Polynomial Formal Verification of Multipliers*

Martin Keim[1] Michael Martin[2] Bernd Becker[1] Rolf Drechsler[1] Paul Molitor[2]

[1]Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg i. Br.
<name>@informatik.uni-freiburg.de

[2]Institute of Computer Science
Martin-Luther-University
06099 Halle-Wittenberg
<name>@informatik.uni-halle.de

October, 1996

Abstract

Until recently, verifying multipliers with formal methods was not feasible, even for small input word sizes. About two years ago, a new data structure, called *Multiplicative Binary Moment Diagram* (*BMD), was introduced for representing arithmetic functions over Boolean variables. Based on this new data structure, methods were proposed by which verification of multipliers with input word sizes of up to 256 Bits is now feasible. Only experimental data has been provided for these verification methods until now.

In this paper, we give a formal proof that logic verification with *BMDs is polynomially bounded in both, space and time, when applied to the class of Wallace-tree like multipliers.

1 Introduction

Verifying that an implementation of a combinational circuit meets its specification is an important step in the design process. Often this is done by applying a set of test-input patterns to the circuit. With these patterns a simulation is performed to ensure oneself in the correct behavior of the circuit.

In the last few years new methods have been proposed for a *formal verification* of circuits concerning the logic function represented by the circuit, e.g. [FFK88], [MWBS88], [BCM90], [Bry92], [ABC+94]. These methods are based on *Binary Decision Diagrams* (BDDs) [Bry86]. But all these methods fail to verify some interesting combinational circuits, such as multipliers, since all BDDs for the multiplication function are exponential in size [Bry91].

Recently, a new type of decision diagrams the *Binary Moment Diagrams* (BMDs) [BC94] have been introduced. BMDs are a data structure for representing linear arithmetic functions with Boolean domain. Extending the BMD data structure by additional multiplicative edge-weights have led to so called *Multiplicative Binary Moment Diagrams* (*BMDs) [BC94]. *BMDs have been successfully used to verify the behavior of combinational arithmetic circuits such as multipliers with a hierarchical description [BC95]. Now it became possible to verify the behavior of some multiplier circuits with input word sizes of up to 256 bits [BC95]. Motivated by these results, several extensions of this approach have been discussed [DBR96], [CFZ95] and in the meantime integrated in systems for verification of arithmetic circuits [CB96]. Also floating point units have successfully been verified [CCH+96]. The verification method of [BC95] is based on partitioning the circuit into components with easy word-level specifications. First it is shown, that the bit-level implementation of a component module implements correctly its word-level specification.

*Research supported by DFG grant Mo 645/2-1 and BE 1176/8-2

Then the composition of word-level specifications according to the interconnection structure of the whole circuit is derived. This composition is an algebraic expression for which a *BMD is generated. Then this *BMD is compared with the one generated from the overall circuit specification.

One problem arising with this methodology is the need of high-level specifications of component modules. Another one is, that once having changed from the bit-level to the word-level, information about the exact ordering of connections at module boundaries is no longer accessible. Taking these problems into account, Hamaguchi et.al. have proposed another method for verifying arithmetic circuits. They called their method *verification by backward construction* [HMY95]. This method does not need any high-level information. We will give an overview of the method in Section 4. In [HMY95] only experimental data has been provided to show the feasibility of their method.

In this paper, we will give a formal proof that verification by backward construction is polynomially bounded with respect to input word size. We consider not a specific multiplier circuit, but the class of Wallace-tree like multipliers, e.g. [Wal64], [LV83]. Additionally, we consider not only the costs of intermediate results, but also the costs, that arise *during* the synthesis operation on the *BMDs.

The paper is organized as follows: In the next Section we briefly review the *BMD data structure. We give detailed insights in some basic operations on *BMDs. In Section 3 we introduce our example class of multiplier circuits. The method of backward construction applied to this class of multipliers is explained in Section 4. There, we also analyze the complexity of the method in terms of execution steps and space-requirement. In Section 5 we will summarize our results and show the application to verification. Finally, we point out directions of further research.

2 Multiplicative Binary Moment Diagrams

In this section, we give a short introduction how to represent integer-valued functions by the proposed decision diagram type *BMD. Details can be found in [BC94], [BC95].

*BMDs have been motivated by the necessity to represent functions over Boolean variables but having non-Boolean ranges. This means functions

$$f : \mathbb{B}^n \to \mathbb{Z}$$

$$f : \mathbb{B}^n \to \mathbb{Q}$$

mapping a Boolean vector onto integer numbers or onto rational numbers, respectively [BC94]. Here, we will restrict ourselves to integer functions.

For this class of functions the Boole-Shannon expansion can be generalized as

$$f = (1 - x) \cdot f_{\bar{x}} + x \cdot f_x,$$

where $\cdot, +$ and $-$ denote integer multiplication, addition and subtraction, respectively. Note, that x is interpreted here as an integer-valued function, having the image of zero and one. The Boole-Shannon expansion can be rearranged as

$$f = f_{\bar{x}} + x \cdot f_{\dot{x}}, \tag{1}$$

where $f_{\bar{x}}$ is called the *constant moment* and where $f_{\dot{x}} := f_x - f_{\bar{x}}$ is called the *linear moment* of f with respect to variable x. Equation (1) is the *moment decomposition* of function f with respect to variable x.

