Polynomial-Time Formal Verification of Adder Circuits for Multiple-Valued Logic

Philipp Niemann[*†] Rolf Drechsler[*†]
[*]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[†]Department of Computer Science, University of Bremen, Bremen, Germany
philipp.niemann@dfki.de, drechsler@uni-bremen.de

***Abstract*—Functional correctness of a circuit implementation can only be ensured by applying formal verification approaches. Although the underlying verification problem is NP-complete and, thus, no efficient verification is possible in general, for many circuits fast verification is indeed possible. There has been lots of work on formal verification of arithmetic circuits in the binary, two-valued logic domain. In this paper, we consider formal verification of adder circuits for multiple-valued logic and prove that for different types of adder circuits polynomial-time verification can be performed based on decision diagrams (DDs). While it is known that the output functions for addition are polynomially bounded, we show in the following that the entire construction process of the corresponding DD representation 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 Lookahead Adder.**

I. Introduction

Functional correctness is an essential quality measure in the design of circuits and systems and also one of the major design challenges. Due to the increasing complexity according to Moore's Law, simulation and emulation approaches quickly reach their limits and only formal proof techniques can ensure correctness according to the specification (see e.g. [1], [2]). In these approaches, techniques for efficient function representation, e.g. decision diagrams like *Binary Decision Diagrams* (BDDs) or *Multiple-valued Decision Diagrams* (MDDs, [3]), or formulations of the verification problem in terms of Boolean satisfiability (SAT, SMT) are employed together with highly elaborated proof engines [4].

In [5], it has been shown that polynomial-time verification is possible for adder circuits in the two-valued, Boolean logic domain using their representation as BDDs [6].

In this paper, we consider formal verification of adder circuits in the multiple-valued domain. We show that the results from [5] can be generalized to the multiple-valued domain using MDDs, i.e. the multiple-valued generalization of BDDs.

The remainder of the paper is structured as follows: Section II provides basic definitions of multiple-valued adders and their functional representation in terms of decision diagrams. In Section III we briefly recall a few adder realizations. In Section IV we formally prove that polynomial-time verification is possible for these adder realizations. Finally, conclusions are provided in Section V.

II. Background and Notation

In this section, we provide basic definitions of multiple-valued operators and adders and review MDDs as an efficient mean for representing and manipulating MVL functions.

A. Multiple-Valued Operators

For the construction of the adder circuits we use the following multiple-valued operators as defined in [7]:

Definition 1. *For $a, b \in \{0, \dots, p-1\}$ we define binary operators $SUM, \times, \oplus, \cdot$ as follows:*

- $SUM(a, b) = (a + b) \mod p$
- $a \times b = \begin{cases} 1 & a + b \geq p \\ 0 & \text{\textit{else}} \end{cases}$
- $a \oplus b = \max(a, b)$
- $a \cdot b = \max(0, a + b - (p - 1))$

Note that in the binary case, i.e. $p = 2$, the SUM operator as well as the $\oplus$ are equivalent to the logical OR, while the $\times$ and $\cdot$ operators are equivalent to the logical AND.

It is immediately clear from these definitions that all operators are commutative and SUM as well as $\oplus$ are associative. As shown in [7, Lemma 1], associativity also holds for $\cdot$, but clearly not for $\times$.

B. Multiple-Valued Addition

Let $a, b, s \in \{0, \dots, p-1\}^n$ be p-valued numbers of n digits with the interpretation as unsigned positive numbers

$$d_p = [d_{n-1}, \dots, d_0]_p = \sum_{i=0}^{n-1} d_i p^i$$

and $c_{n-1}, c_{-1} \in \{0, \dots, p-1\}$.

The function that maps $(a, b, c_{-1}) \mapsto (c_{n-1}, s)$ such that $[c_{n-1}, s_{n-1} \dots, s_0]_p = a_p + b_p + c_{-1}$ is called an n-digit adder (and is denoted ADD_n in the following). Here, c_{-1} is called *input carry* and c_{n-1} is called *output carry*.

