Vérification automatique des multiplicateurs

Pierre Senellart

Février 2002 – Juin 2002

The purpose of my internship at Chalmers University of Technology was to study the possible methods for checking the correctness of a multiplier, that is an electronic circuit which computes the multiplication of two integer numbers.

1 Related works

Classical methods for automatic verification of circuits, such as the famous Binary Decision Diagrams introduced in [Bry86] are very inefficient for verifying multipliers. Bryant proved in [Bry86] that whatever the chosen ordering of the variables is, there exists an output of the multiplier whose BDD representation has an exponential size, which is unsuitable for common-size (e.g. 32 or 64 bits) multipliers. Several techniques have been proposed to deal with this issue. Kapur et al. showed in [DM96] how to prove the correctness of a large family of common multipliers but their method heavily relies on hand-made lemmas, whose automatically generation is only speculated. [SB98], [Sta99] and [CC01] proposed different methods based on an inductive approach to the problem.

However, the most efficient approach for the time being seems to be the Multiplicative Binary Moment Diagrams (*BMD) described in [BC95]. The first approach was a component-level one, which required high level information, but Hamaguchi and al. suggested in [HMY95] a backward sweeping method that allows to build the *BMD representation of a circuit in a completely automatic way. The experimental results show an $O(n^{3.5})$ asymptotic behaviour and Keim et al. proved in [KMB$^+$97] an asymptotic $O(n^4)$ bound for Wallace-tree-like multipliers.

2 *BMDs

2.1 BDDs

*BMDs are an extension to well-known (Ordered) Binary Decision Diagrams. An BDD is a directed acyclic graph (DAG) which represents in unique way a function of booleans variables to a boolean value, if an initial total ordering of the variables is assumed. The function value is computed by following a path in the DAG, according to the values of the variables. Figure 1, for instance, reprensents the carry of a full adder: continuous lines from a node must be followed if the corresponding variable is true, dotted lines must be followed otherwise. The DAG structure often gives a concise representation of the boolean function, which can be computed in linear time in the size of the BDD (these results are proved in [Bry86]). Verifying a circuit is then as simple as computing the BDD of the boolean function performed by the circuit and comparing it to the BDD of the specification function. This technique works for a large class of circuits, in particular arithmetic adders, since the size of their BDDs is linear in the number of variables. Unfortunately, Bryant also proved that for any ordering of the variables, the BDD representation of some multiplier output will have a graph of exponential size: BDDs are thus inefficient on multipliers.

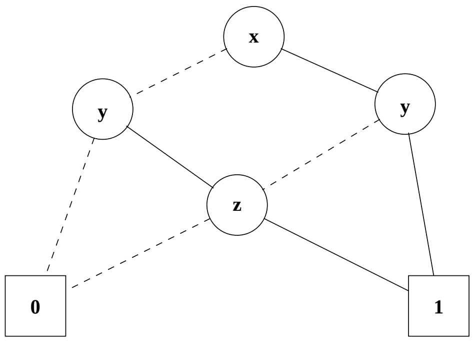
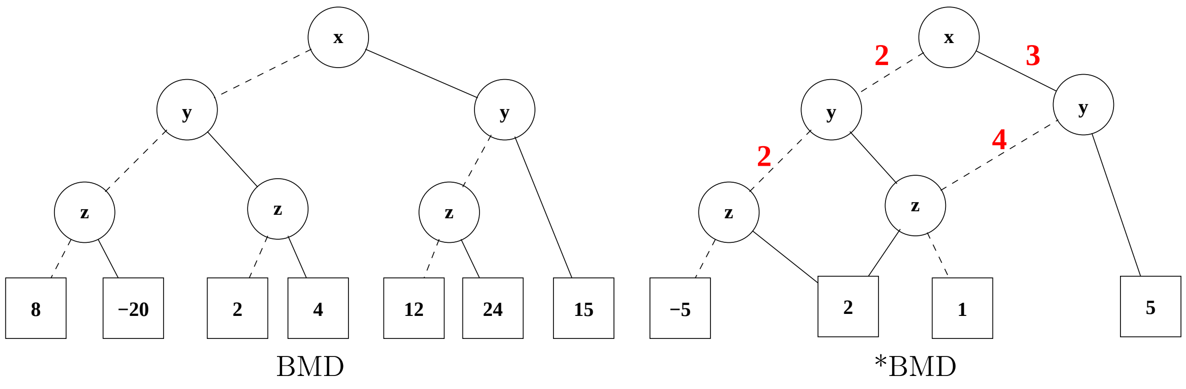