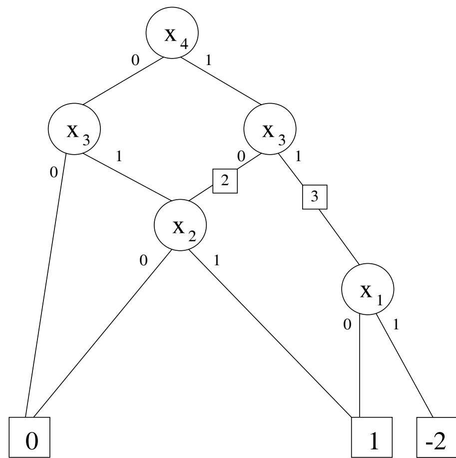

Figure 1: Representation of the function $x_2x_3 + 2x_2x_4 + 3x_3x_4 - 6x_1x_3x_4$ as a *BMD

*BMDs are rooted, directed, acyclic graphs with nonterminal and terminal vertices. Each nonterminal vertex has exactly two successors. The edges pointing to the successors are named low- and high-edge and the successors themselves are named low- and high-successor, respectively. The nonterminal vertices are labeled with Boolean variables the function depends on. The variables appear in the same ordering on every path from the root to the terminal vertices, i.e., the *BMD is ordered. The terminal vertices have no successor and are labeled with integer values. The low- (high-) edge of a nonterminal node points to the constant (linear) moment of the function represented by this node. The *BMD is reduced with respect to isomorphic subgraphs and unnecessary variables, i.e., there are no two distinct vertices that are roots of isomorphic subgraphs and there is no nonterminal vertex with the high-edge pointing to a terminal vertex with value 0. This would indicate a variable on which the function does not depend, as can be deduced from Equation1.

Additionally, a *BMD makes use of a common factor in the constant and linear moment. It extracts this factor and places it as a so called *edge-weight* on the incoming edge to the node. This could lead to smaller representations.

For example, a *BMD with root vertex v labeled with x and weight m on the incoming edge to v represents the function

$$f_v = m \cdot (f_{low(v)} + x \cdot f_{high(v)}), \tag{2}$$

In the following, we will no longer distinguish between the node v, labeled with a variable x and simply 'the node x', if the context is clear.

In this paper, we will consider integer edge-weights and integer values of terminal vertices since we are interested in integer linear functions only. To make the *BMD-representation canonical, a further restriction on the edge-weights is necessary. The edge-weights on the low- and high-edge of any nonterminal vertex must have *greatest common divisor* 1. Additionally, weight 0 appears only as a terminal value, and if either outgoing edge of a node points to this terminal node, the weight of the other outgoing edge is 1 [BC95].

As an example for a *BMD consider Figure 1. Low- and high-edges are labeled with 0 and 1, respectively. Edge-weights are written in square boxes on the corresponding edges and nodes are labeled with corresponding variables. Edges without square boxes have edge-weight 1.

```
subst(*BMD(f), x <-- *BMD(g))
{ 1     check for terminal cases;
  2     if (topvar(*BMD(f)) == x)
  3         S = add(low(*BMD(f)),mult(*BMD(g),high(*BMD(f))));
        else
        {
  4         E1 = subst(low(*BMD(f)), x <-- *BMD(g));
  5         E2 = subst(high(*BMD(f)), x <-- *BMD(g));
  6         E3 = MakeNode(topvar(*BMD(f)));
  7         E4 = mult(E2,E3);
  8         S = add(E1,E4);
        }
        return S;
}
```

Figure 2: Sketch of the substitution algorithm

2.1 Basic Operations on *BMDs

We now present in detail an algorithm, which is the basis for verification by backward construction.

The method of backward construction is based on substituting variables in a *BMD for a function f by a *BMD for another function g. According to Equation (1), this substitution is based on the following formula:

$$f|_{x \leftarrow g} = f_{\bar{x}} + g \cdot f_{\dot{x}} \tag{3}$$

A sketch of the substitution algorithm can be seen in Figure 2. It starts with a *BMD(f) and a *BMD(g), which is a notation for the *BMD representing the function f and g, respectively. Additionally, the variable x that is to be substituted, is handed to the algorithm. If the top variable of *BMD(f) is not the variable to be substituted, the algorithm calls itself recursively with the low- and high-successor. If the variable x is found, an integer multiplication and an integer addition is performed (line 3).

The algorithms *add* for integer addition and *mult* for integer multiplication are based on the following recursive formulae [BC94]:

$$f + g = (f_{\bar{x}} + g_{\bar{x}}) + x \cdot (f_{\dot{x}} + g_{\dot{x}}) \tag{4}$$

$$f \cdot g = (f_{\bar{x}} \cdot g_{\bar{x}}) + x \cdot (f_{\bar{x}} \cdot g_{\dot{x}} + f_{\dot{x}} \cdot g_{\bar{x}} + f_{\dot{x}} \cdot g_{\dot{x}}) \tag{5}$$

Remark:

- Both algorithms have exponential worst case behavior [End95].

In the following we show, that under specific conditions, the worst case will not occur. The key to this is a *BMD with a specific structure, which we call **S***um* **O***f weighted* **V***ariables (SOV)*. The *BMD for the *unsigned integer encoding* function

$$f = \sum_{i=0}^{n-1} a_i 2^i$$

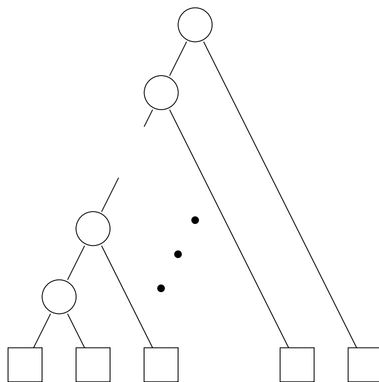

Figure 3: The SOV-structure

is is an example of a *BMD in SOV-structure. Its outline is given in Figure 3. A *BMD in SOV-structure has the following properties:

1. There is exactly one nonterminal node for each variable.
2. The high-edge of each nonterminal node points to a terminal node.
3. The high-edges are labeled with the weights 1 and point to a terminal node, marked with a value of the corresponding variables.
4. All but the last low-edge point to a nonterminal node. (The last one points to a terminal node.)