Special cases are given by the *Half Adder* (HA), a 1-digit adder without an input carry, i.e. $c_{-1} = 0$, which realizes the addition of two p-valued digits, and the *Full Adder* (FA), which is a 1-digit adder of two p-valued digits and an input carry, i.e. ADD_1.

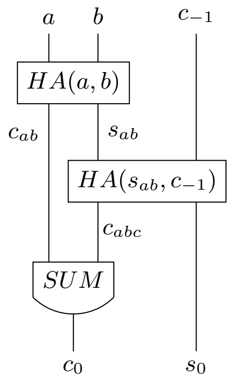


Fig. 1. Full Adder constructed from Half Adders and SUM operation

Using the operators from Section II-A, the functionality of Half and Full Adders can be described as follows:

$$HA(a,b) := (a \times b, SUM(a,b)) \quad \text{and}$$

$$FA(a,b,c_{-1}) := (c_0, s_0)$$

where $c_0 = SUM(a \times b, SUM(a,b) \times c_{-1})$ holds for the output carry and $s_0 = SUM(SUM(a,b), c_{-1}))$ for the sum.

For the Half Adder, the correctness of the definition can easily be verified:

$$[HA(a,b)]_p = p \cdot (a \times b) + SUM(a,b) = a + b.$$

For the Full Adder, the sub-term $SUM(a,b)$ shared by both outputs c_0, s_0 allow for a realization using two half adders and one SUM operation as shown in Fig. 1. This yields:

$$\begin{aligned}
[c_0, s_0]_p &= p \cdot c_0 + s_0 && \mid SUM(c_{ab}, c_{abc}) \\
&= p \cdot [(c_{ab} + c_{abc}) \mod p] + s_0 && \mid c_{ab}, c_{abc} \leq 1 \\
&= p \cdot (c_{ab} + c_{abc}) + s_0 \\
&= p \cdot c_{ab} + (p \cdot c_{abc} + s_0) && \mid HA(s_{ab}, c_{-1}) \\
&= p \cdot c_{ab} + (s_{ab} + c_{-1}) \\
&= (p \cdot c_{ab} + s_{ab}) + c_{-1} && \mid HA(a,b) \\
&= (a + b) + c_{-1}
\end{aligned}$$

Example 1. *The truth-table of a ternary (3-valued) Full Adder is given as follows:*

Inputs			Outputs	
c_{-1}	a	b	c_0	s_0
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
⋮	⋮	⋮	⋮	⋮
2	1	2	1	2
2	2	0	1	1
2	2	1	1	2
2	2	2	2	0

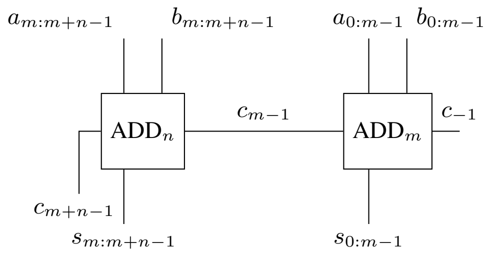


Fig. 2. Concatenating Adders

As can already be seen for this simple example, the output carry may not only assume the values 0 and 1 (as in the binary case), but can also become 2 for radices $p > 2$.

An $n + m$-digit adder, ADD_{n+m}, can be constructed by concatenating ADD_n and ADD_m, i.e. the output carry of ADD_m (summing up the m least significant digits) is used as input carry for ADD_n (which sums of the remaining n digits) as shown in Fig. 2.