Figure 1: Binary Decision Diagram for the carry of a full adder



$$f(x, y, z) = 8 - 20z + 2y + 4yz + 12x + 24xz + 15xy$$

Figure 2: BMD and *BMD of a function from booleans to integers

2.2 BMDs and *BMDs

Let $f : \{0, 1\}^n \to \mathbb{N}$ be a function from booleans to integers. Let $f_{\bar{x}_i}$ and f_{x_i} be defined as follows:

$$\begin{array}{llll} f_{\bar{x}_i}: & \{0,1\}^{n-1} & \to & \mathbb{N} \\ & (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) & \mapsto & f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \end{array}$$

$$\begin{array}{llll} f_{x_i}: & \{0,1\}^{n-1} & \to & \mathbb{N} \\ & (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) & \mapsto & f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \end{array}$$

If $f_{\dot{x}_i} = f_{x_i} - f_{\bar{x}_i}$, we have a decomposition of the function f:

$$f(x_1, \ldots, x_n) = \underbrace{f_{\bar{x}_i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)}_{\text{constant moment}} + x_i \underbrace{f_{\dot{x}_i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)}_{\text{linear moment}}$$

Binary Moment Diagrams (BMDs) are directed acyclic graphs based on this decomposition, instead of the more direct decomposition (known as the Shannon decomposition) BDDs use. The difference between BMDs and *BMDs (*Multiplicative* Binary Moment Diagrams) is that edges of the DAG may be weighted with a value, which is a multiplying factor to be applied to the value of the function. An example of BMD and *BMD is given in Figure 2.

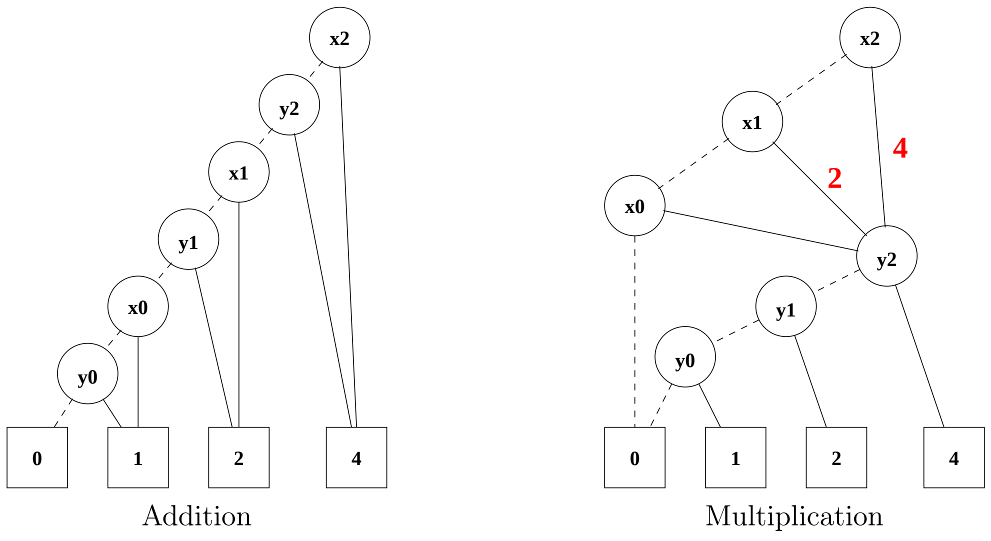


Figure 3: *BMDs of classical arithmetic operations

The important difference between BDDs and *BMDs is that *BMDs represent function from booleans to *integers* whereas BDDs represent functions from booleans to *booleans*. *BMDs work as a higher level than BDDs. A consequence is that *BMDs for classical arithmetic operations, included multiplication, are of linear size in the number of variables (cf [BC95]), as illustrated on Figure 3.

Because BMDs are at a higher level than BDDs, Bryant's original algorithm needed high-level knowledge of the circuit: components of the circuits were both described at bit level and word (i.e. integer) level, each component was checked at bit level against their word level interpretaion, and the composition of the components was checked as word level against the word level specification of the circuit. The direct construction of a *BMD of the entire circuit was impossible due to the size of intermediate results.

3 Backward construction algorithm

The backward construction algorithm, proposed by Hamaguchi et al in [HMY95], is a method for building the *BMD of a circuit which does not need high-level information and which is more efficient than the direct construction. The idea is to move a cut from the outputs of the circuits to the inputs. At the beginning of the algorithm, the cut crosses all the primary outputs. The *BMD of the word level interpretation of the output is built. At each step, a gate just left to the cut is chosen (with the condition that a gate may be chosen only if its ouput is only connected to input of gates that have already been taken) and its output is substituted in the *BMD by the corresponding function of its inputs. At any time, the *BMD expresses the word level representation of the output as a function of the nets currently crossed by the cut. At the end of the algorithm, the cut crosses all primary inputs. The *BMD expresses the word level representation of the output as a function of the inputs. An illustration of this process is given on a sample in Figure 4.