Because of these properties the *substitute* algorithm simplifies considerably. Let us assume f is a *BMD in SOV, g is a *BMD not in SOV and the variable x is somewhere in the middle of f. In a first phase lines 2 and 4 are executed continually until x is reached. Line 3 calls a multiplication and an addition. The multiplication returns its result immediately since $high(f)$ is a terminal node. The complexity of the addition depends on the *BMDs f and g. But again, the SOV-structure reduces the number of execution steps, since $f_{\dot{y}}$ is a terminal node for all variables y. Later on in the paper g will be a *BMD of constant size, so we can give a more detailed analysis. Returning from the recursive calls of line 4, line 5 does no recursive calls because of $high(f)$. For the same reason line 7 ends immediately and the *BMD $E4$ in line 8 has a depth of 1.

For these reasons, maintaining the SOV-structure as long as possible reduces the number of execution steps of the *substitute* algorithm. Since substitution is the basic operation in the method of verification by backward construction the overall number of execution steps of the method reduces considerably, too. In addition, the SOV-structure simplifies the analysis of the complexity of the method extremely.

3 The Class of Wallace-Tree like Multipliers

In this Section, we briefly introduce Wallace-tree like multiplier circuits. For a more detailed description of a representative of that class see for example [LV83], [Be87]. It turns out that the suppositions of our statements do not depend much on the structure of the multiplier. The only assumption we need is, that we can divide the circuit into two parts. One part calculates the partial products bits, that is the AND of two input bits.

The other part adds these partial products bits to the final result using (only) fulladder-cells. How these cells are arranged, e.g. as a 3to2 or as a 4to2 reduction [Be87], as a binary tree or as a linear list, has no influence on the proof. (Note, that this is a more general approach than the one considered in [BC95].)

Let $a = a_{n-1} \dots a_0$ and $b = b_{n-1} \dots b_0$ be two n-bit numbers ($n = 2^m$) in binary representation. (If n is not a power of 2 extend n to a power of 2 and set the superfluous inputs to zero.) The multiplication of a and b is then equivalent to summing up n $2n$-bit *partial products* corresponding to the rows of the *partial product matrix* P:

$$P = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & a_{n-1}b_0 & \dots & a_1b_0 & a_0b_0 \\ 0 & 0 & \dots & 0 & a_{n-1}b_1 & a_{n-2}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & \ddots & & & \vdots & \ddots & & \vdots \\ 0 & a_{n-1}b_{n-1} & \dots & & & a_0b_{n-1} & 0 & \dots & 0 \end{pmatrix}$$

4 Complexity of Backward Construction

In this Section we analyze the complexity of the method of verification by backward construction, as introduced in [HMY95]. In Subsection 4.1 we explain the principles of the method. In Subsection 4.2 the backward construction of the *BMDs for the adder part of the circuit is analyzed. In Subsection 4.3 we take the partial product bits into account and discuss the resulting complexity. In the last subsection we put together these parts to get the final overall complexity.

4.1 The Method of Backward Construction

In general, the method of verification by backward construction works as follows: First, to each primary output a distinct variable is assigned. In the next step the *BMD for the output word is constructed by weighting and summing up all *BMDs of these variables according to the given output encoding. (We assume an unsigned integer encoding of the outputs.) Note, that we obtain the SOV-structure for the resulting *BMD. Then, a cut is placed, crossing all primary outputs of the circuit. The cut is moved towards the primary inputs, such that the output lines of each gate move onto the cut according to some reverse topological order of the gates. While moving the cut towards the primary inputs of the circuit, the *BMD is constructed as follows: For the next line of the circuit crossing the cut, which is also an output line of some gate of the circuit, find its corresponding variable in the *BMD constructed so far. Now, substitute this variable by the (output) function of the gate, delete the output line from the cut, and add all input lines of the gate to the cut. Finally, the cut has been moved to the outputs of AND gates, computing the initial partial product bits. The very final step is the substitution of corresponding variables in the obtained *BMD by the initial partial product bits. After this step, we obtain the *BMD representing the multiplication of a and b if the circuit is correct. As mentioned above, the cut is moved according to a reverse topological order of the gates.

An example of a combinational circuit and the directed acyclic (multi)graph representing its structure is given in Figure 4. We have labeled the nodes of the DAG with numbers $1, 2, \dots, 10$ as an example of a reverse topological order. In general, there exists

