BDD size is linearly bounded (see Section 4.4 in [11]), where exact estimates are given for BDD sizes. There, addition without the incoming carry bit has been considered. The results can be extended to also consider the incoming carry bit as it is required for all adder circuits in the following.

Theorem 1. 1) *The sum bit s_i of an adder has the BDD size bounded by $3 \cdot i + 7$.*
2) *The carry bit c_i of an adder has the BDD size bounded by $3 \cdot i + 6$.*

Proof. We use the interleaved variable ordering from the least to the most significant bits. For the sum bits the results

from Lemma 4.4.2 in [11] can be generalized, where an upper bound of $3 \cdot i + 5$ has been proven for the adder function without an incoming carry bit. For the additional carry bit two more nodes are required, i.e. one for the carry bit itself and one for the a_0 variable.

The same argument holds for the carry bit, but here on the lowest level one node is saved, since in case of generation by a_i and the incoming carry, b_i does not have to be tested any more (see Figure 6 for the case of 4 variables). □

It is important to notice that these results were always related to the representation size of the output functions, but not for the entire construction process.

Remark 1. *In the following, detailed bounds are not provided, since the goal of this paper is to show that the construction process is polynomial.*

Thus, it is sufficient to show that each individual step can be carried out in polynomial time and space. We make use of the following observation:

Remark 2. *If for each internal signal the size of the BDD representation and the number of gates in the circuit is polynomially bounded in the number of inputs n, the whole circuit can be formally verified in polynomial time due to the polynomially bounded synthesis operations on BDDs.*

This method can be applied to general circuits, but is used for adders only in the following. For the adder circuits from Section III the upper bounds hold, that each circuit only has a number of gates polynomial in the number of inputs n.

A. Ripple Carry Adder

For the RCA it is very simple to see that the complete construction is polynomially bounded. For the HA of the least significant bit and all FAs the BDD can be locally constructed and has only a constant size. Due to the structure of the RCA each carry output of a cell is connected to the carry input of the next cell. The substitution of the input variable can be carried out by the compose algorithm based on *ite* and has a polynomial worst-case complexity. Furthermore, according to Theorem 1 the size of the BDD for the carry signal for all i is always linear. Thus, the whole construction process is polynomially bounded, since the composition only has to be carried out n times.

Theorem 2. *The BDD for the RCA can be constructed polynomially.*

B. Conditional Sum Adder

The n bit CoSA consists of three CoSAs of bit-size $n/2$ and a multiplexer stage. From Theorem 1 it follows that each of the connecting signals shown in Figure 4 can be represented by a BDD of linear size. Only the carry inputs have to be set to 0 and 1, respectively. The only operation that has to be carried out is the one corresponding to the MUX unit. But this can be described by *ite* and is polynomially bounded. Thus, we obtain:

Theorem 3. *The BDD for the CoSA can be constructed polynomially.*

Remark 3. *The results of Theorems 2 and 3 can easily be generalized to further adder types that are based on full adders connected together using MUX cells, like e.g. the* Carry Select Adder *in [12] with a run time of* $O(\sqrt{n})$.

C. Carry Look Ahead Adder

In the CLA the sum bits are computed by determining the carry bits first and finally EXOR-ing them with the

corresponding a_i and b_i inputs according to Equation (2). Thus, the core circuit computes the carry bits starting based on the property of generation and propagation, i.e. functions p and g. The union of propagation intervals is based on Boolean AND-operations, i.e. larger interval only propagates a carry bit, if the left and the right part of the interval do so. For the generation part it holds that either the left part (using the higher bits) already propagates or the lower part generates, while the higher part propagates. In both cases, the structure consists of AND- and OR-operations only and it can be seen that the whole structure can be represented by BDDs of polynomial size. More formally, this can be proven as follows:

Lemma 1. 1) *Function $p_{j,i}$ has the BDD size bounded by $3 \cdot (j - i + 1)$ ($j > i$).*
2) *Function $g_{j,i}$ has the BDD size bounded by $3 \cdot (j - i) + 2$ ($j > i$).*

Proof. For function $p_{j,i}$ it holds:

$$p_{j,i} = (a_j \oplus b_j)(a_{j-1} \oplus b_{j-1}) \ldots (a_i \oplus b_i)$$

The BDD for the EXOR of two variables has three nodes. Since each variable only appears once, the corresponding BDDs can simply be connected (see Figure 7 for the case of 4 variables).