Experimental results show good performances of the backward construction algorithm on multipliers, with an apparent complexity of $O(n^3.5)$ (where n is the number of input bits) for a number of multipliers. Keim et al. proved in [KMB$^+$97] a bound of $O(n^4)$ for a large class of multipliers, known as Wallace-tree like multipliers.

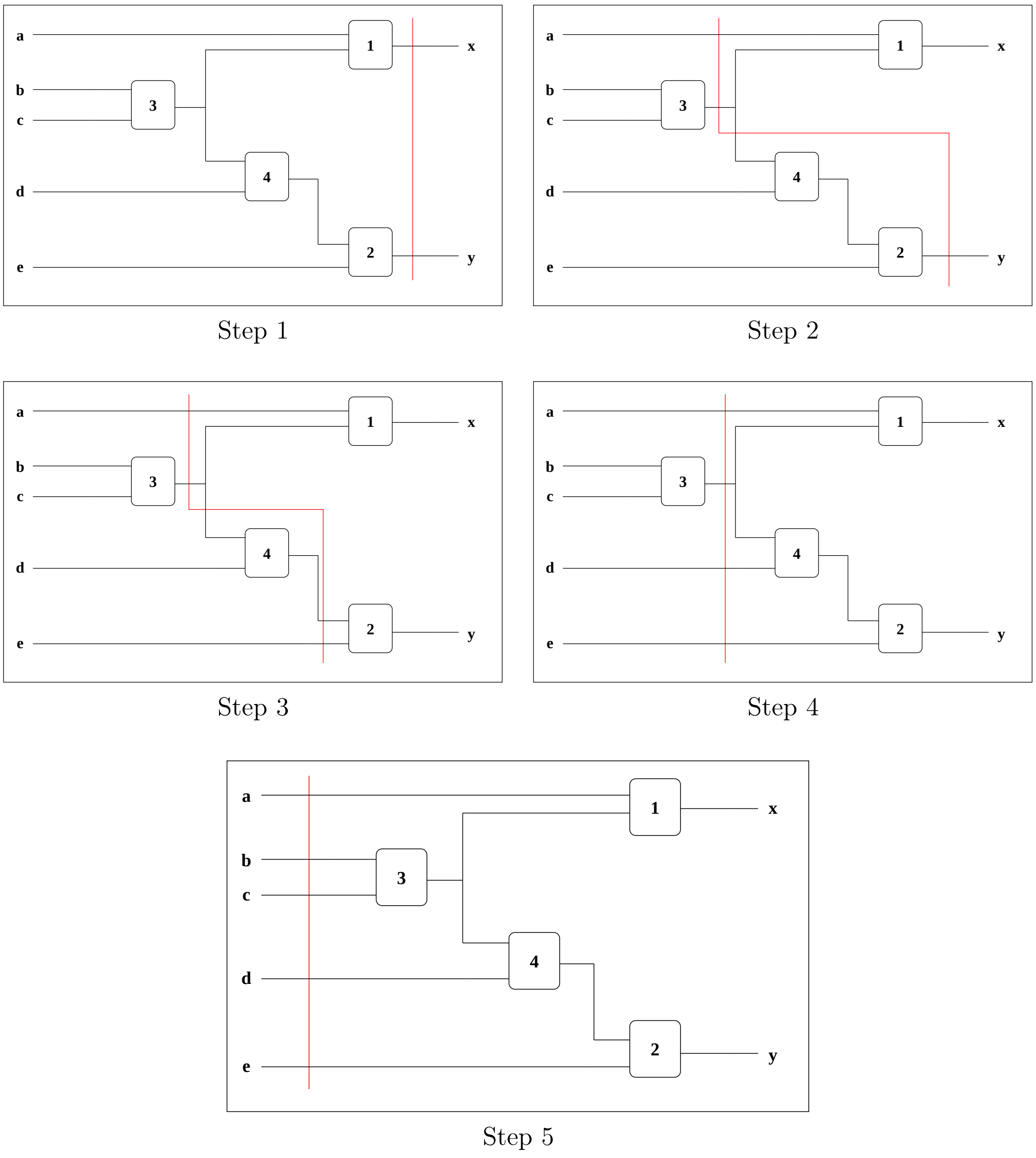


Figure 4: Example of use of the backward construction algorithm

Number of bits	Time Add-step (s)	Time Carry-save (s)
4	1	3
8	12	58
16	161	1115
32	2083	
	$O(n^{3.7})$	$O(n^{4.3})$

Table 1: Execution times for checking classical multipliers with the backward construction algorithm