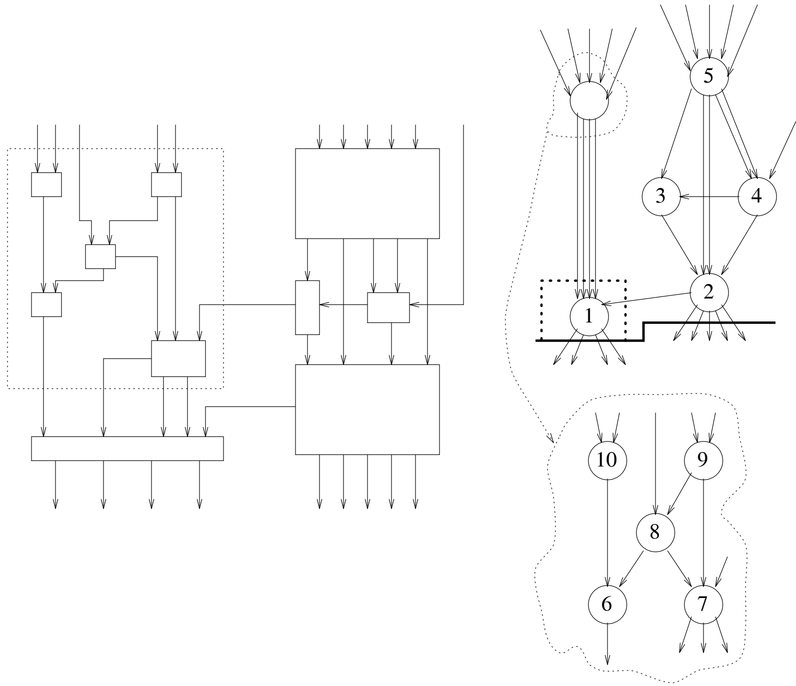

Figure 4: The method of backward construction

more than one such order. As we shall see, the two parts of our proof in Subsection 4.2 and Subsection 4.3 are not based on the knowledge of a specific reverse topological order, as long as we substitute the variables representing the fulladder outputs before the variables representing the initial partial product bits. If this is not the case, we can give one specific class of variable orders, for that our statements hold, nevertheless. This is discussed in Subsection 4.4.

4.2 Constructing the *BMD for the adder part

In this subsection we analyze the costs of constructing the *BMD for the adder-part of the multiplier. As mentioned, we start at the primary outputs of the circuit and move toward the inputs of this part of the circuit. These inputs are the output-lines of AND gates, representing initial partial product bits.

The first Lemma shows, that the substitution of the sum- and carry-output of the same fulladder in a *BMD in SOV maintains the SOV-structure. This means, that SOV is invariant against these substitutions. For that reason, we have to analyze the substitution costs for a single fulladder, only (Lemma 4.2 and Corollary 4.1), to conclude the overall costs (Theorem 4.1 and Theorem 4.2).

Lemma 4.1 *Let F be a *BMD in SOV and $\mathcal{X}$ denote the variable set of F. Let x_i, x_j be two variables in $\mathcal{X}$, which represent the sum- and carry-output of the same fulladder. The inputs to the fulladder are represented by variables $x_k, x_l, x_m \notin \mathcal{X}$. Independent of the order on the variable sets of the *BMDs involved in the substitution process, it holds that substituting x_i, x_j in F by the *BMDs for the functions $sum(x_k, x_l, x_m), carry(x_k, x_l, x_m)$ generates a *BMD F' with variable set $(\mathcal{X} \setminus \{x_i, x_j\}) \cup \{x_k, x_l, x_m\}$ and:*

1. *F' is in SOV*
2. *the terminal value of the high-successors of the nodes for x_k, x_l, x_m in F' is the same as that for the high-successor of the node for x_i in F.*

Proof: The proof is based on a merely functional argument. The function f, that is represented by the *BMD F, which is in SOV, is the sum of the variables in $\mathcal{X}$. These variables are weighted with an integer value. Since the variables x_i, x_j represent the sum- and carry-output of the same fulladder, they have weights w and $2w$, respectively. For the sum- and carry-output we get the following functions:

$$\begin{aligned} sum(x_k, x_l, x_m) &= x_k \oplus x_l \oplus x_m \\ &= x_k + x_l + x_m - 2x_k x_l - 2x_k x_m - 2x_l x_m + 4x_k x_l x_m \\ carry(x_k, x_l, x_m) &= (x_k \wedge x_l) \vee (x_k \wedge x_m) \vee (x_l \wedge x_m) \\ &= x_k x_l + x_k x_m + x_l x_m - 2x_k x_l x_m \end{aligned}$$

by expressing the boolean operations $\oplus$ and $\wedge, \vee$ by integer addition, subtraction and multiplication, i.e.,

$$\begin{aligned} x \oplus y &= x + y - 2xy \\ x \wedge y &= xy \\ x \vee y &= x + y - xy. \end{aligned}$$

The substitution of x_i, x_j in f by $sum(x_k, x_l, x_m), carry(x_k, x_l, x_m)$ yields the following function f'

$$\begin{aligned} f' &= \ldots + w \cdot sum(x_k, x_l, x_m) + 2w \cdot carry(x_k, x_l, x_m) + \ldots \\ &= \ldots + w \cdot x_k + w \cdot x_l + w \cdot x_m + \ldots, \end{aligned}$$

as can easily be verified. The rest of the variables in f are not affected since $x_k, x_l, x_m \notin \mathcal{X}$ and the weighted variables x_i, x_j appear only once in f. Therefore the *BMD F', representing function f', must be in SOV again, independent of the variable order. Furthermore all three high-edges of the nodes for x_k, x_l, x_m in F' point to a terminal node with value w. This completes the proof. ■

With Lemma 4.1 we can be sure, that the SOV-structure of the *BMD is maintained, if we have fulladders as basic cells and if we substitute the variables corresponding to both outputs of a particular fulladder directly one after another. This is independent of the chosen variable order and the chosen reverse topological order of the DAG representing the adder part of the multiplier.

We proceed now with calculating the costs by counting the calls of the *substitute* algorithm.

Lemma 4.2 *Let F be a *BMD in SOV with size $|F|$. Let $\mathcal{X}, x_i, x_j, x_k, x_l, x_m, sum(x_k, x_l, x_m)$ and $carry(x_k, x_l, x_m)$ be defined as in Lemma 4.1.*
Substituting x_i, x_j in F by $sum(x_k, x_l, x_m)$ and $carry(x_k, x_l, x_m)$ is bounded by $\mathcal{O}(|F|)$ with respect to time, independent of the variable order or order of substitutions.