C. Multiple-Valued Decision Diagrams

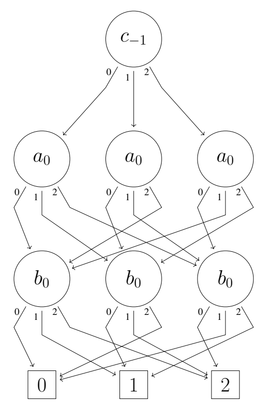
Multiple-Valued Decision Diagrams (MDDs) [8] are means for the efficient representation of MVL functions $f\colon \{0,1,\dots,p-1\}^n \to \{0,1,\dots,p-1\}$. MDDs are *Directed Acyclic Graphs* (DAGs) with up to p terminal nodes each labelled by a distinct logic value $0,1,\dots,p-1$. Each non-terminal node (labelled x_i) has p successors (termed as *children*), each representing a *co-factor* w.r.t. x_i. A co-factor $f_{x_i=v}$ ($v = 0,1,\dots,p-1$) of an n-ary function f w.r.t. variable x_i is obtained from f by assuming a fixed value for that variable, i.e. $x_i = v$. Thus, it is an $n-1$-ary function over variables $x_1 \dots, x_{i-1}, x_{i+1}, \dots, x_n$.

Typically, we consider reduced ordered MDDs in which redundant nodes (i.e., multiple nodes representing the same function or pointing to a single successor node) are removed and the variables occur in a fixed order on all paths from the root to the terminal nodes. These constitute canonical representations of MVL functions, i.e. given a function f and a fixed variable order there is a unique MDD representing f (up to isomorphism).

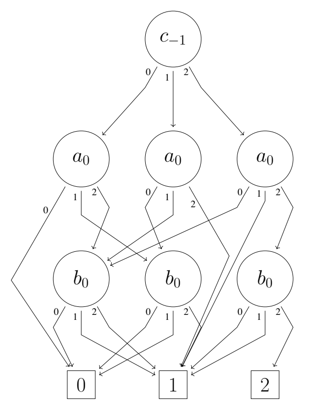
Example 2. *Reduced ordered MDDs for the sum and carry output of the ternary Full Adder from Example 1 are shown in Fig. 3. Both MDDs adhere to the variable order $c_{-1} < a_0 < b_0$. Note that the variable b_0 is skipped on some paths in the MDD in Fig. 3b. In fact, some outgoing edges of the a_0 nodes point directly to the terminal nodes indicating that the corresponding co-factors are constant and do not depend on b_0. For instance, in the case $c_{-1} = a_0 = 0$ which is represented by the leftmost path in the MDD, there will be no output carry regardless of the value of b_0. Thus, the edge from the leftmost a_0 node points directly to terminal 0.*

The complete function f can be re-constructed from its co-factors using the $CASE$ operator:

$$f = CASE(x_i, f_{x_i=0}, \dots, f_{x_i=p-1})$$



(a) Sum output



(b) Carry output

Fig. 3. MDDs for Ternary Full Adder outputs

Here, the $p + 1$-ary $CASE$ operator is given by the mapping $(s, f_0, \ldots, f_{p-1}) \to f_s$ such that the first argument specifies which of the following arguments is selected for the output. The $CASE$ operator is the MVL generalization of the well-known ITE operator for Boolean logic. Overall, MDDs are the multiple-valued generalizations of the well-known BDDs [6].

Another important property of MDDs is that the synthesis operations, like MIN, MAX, or composition, can be carried out in polynomial time and space using $CASE$ [9], [10]. A sketch of the algorithm is as follows, where $CHILD(G, i)$ denotes the i-th successor of the top node of the MDD G [8]:

```
CASE(A,B0,B1,...,Bp-1)
  if(terminal(A)) return (BA)
  if ((A,B0,...,Bp-1) in computed table)
    return result
  TOP=top variable of A,B0,B1,...,Bp-1
  for 0<=i<=p-1
    if(id(A)==TOP) EA=CHILD(A,i)
    else EA = A
    for 0<=j<p
      if(id(Bj)==TOP) EBj=CHILD(Bj,i)
      else EBj = Bj
     Ci=CASE(EA,EBj, ..., EBj)
  R=create node(TOP,C0,...,Cp-1)
  insert_computed_table(A,B0,...,Bp-1,R)
  return(R)
```

The $CASE$-operator has a polynomial worst case behavior, i.e. for graphs A, B_0, ..., B_{p-1} the result is bounded by $O(|A| \cdot \prod_{j=0}^{p-1} |B_j|)$. This bound holds under the assumption of an optimal hashing of intermediate results in a *computed table* with $O(1)$.