Since the BDD is a cannonical representation, in

$$g_{j,i} = g_{j,k+1} + (g_{k,i} \cdot p_{j,k+1})$$

the choice of k does not influence the BDD size and we choose $k = j - 1$ resulting in

$$g_{j,i} = g_{j,j} + (g_{j-1,i} \cdot p_{j,j}).$$

For each pair of variables a_l, b_l at most 3 nodes can be generated (see Figure 8 for the case of 4 variables). For the top variables even one more node is saved. $\square$

Based on this observation, the whole BDD for the CLA can be computed based on *ite*.

Theorem 4. *The BDD for the CLA can be constructed polynomially.*

V. Conclusion

In this paper it has been proven for three different adder architectures that the complete formal verification process can be carried out polynomially. It was proven that the underlying BDDs remain polynomial during the whole construction process. This was ensured by proving upper bounds on the BDD sizes for each internal signal. While the BDD sizes for the outputs of the adder functions were known to be polynomially bounded, this is the first time that for efficient adder circuits of logarithmic run time a polynomial proof process could be ensured.
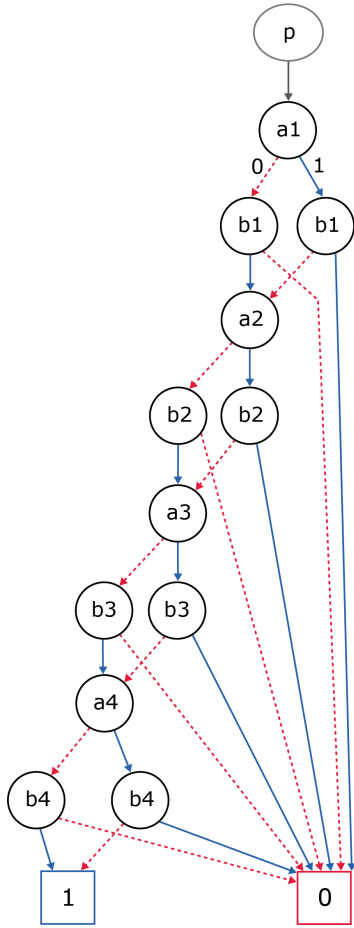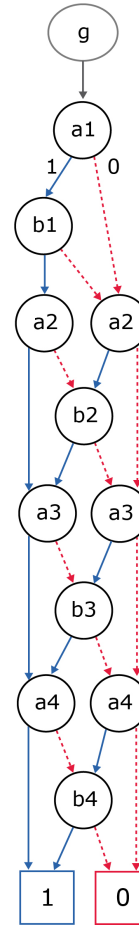
Fig. 7. BDD for p function for 4 variables

Fig. 8. BDD for g function for 4 variables

It is focus of future work to identify further classes of circuits and functions that can be polynomially verified using BDDs. Furthermore, alternative proof engines on the Boolean level, like SAT or O(K)FDDs, can be considered. Also extension to the word-level, like SMT or WLDDs, will be studied.

Acknowledgment

References

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

[2] ——, *Formal System Verification*. Springer, 2018.

[3] R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Comp.*, vol. 40, pp. 205–213, 1991.

[4] R. Bryant and Y.-A. Chen, "Verification of arithmetic functions with binary moment diagrams," in *Design Automation Conf.*, 1995, pp. 535–541.

[5] M. Keim, M. Martin, B. Becker, R. Drechsler, and P. Molitor, "Polynomial formal verification of multipliers," in *VLSI Test Symp.*, 1997, pp. 150–155. [Online]. Available: http://ira.informatik. uni-freiburg.de/papers/Year_97/KMBDM_97.ps.gz

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

[7] R. Drechsler and B. Becker, *Binary Decision Diagrams – Theory and Implementation*. Kluwer Academic Publishers, 1998.

[8] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in *Design Automation Conf.*, 1990, pp. 40–45.

[9] R. Drechsler and D. Sieling, "Binary decision diagrams in theory and practice," *Software Tools for Technology Transfer*, vol. 3, pp. 112–136, 2001.

[10] B. Becker, R. Drechsler, and P. Molitor, *Technische Informatik - Eine Einführung*. Pearson Studium, 2005.

[11] I. Wegener, *Branching Programs and Binary Decision Diagrams - Theory and Application*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[12] B. Becker, R. Drechsler, R. Krieger, and S. Reddy, "A fast optimal robust path-delay-fault testable adder," in *European Design & Test Conf.*, 1996, pp. 491–498.