Proof: We denote the *BMDs for $sum(x_k, x_l, x_m)$ and $carry(x_k, x_l, x_m)$ by Su and Ca. We consider first the substitution of x_i by Su. The substitution process can be visualized in Figure 5. (Note that the order between x_k, x_l, x_m does not matter because the sum- and carry-function are totally symmetric. Therefore, the nonterminal node labels are omitted.) The *subst*-algorithm calls itself recursively until it reaches the node in F labeled with variable x_i. Obviously, the number of recursive *subst*-calls is bounded by $\mathcal{O}(|F|)$. At the node labeled with x_i, the operation $F_{low(x_i)} + Su \cdot F_{high(x_i)}$ has to be

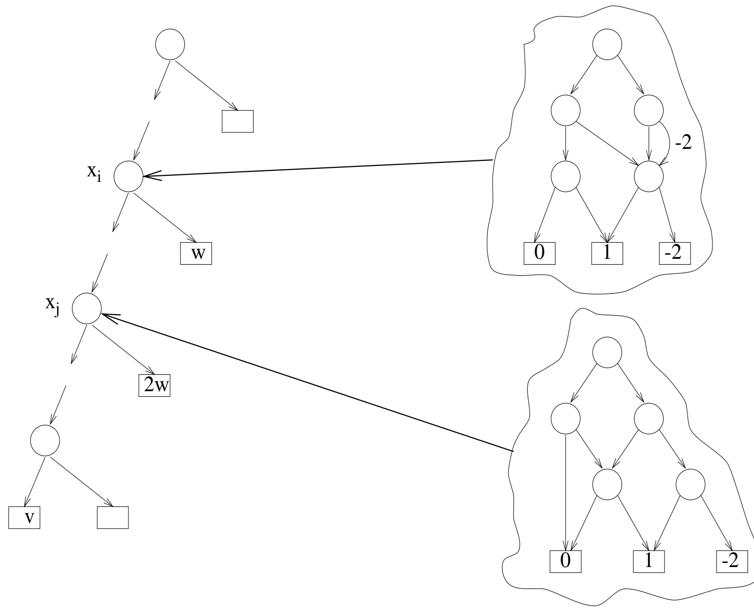

Figure 5: Representation of substituting x_i by Su and x_j by Ca.

carried out, where $F_{low(x_i)}, F_{high(x_i)}$ denote the *BMDs to which the low- and high-edge of node x_i point. Since F is in SOV, $F_{high(x_i)}$ is a terminal node and the call to *mult* ends immediately. Since $F_{low(x_i)}$ is in SOV, too, and Su is of constant size, the addition is bounded by $\mathcal{O}(|F|)$. For all other recursive *subst*-calls, there will be only a constant number of *mult*- and *add*-calls because of the SOV-structure, as can easily be verified. We get an overall bound of $\mathcal{O}(|F|)$ for the substitution of x_i by Su. The *BMD after this substitution can be seen in Figure 6 for a variable order $x_k < \ldots < x_l < \ldots < x_m$.

For the substitution of x_j by Ca, analogous arguments hold, except for the fact, that the SOV-structure of F is destroyed by Su (see Figure 6). First of all, this leads to some additional recursive *subst*-calls. But this will be only a constant number, because Su has constant depth. Furthermore, we get some additional calls to *add* and *mult* during some of the recursive *subst*-calls, when the nodes labeled with variables x_k, x_l, x_m meet each other. But this number is bounded by a constant value too, since Su and Ca have constant depth.

Therefore, we get a time bound of $\mathcal{O}(|F|)$ for the substitution of x_i by Su and x_j by Ca independent of the variable order on $\mathcal{X}$ and $(\mathcal{X} \setminus \{x_i, x_j\}) \cup \{x_k, x_l, x_m\}$.

The proof for first substituting x_j by Ca and then x_i by Su is analogous. ■

From Lemma 4.1 and Lemma 4.2 we get immediately the following Corollary:

Corollary 4.1 *Substitution of x_i, x_j in a *BMD F in SOV by the *BMDs Su and Ca according to Lemma 4.2 is bounded by $\mathcal{O}(|F|)$ with respect to space.*

Corollary 4.1 leads directly to the space bounds for the construction of the *BMD for the adder part.

Theorem 4.1 *Constructing the *BMD for the adder part of the multiplication circuit using substitution is bounded by $\mathcal{O}(n^2)$ with respect to space. This is independent of the chosen reverse topological order for the fulladder-cells.*

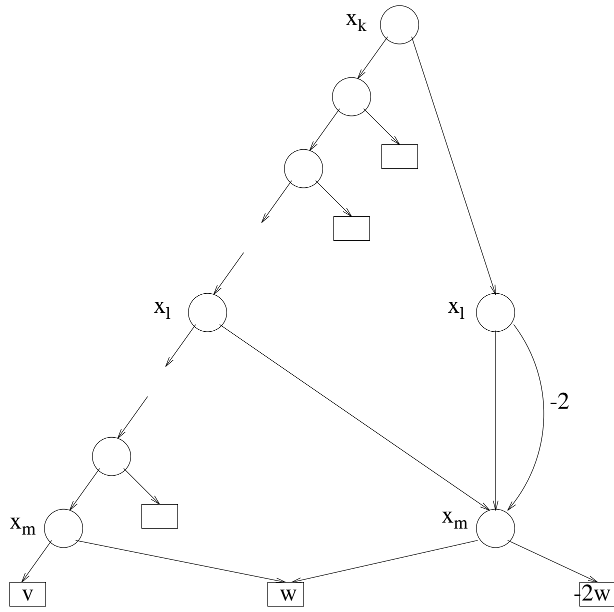

Figure 6: *BMD after substitution of variable x_i only.

Proof: The *BMD F_0 with which we start has size $\mathcal{O}(n)$ as follows: There is one nonterminal node for each variable representing a primary output. The resulting *BMD constructed for the adder-part has size $\mathcal{O}(n^2)$: One nonterminal node for each variable representing an initial partial product bit. The exact size depends on the chosen realization of the multiplier.