D. Symbolic Simulation

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

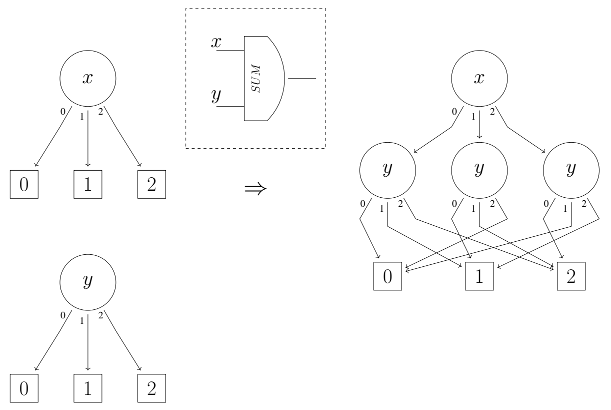


Fig. 4. Symbolic Simulation

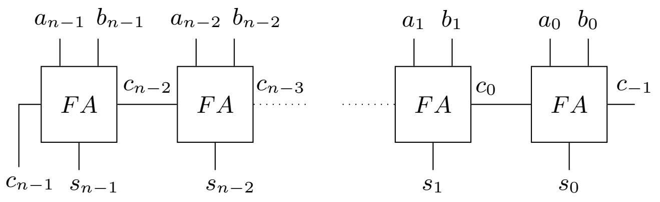


Fig. 5. Ripple Carry Adder

Example 3. *The symbolic simulation for a circuit consisting of a single SUM gate is shown in Fig. 4.*

III. Circuit Realizations

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 on the multiple-valued constructions see [7].

A. Ripple Carry Adder

According to the discussion in Section II-B, an n-bit adder can be constructed by concatenating a series of n full adders. The resulting circuit is shown in Fig. 5 and is termed *Ripple Carry Adder* (RCA), since the cells are connected via a carry chain.

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 basic idea of the *Conditional Sum Adder* (CoSA, [11]) is to speed up the computation by splitting up the addition of $2n$ digits into multiple additions of n digits (as shown in Fig. 6) which can be conducted in parallel. While the lower n digits can be computed straight-away, the addition of the higher n digits require the value of the input carry at digit n, i.e. the output carry of the lower n digits. In order to compute both halves in parallel, the sum of the higher n digits is computed multiple times in parallel, once for each possible value of the input carry at digit n, i.e. the output carry of the lower n digits. Once the correct value of the intermediate

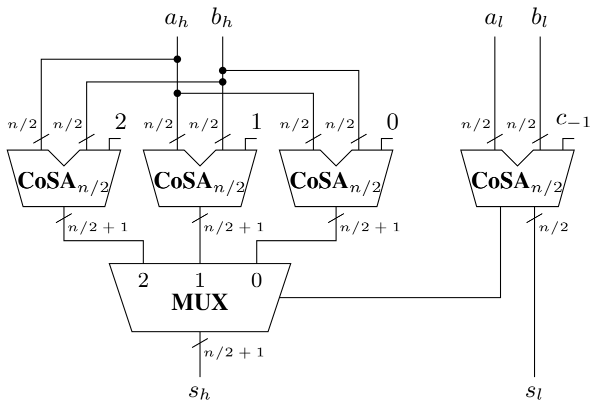


Fig. 6. Conditional Sum Adder

carry has been computed, a multiplexer is used to select the outputs of the corresponding addition of the higher n digits. The computation scheme is shown in Fig. 6.

Note that, while in the binary case the internal carrys may only assume the values 0 and 1, in the general MVL case they may also assume the value 2, namely if the input carry c_{-1} is greater than or equal to 2 (since $a_i + b_i \leq 2p - 2$). Thus, we require three computations for the higher n digits and a 3-to-1 multiplexer (MUX_3) can be used to select the appropriate output. This multiplexer can easily be described in terms of the $CASE$-operator: $MUX_3 = CASE(c_l, s_h^{c=0}, s_h^{c=1}, s_h^{c=2})$ where $s_h^{c=i}$ denotes the sum of the higher n digits given that the input carry of digit n assumes value i.

This approach can be repeated recursively by using CoSAs to compute the additions for n, $n/2$, $n/4, \ldots$ digits. For 1-digit adders, simply full adders can be used. By this, the depth of a CoSA becomes logarithmic in n, while the cost becomes worse, i.e. it grows faster than linearly (super-linearly) for increasing n. For more details on this, see [7].

C. Carry Lookahead Adder

The *Carry Lookahead Adder* (CLA, [12]) is a more sophisticated adder that combines logarithmic depth *and* linear cost. The basic idea of the construction is that it is sufficient to compute the internal carry signals c_i for all i, from which the output signals can be computed as

$$s_i = SUM(SUM(a_i, b_i), c_{i-1}). \tag{1}$$

Here, a parallel prefix computation based on the carry generation and propagation properties for addition is employed to efficiently compute the carry values (shown as a block P_n in Fig. 7).

More precisely, generation and propagation properties for addition can be described by function families g, p with $g = \{g_{j,i} \mid 0 \leq i \leq j < n\}$ and $p = \{p_{j,i} \mid 0 \leq i \leq j < n\}$, respectively, where

- $g_{j,i}$ indicates whether digits i to j generate a carry,
- $p_{j,i}$ indicates whether digits i to j propagate a carry.

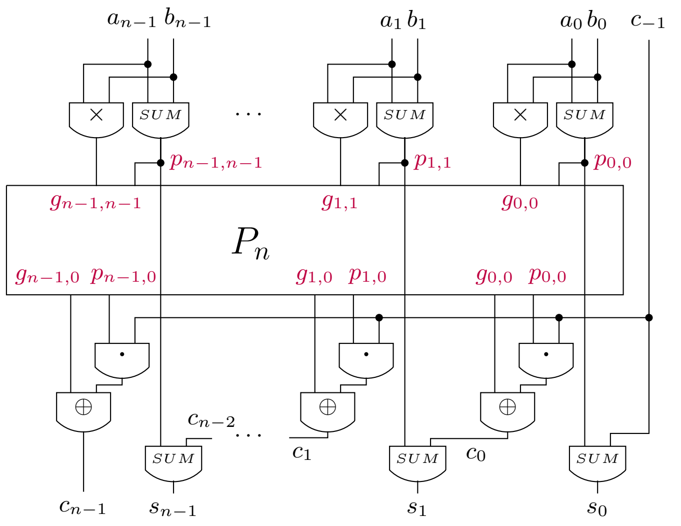


Fig. 7. Carry Lookahead Adder

It is clear that there can only be a carry at digit i if the digits 0 to i generate a carry (independently from the value of the input carry c_{-1}) or if they propagate the input carry c_{-1}:

$$c_i = g_{i,0} \oplus (p_{i,0} \cdot c_{-1}) \tag{2}$$

The required functions $g_{i,0}$ and $p_{i,0}$ can be computed as

$$(g_{i,0}, p_{i,0}) = (g_{i,i}, p_{i,i}) \circ \ldots \circ (g_{1,1}, p_{1,1}) \circ (g_{0,0}, p_{0,0})$$

using an appropriate associative operator as $\circ$. The functions on the right-hand side of the equation can be computed directly as $p_{j,j} = SUM(a_j, b_j)$ and $g_{j,j} = a_j \times b_j$ (for $j = 0, \ldots, i$), while a proper definition of $\circ$ is given by

$$(g_2, p_2) \circ (g_1, p_1) := \big(g_2 \oplus (g_1 \times p_2), p_2 \cdot p_1\big) \tag{3}$$

Since $\circ$ is associative, an efficient parallel prefix computation P_n is possible which computes all functions $g_{i,0}, p_{i,0}$ simultaneously in linear time and logarithmic depth.