Substitution of the variables corresponding to the outputs of a fulladder by the *BMDs for the input functions increases the size of the *BMD by 1. Each such substitution is bounded in space by the size of the *BMD, in which the substitution is performed. Therefore, the largest amount of space is needed for the substitution of the very last fulladder.

If we consider the fact, that we can delete a *BMD F after substitution of the variables x_i, x_j by the *BMDs for the fulladder output functions, we get an overall bound of $\mathcal{O}(n^2)$ with respect to space, independent of the chosen substitution order. ■

Additionally, at each point of the substitution process, we *know exactly* the size of the *BMD: After having processed the i. fulladder, the size is $|F_0| + i$.

After analyzing the space requirements for the substitutions, we now consider the time requirements.

Theorem 4.2 *Constructing the *BMD for the adder-part using substitution is bounded by $\mathcal{O}(n^4)$ with respect to time, independent of the chosen reverse topological order for the fulladder-cells.*

Proof: For the proof we first count the number of fulladder elements. Depending on the realization, the exact number of these elements differs. Asymptotically there are $m = \mathcal{O}(n^2)$ fulladder cells, forming the adder-part of any meaningful multiplier. Therefore the number of execution steps has an upper bound of

$$\sum_{i=0}^{m-1} (|F_0| + i)$$

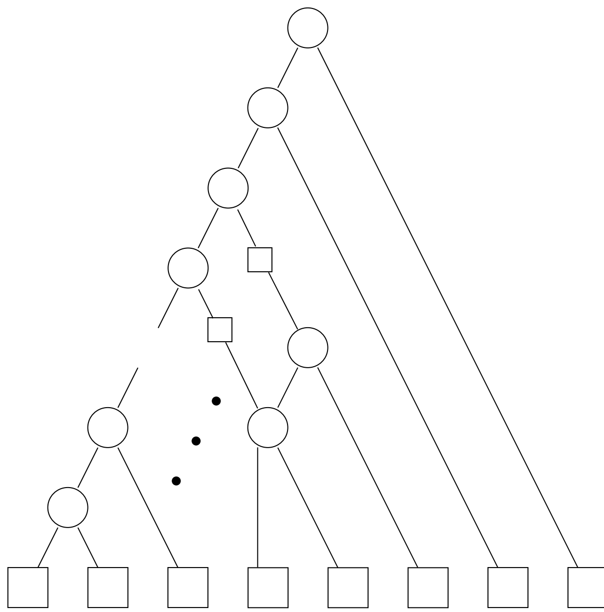

Figure 7: *BMD after substituting some initial partial product bits.

where $|F_0|$ is the size of the initial *BMD according to the proof of Theorem 4.1. The above sum is figured out as follows:

$$\begin{aligned}\sum_{i=0}^{m-1}(|F_0|+i) &= m\cdot|F_0|+\sum_{i=0}^{m-1} i\\ &= m\cdot|F_0|+\frac{m(m-1)}{2}\\ &= \mathcal{O}(n^4)\end{aligned}$$

with $|F_0| = \mathcal{O}(n)$ and $m = \mathcal{O}(n^2)$. ■

4.3 Substitution of the Initial Partial Products

Up to this point we analyzed the complexity of the method of verification by backward construction for the part of the multiplier circuit that adds the partial product bits to obtain the result of the multiplication. In the sequence we analyze the costs of the final step, substituting the initial partial product bits into the *BMD. By doing so, we will destroy the SOV-structure of the *BMD. Our starting point is the *BMD constructed up to the outputs of the AND gates. It has size $m' = \mathcal{O}(n^2)$, since there are n^2 partial product bits. In fact, it holds $m' = |F_0| + m = |F_0| + \#FA$, with m from the proof of Theorem 4.2 and with $\#FA$ denoting the number of fulladder cells.

We assume in the following a fixed order among the a- and b-word, assigned to the primary inputs of the circuit. Let the variable order be $a_{n-1} < \ldots < a_0 < b_{n-1} < \ldots < b_0$. The reason for this unique variable order is, that we must compare the constructed *BMD with the one for the specification. These two must have the same variable order. In fact, the variable order within the a_i and within the b_j is of no interest as long as *all* variables forming one input word are before the variables forming the second input word. Otherwise the final *BMD may be larger than $2n$ [BC94].

We define a *low-path* of a *BMD F as the path in F from the root to a leaf, consisting of only low-edges. Note, that there is only one such path in a *BMD.

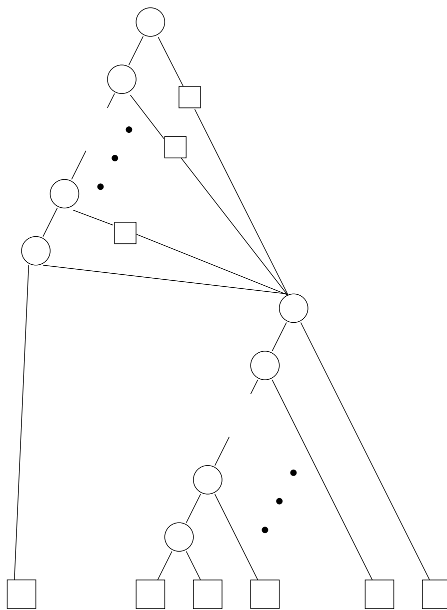

Figure 8: Final *BMD for the multiplication.

We now show, that the intermediate *BMDs have a structure like that in Figure 7. (The small box on an edge denotes the multiplicative factor.) All nodes with a terminal high-successor in the upper diagonal line, the low-path, are marked with a variable x_k, assigned to a fulladder input line. These lines are also output lines of AND gates. Nodes with a nonterminal high-successor are labeled with a variable a_i (also in the low-path). The high-successors of them are labeled with a variable b_j. These are located in the lower diagonal line.

The final *BMD is structured as shown in Figure 8.

Theorem 4.3 *Let F be the *BMD constructed for the adder part. Substitution of the variables of F by the *BMDs for the initial partial product bits $a_i \cdot b_j$ is bounded by $\mathcal{O}(n^2)$ with respect to space and $\mathcal{O}(n^4)$ with respect to time independent of the variable order in F and independent of the chosen reverse topological order for the AND gates.*

Proof: Let F' denote an intermediate *BMD generated after the substitution of some initial partial product bits. If the node labeled with variable x has not yet been substituted, it must be on the low-path of F'. Now we consider the substitution of node x by the *BMD for an initial partial product bit $a_i \cdot b_j$, denoted $*BMD(a_i \cdot b_j)$. Obviously, the number of recursive calls to the substitute algorithm (Figure 2) is bounded by $\mathcal{O}(|F'|)$. During the last of the recursive *subst* calls, i.e., at node x, the following operations have to be carried out:

$$F_{low(x)} + *BMD(a_i \cdot b_j) \cdot F_{high(x)},$$

The call to *mult* ends immediately, since x has a terminal high-successor because of the SOV-structure of F at the beginning of the substitution process. For the *add* calls we have to distinguish two different cases.

1. No nodes have been substituted by initial partial products bits with variable a_i until now. Since node x (1 nonterminal node) has to be substituted by $*BMD(a_i \cdot b_j)$ (2 nonterminal nodes) the size of F' increases by 1 if no node for variable b_j can be shared, and it remains unchanged otherwise.

If variable a_i comes after the predecessor of variable x in the variable order, $*BMD(a_i \cdot b_j)$ reaches its final position in *BMD F' by calls to *add* during the final substitute call, i.e.line 3 of Figure 2. The number of these calls is obviously bounded by $\mathcal{O}(|F'|)$.

If variable a_i comes before the predecessor of variable x in the variable order, a_i reaches its final position in *BMD F' by calls to *add* during resolving previous substitute calls (line 8 of Figure 2). The number of these *add* calls is constant, as one can easily make sure. Furthermore, there is one call to *MakeNode* and one call to *mult* for each of the previous *subst* calls (lines 6 and 7 of Figure 2).

2. There exists already a node for variable a_i, i.e., some nodes of F have already been substituted by initial partial product bits $a_i \cdot b_{i_1}, \ldots, a_i \cdot b_{i_k}$ so far. If there exists also a node for b_j, e.g. created during the substitution of an initial partial product bit $a_l \cdot b_j$, the size of F' during the substitution may decreases by 1, if the node b_j can be shared, or remains unchanged, otherwise.

   The only difference to the first case is, that there are an additional number of at most n *add* calls, because of the position of b_j in the variable order among the $b_{i_1}, \ldots, b_{i_k}$ (the worst case occurs, if $b_j = b_0$ and $\{b_{i_1}, \ldots, b_{i_k}\} = \{b_{n-1}, \ldots, b_1\}$). So we have a bound on the total number of calls of $\mathcal{O}(|F'| + n) = \mathcal{O}(|F'|)$, because $|F'| = \Omega(n)$.

Cases 1 and 2 together give a bound on the total number of algorithm calls of $\mathcal{O}(|F'|)$ for the substitution of node x by $*BMD(a_i \cdot b_j)$ in F'. Since we have at most an increase of n in the size of the starting BMD F (the first n initial partial product bits of the substitution process all have different a-variables) and the size of F is $\mathcal{O}(n^2)$, the size of the intermediate *BMDs during substitution is bounded by $\mathcal{O}(n^2)$ too. This proofs the first part of the theorem. Furthermore, since the number of substitutions is bounded by $\mathcal{O}(n^2)$ and the size of the starting *BMD F is $\mathcal{O}(n^2)$ we get an overall time bound of

$$\sum_{i=1}^{m'} \mathcal{O}(n^2) = \mathcal{O}(n^4).$$

This proves the second part of the theorem and we have completed the proof. ■

4.4 Complexity of Backward Construction

With Theorema 4.1, 4.2 and 4.3, we conclude, that the method of backward construction applied to the class of Wallace-tree like multipliers is bounded by $\mathcal{O}(n^2)$ with respect to space and by $\mathcal{O}(n^4)$ with respect to time. These bounds do (largely) not depend on the chosen variable order during the single substitution steps. Additionally, these bounds do not depend on the order of substitutions as long as we first substitute *all* fulladder-cells and afterwards the initial partial product bits.

If we allow one further restriction on the variable ordering, we attain to be totally free in the substitution ordering across both circuit parts. This means, there is a class of variable orders, with that we can substitute an initial partial product bit before having processed all fulladder cells. Nevertheless, the complexity remains polynomial. We only have to regard the reverse topological order over the whole multiplier circuit.

We define an (x, a, b) variable order to be any variable order, which has as a first block all x-variables, as a second block all a-variables and as a third block all b-variables. The x-variables denote the fulladder in- and outputs and the a- and b-variables the variables of the input words to the multiplier. Then we can give the following theorem:

Theorem 4.4 *Given an (x, a, b) variable order, the method of backward construction, applied to the class of Wallace-tree like multipliers, is bounded by $\mathcal{O}(n^2)$ with respect to space and $\mathcal{O}(n^4)$ with respect to time independent of the reverse topological order on the circuit.*

Proof: The *BMD decomposes into two parts. The upper part consists of x-variables only, and is in SOV. (Despite the fact, that the last high-edge points to a nonterminal node, labeled with an a-variable.) The lower part consists of a- and b-variables. We have to consider two cases:

1. Substituting Su (Ca), we first have to find the substituted variable. That will be found in the upper part of the *BMD. Depending on the variable ordering within the x_k, the *add* operation of the substitution continues downward the *BMD. Maximally, it reaches the edge, pointing to a_0 (respectively any a_i, which ever is the 'smallest' a-variable so far). There, the recursive calls terminate, since $x_k < a_i$, for all k, i. Theorema 4.1, 4.2 can be applied for the costs, considering only the size of the upper part of the *BMD.