IV. Polynomial Verification

It is well known from the binary world that the size of BDDs for the adder functions largely depend on the variable ordering. The naive ordering $a_0 < a_1 < \ldots < a_{n-1} < b_0 < \ldots < b_{n-1}$ leads to an exponential size, while an interleaved ordering $c_{-1} < a_0 < b_0 < a_1 < b_1 < \ldots$ allows for linear-sized representations.

In fact, only two different nodes at level a_i are required: one represents the case that there is an input carry c_{i-1} and the other represents the case that there is no input carry. At most three nodes are required at level b_i, one for each possible value of $c_{i-1} + a_i \in \{0, 1, 2\}$. Note, however, that the node for value 0 is redundant, since it is clear that regardless of the value of b_i no output carry will result ($0 + b_i < 2$). The same holds for value 2, since this already guarantees an output carry. The only exception is for b_j in the BDD for the j-th sum bit s_j where two nodes are required at b_j level, since the nodes for 0 and 2 would be identical. Overall, the BDD for

s_j requires $3(j+1)+2$ non-terminal nodes, while the BDD for c_j requires $3(j+1)+1$ nodes.

This argumentation can be generalized to the multiple-valued case in a straightforward fashion. Using the interleaved variable ordering, only 3 different nodes at level a_i are required ($i > 0$), one for each possible value of the input carry c_{i-1} which can assume the values 0, 1, and also 2. Since c_{-1} is unrestricted, there are p different nodes required at a_0 level. At level b_i there are at most $p+2$ different nodes required, one for each possible value of $c_{i-1} + a_i \in \{0, \ldots, p, p+1\}$. Note, however, that the node for value 0 is redundant, since it is clear that regardless of the value of b_i no output carry will result ($0 + b_i < p$). The same holds for value p, since this already guarantees that the output carry will have value 1. The only exception is for b_j in the MDD for the j-th sum digit s_j where p nodes are required, since the nodes for 0 and p as well as 1 and $p+1$ would be pairwise identical. For instance, the case for $p = 3$ and $j = 0$ is depicted in Fig. 3.

Overall, there are $p+3$ nodes required for each pair of variables a_i,b_i ($i > 0$) together with a single root node labelled c_{-1} and $2p$ nodes labelled a_0, b_0. This is summarized in the following

Theorem 1 (MDD sizes for carry and sum outputs)**.**

1) *The MDD for the j-th carry digit c_j requires*

$$(p+3) \cdot j + 2p + 1 \text{ nodes.}$$

2) *The MDD for the j-th sum digit s_j requires*

$$(p+3) \cdot j + 2p + 1 \text{ nodes.}$$

It is important to note that these results are only related to the representation size of the output functions (regardless of what kind of adder realization is chosen), but not for the BDDs/MDDs that need to be constructed during the entire construction process, i.e. the symbolic simulation of the circuit. Since the primary goal of this paper is to show that the construction process is polynomial, it is sufficient to show that each individual step can be carried out in polynomial time and space. Detailed bounds are not required for the proof and will, thus, not be provided in the following.

We will make use of the following general observation:

Observation 1. *If for each internal signal of a circuit the size of the corresponding MDD representation and the number of gates in the circuit are 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 MDDs.*

This observation can be applied to general circuits, but is used for adders only in the following. It can be readily checked that for the adder circuits from Section III the required property is satisfied that each circuit consists of a number of gates that is polynomial in the number of inputs.

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 digit and all FAs the MDD can be locally constructed and has only a constant size. According 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 CASE and has a polynomial worst-case complexity. Figuratively speaking, the composed MDD is obtained by redirecting all edges pointing to a terminal node in the MDD for the carry output to the corresponding successor of the root node of the other MDD.

Furthermore, according to Theorem 1 the size of the MDD 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 MDD for the multiple-valued RCA can be constructed polynomially.*

B. Conditional Sum Adder

The n-digit CoSA consists of four CoSAs of size $n/2$ and a multiplexer stage. From Theorem 1 it follows that each of the connecting signals shown in Figure 6 can be represented by an MDD of linear size. Only the carry inputs have to be set to 0,1 and 2, respectively. The only operation that has to be carried out is the one corresponding to the MUX unit. But this can be described by CASE and is polynomially bounded. Thus, we obtain:

Theorem 3. *The MDD for the multiple-valued CoSA can be constructed polynomially.*

C. Carry Lookahead Adder

In the CLA the sum digits are computed by determining the carry digits first and finally SUM-ing them with the corresponding a_i and b_i inputs according to Equation 1.Thus, the core circuit computes the carry digits based on the properties of generation and propagation, i.e. functions g and p.

It is clear from Fig. 7 that the construction of the outer part can be conducted in polynomial time and space as long as the outputs of the P_n block are given by MDDs of polynomial size, since there at most three gates ($\cdot$, $\oplus$ and SUM) between the outputs of P_n and any primary output. Thus, we can restrict our consideration to the P_n block.

It has been argued in [7] that the generation/propagation properties for an interval of digits $[i, j]$ can be computed from the properties for sub-intervals $[i, k]$ and $[k, j]$ as follows:

$$g_{j,i} = g_{j,k+1} \oplus (g_{k,i} \times p_{j,k+1}) \tag{4}$$

$$p_{j,i} = p_{k,i} \cdot p_{j,k+1} \tag{5}$$

Thus, all signals computed within P_n using applications of $\circ$ essentially represent a function $g_{j,i}$ or $p_{j,i}$ for some i, j. Since there are only polynomially many of such signals, it only remains to show that these can be represented by MDDs of polynomial size. This is proven in the following

Lemma 1.

1) *Function $p_{j,i}$ has the MDD size bounded by*

$$(p+1)+(j-i)(p^2+p)$$

2) *Function $g_{j,i}$ has the MDD size bounded by*

$$(p+1)+(j-i)(2p+1)$$

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

$$p_{j,i} = p_{j-1,i} \cdot p_{j,j}$$

As can be seen, $p_{j,j}$ only depends on a_j and b_j, while $p_{j-1,i}$ does not depend on a_j or b_j. Thus, the MDD for $p_{j,i}$ can be constructed by conjunction. This means, each terminal node of the MDD for $p_{j-1,i}$ (representing the value k) is replaced by an MDD representing the function $k \cdot p_{j,j}$ (as illustrated in the top part of Fig. 8). The MDD for $p_{j,j}$ has 1 node labelled a_j and p nodes labelled b_j. All MDDs for $k \cdot p_{j,j}$ have at most the same size, since they can be obtained by replacing terminal values $t \mapsto k \cdot t$.[1] Thus, the MDD grows by $p \cdot (p+1)$ nodes labelled a_j or b_j and the overall size can iteratively be computed as $(p+1)+(j-i)\cdot p \cdot (p+1)$.

Regarding the function $g_{j,i}$, we consider the case $k = j-1$ for Equation (4):

$$g_{j,i} = g_{j,j} \oplus (g_{j-1,i} \times p_{j,j})$$

Here, $p_{j,j}$ and $g_{j,j}$ only depend on a_j and b_j, while $g_{j-1,i}$ does not depend on a_j or b_j. Thus, the MDD for $g_{j,i}$ can be constructed by conjunction (as illustrated in the bottom part of Fig. 8). Since g can only assume values 0 and 1, we only need construct two MDDs for $g_{j,j} \oplus (0 \times p_{j,j})$ and $g_{j,j} \oplus (1 \times p_{j,j})$, both have size at most $1+p$. More precisely, there is no b_j node for the case that $a_j = g_{j-1,i} = 0$, since then $0 \times p_{j,j} = 0 = g_{j,j}$ regardless of the value of b_j. Thus, the MDD grows by $2p+1$ nodes labelled a_j or b_j and the overall size can iteratively be computed as $(p+1)+(j-i)(2p+1)$. $\square$

Based on this observation, the whole MDD for the CLA can be computed based on CASE.