2. Substituting a partial product, Theorem 4.3 can be applied.

■

One open problem is the complexity if we allow, that initial partial product bits are substituted before *all* fulladder-cells are processed, but with a different variable order as that from Theorem 4.4, i.e., not all x-variables are at the beginning of the variable order. We expect, that the complexity remains still polynomial, since the structure of the resulting *BMD is similar to the one used here. The crucial point is the SOV-structure. There we have a chain of x-variables together with a-variables and one chain of b-variables (see Figure 7). The b-chain forms a SOV-structure, the other one has similar properties. All x-variables have a terminal node as high-successor like in a regular SOV-structure. Only the high-successors of a-variables point to b-variables.

Before we come to our conclusions, we briefly discuss the case, that the multiplier circuit to be verified is *not* a multiplier circuit, e.g. it has a design error. With our results, we can *prematurely* detect such an error by watching the *BMD size.

Assume, we have processed only gates in the adder-part of the circuit. After processing the i. gate the *BMD has size $|F_0| + i$. If the actual *BMD has not that size, there is an error. Assume now, that we considered at least some initial partial product bits of the circuit. Then, the size of the *BMD depends on which initial partial product bits have been substituted. Not considering an accurate size bound, the *BMD must not be larger than $|F_0| + \#FA + n$, and must not be smaller that $2n$, at any time.

5 Conclusions and Future Research

In this paper we analyzed the complexity of the method of verification by backward construction applied to the class of Wallace-tree like multipliers. We gave a formal proof of polynomial upper bounds on run-time and space requirements with respect to the input word sizes for that method. Note, that until now, only experimental data has been given to show the feasibility of the method.

Future research directions are to find a more efficient way for backward construction, e.g., by parallelizing the substitution process according to the given hierarchical circuit structure. Furthermore, we will take a look at integer dividers, which could not be verified by backward construction so far.

References

[ABC+94] A. Aziz, F. Balarin, S.-T. Cheng, R. Hojati, T. Kam, S.C. Krishnan, R.K. Ranjan, T.R. Shiple, V. Singhal, S. Tasiran, H.-Y. Wang, R.K. Brayton, A.L. Sangiovanni-Vincentelli, *HSIS: A BDD-Based Environment for Formal Verification.* 31nd Design Automation Conference, pp. 454-459, 1994.

[BC94] R.E. Bryant, Y. Chen, *Verification of Arithmetic Functions with Binary Moment Diagrams.* Technical report CMU-CS-94-160,Carnegie Mellon University, 1994.

[BC95] R.E. Bryant, Y. Chen, *Verification of Arithmetic Circuits with Binary Moment Diagrams.* 32nd Design Automation Conference, pp. 535-541, 1995.

[BCM90] J.R. Burch, E.M. Clarke K.L. McMillan D.L. Dill, *Sequential circuit verification using symbolic model checking.* 27th Design Automation Conference, pp. 46-51, 1990.

[Be87] B. Becker, *An Easily Testable Optimal-Time VLSI-Multiplier.* Acta Informatica 24, pp. 363-380, 1987.

[Bry86] R.E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Trans. on Computers, Vol. C-35, No. 8, August 1986.

[Bry91] R.E. Bryant, *On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication.* IEEE Trans. on Computers, Vol. 40, pp. 205-213, 1991.

[Bry92] R.E. Bryant *Symbolic boolean manipulation with ordered binary decision diagrams.* ACM, Comp. Surveys, 24:293-318, 1992

[CCH+96] Y.-A. Chen, E.M. Clarke, P.-H. Ho, Y. Hoskote, T. Kam, M. Khaira, J. O'Leary, X. Zaho, *Verification of all circuits in a floating-point unit using word-level model checking.* Int'l Conf. on Formal Methods in CAD, 1996.

[CB96] Y.-A. Chen, R.E. Bryant, *Efficient Branch and Bound Search with Application to Computer Aided Design.* Kluwer Academic Publisher, 1996.

[CFZ95] , E.M. Clarke, M. Fujita, X. Zhao, *Hybrid Decision Diagrams - Overcoming the Limitations of MTBDDs and BMDs.* Int'l Conf. on CAD, pp. 159-163, 1995.

[DBR96] R. Drechsler, B. Becker, S. Ruppertz, *K*BMDs: A New Data Structure for Verification.* European Design & Test Conf., pp 2-8, 1996.

[End95] R. Enders, *Note on the Complexity of Binary Moment Diagram Representations.* IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp. 191-197, 1995

[FFK88] M. Fujita, H. Fujisawa, N. Kawato, *Evaluation and improvements of boolean comparison method based on binary decision diagrams.* International Conference on CAD, pp. 2-5, 1988.

[HMY95] K. Hamaguchi, A. Morita, S. Yajma, *Efficient Construction of Binary Moment Diagrams for Verifying Arithmetic Circuits.* International Conference on CAD, 1995.

[LV83] W. K. Luk, J. Vuillemin, *Recursive Implementation of Optimal Time VLSI Integer Multipliers.* VLSI'83, F. Anceau,E. J. Aas (Eds.),pp. 155-168, North Holland: Elsevier 1983.

[MWBS88] , S. Malik, A.R. Wang, R.K. Brayton, A.L. Sangiovanni-Vincentelli, *Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment.* Int'l Conf. on CAD, pp. 6-9, 1988.

[Wal64] C.S. Wallace, *A suggestion for a fast multiplier.* IEEE 13, pp 14-17, 1964.