Theorem 4. *The MDD 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 shown that for arbitrary radices the MDD sizes for the outputs of the adder functions are polynomially bounded. This was so far only known for binary adders. Moreover, it was proven that the underlying MDDs remain polynomial during the whole construction process. This was ensured by proving upper bounds on the MDD sizes for each internal signal. This is the first time that for efficient MVL adder circuits of logarithmic run time a polynomial proof process could be ensured. As future work tighter bounds will be considered and also alternative adder architectures will be studied as has been done in the binary case in [13] and [14], respectively.

[1] In fact, they have the same size, since there will not be any redundant nodes after this transformation, but this is not relevant for the proof.

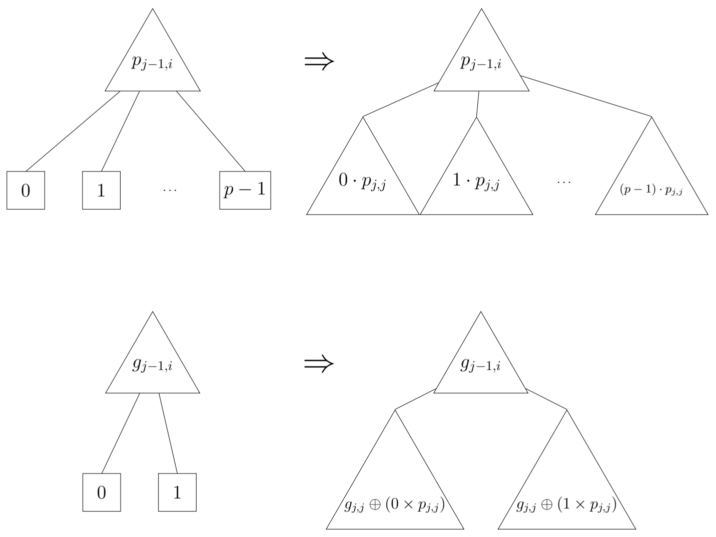


Fig. 8. MDDs for propagation and generation functions

Acknowledgements

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

References

[1] R. Drechsler, *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.
[2] ——, *Formal System Verification*. Springer, 2018.
[3] R. E. Bryant, "Binary decision diagrams and beyond: enabling technologies for formal verification," in *Int'l Conf. on CAD*. IEEE, 1995, pp. 236–243.
[4] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining SAT and computer algebra," in *Int'l Conf. on Formal Methods in CAD*. IEEE, 2019, pp. 28–36.
[5] R. Drechsler, "Polyadd: Polynomial formal verification of adder circuits," in *DDECS*. IEEE, 2021, pp. 99–104.
[6] R. Drechsler and B. Becker, *Binary Decision Diagrams - Theory and Implementation*. Springer, 1998.
[7] P. Niemann and R. Drechsler, "Synthesis of asymptotically optimal adders for multiple-valued logic," in *Int'l Symp. on Multiple-Valued Logic*. IEEE, 2021, pp. 178–182.
[8] D. M. Miller and R. Drechsler, "On the construction of multiple-valued decision diagrams," in *Int'l Symp. on Multiple-Valued Logic*. IEEE Computer Society, 2002, pp. 245–253.
[9] R. Drechsler, M. A. Thornton, and D. Wessels, "Mdd-based synthesis of multi-valued logic networks," in *Int'l Symp. on Multiple-Valued Logic*. IEEE Computer Society, 2000, pp. 41–46.
[10] B. Becker and R. Drechsler, "Decision diagrams in synthesis - algorithms, applications and extensions," in *Int'l Conf. on VLSI Design*. IEEE Computer Society, 1997, pp. 46–50.
[11] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, vol. EC-9, no. 2, pp. 226–231, 1960.
[12] A. Weinberger and J. L. Smith, "A one-microsecond adder using one-megacycle circuitry," *IRE Transactions on Electronic Computers*, vol. EC-5, no. 2, pp. 65–73, 1956.
[13] A. Mahzoon and R. Drechsler, "Late breaking results: Polynomial formal verification of fast adders," in *Design Automation Conf.* IEEE, 2021, pp. 1376–1377.
[14] ——, "Polynomial formal verification of prefix adders," in *Asian Test Symp.* IEEE, 2021, pp. 85–90.