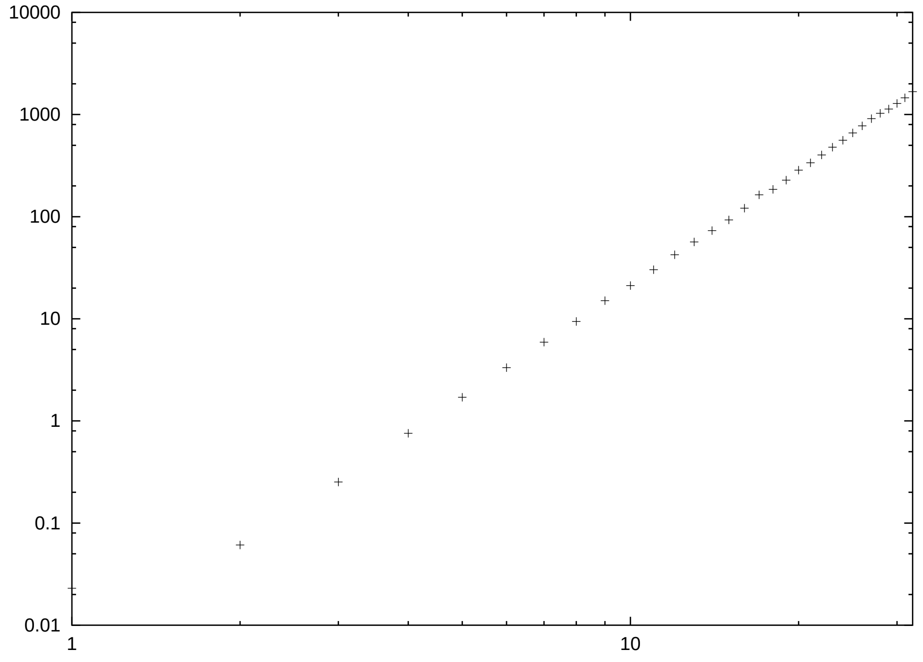


Figure 5: Execution times for checking a regular parallel multiplier with the backward sweeping method

4 Implementation

I implemented *BMDs and the backward construction algorithm in Lava [Cla01], which is a functional language for describing circuits embedded in Haskell. For this purpose, I used Chen's BXD package in C available at `http://www-2.cs.cmu.edu/afs/cs/usr/yachen/www/bxd.html` and Haskell Foreign Function Interface. Experimental time results for two very classical multipliers are given in Table 1; they reproduce more or less the behaviour stated by Hamaguchi et al. (the complexity was likely increased by the use of Haskell garbage collector).

My implementation of the backward construction algorithm was also applied on a Lava specification of a regular parallel multiplier [HE02], which showed the correctness of this multiplier for sizes from 1 to 32 bits, as well as 64 and 92 bits. A log-log chart of the times is given on Figure 5, which show a complexity of about $O(n^3.8)$.

References

[BC95] Randal E. Bryant and Yirng-An Chen. Verification of arithmetic circuits with binary moment diagrams. In *Design Automation Conference*, pages 535–541, 1995.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[CC01] Ying-Tsai Chang and Kwang-Ting Cheng. Induction-based gate-level verification of multipliers. In *Proc. Int'l Conf. on CAD*, 2001.

[Cla01] Koen Claessen. *Embedded Languages for Describing and Verifying Hardware.* PhD thesis, Chalmers University of Technology, Göteborg, 2001.

[DM96] D. Kapur and M. Subramaniam. Mechanically verifying a family of multiplier circuits. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 135–146, New Brunswick, NJ, USA, 1996. Springer Verlag.

[HE02] Per Larsson-Edefors William P. Marnane Henrik Eriksson. A regular parallel multiplier which utilizes multiple carry-propagate adders, 2002.

[HMY95] K. Hamaguchi, A. Morita, and S. Yajima. Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *Proc. Int'l Conf. on CAD*, pages 78–82, 1995.

[KMB$^+$97] Martin Keim, Michael Martin, Bernd Becker, Rolf Drechsler, and Paul Molitor. Polynomial formal verification of multipliers. In *VLSI Test Symposium*, Monterey, USA, 1997.

[SB98] Mary Sheeran and Arne Borälv. How to prove properties of recursively defined circuits using Stålmarck's method. In Mary Sheeran and Bernhard Möller, editors, *Informal Proceedings Workshop on Formal Techniques for Hardware and Hardware-Like Systems, FTH'98, Marstrand, Sweden, 19 June 1998.* Dept. of Computing Science, Chalmers Univ. of Technology and Univ. of Göteborg, 1998.

[Sta99] Ted Stanion. Implicit verification of structurally dissimilar arithmetic circuits. *Proceedings of the 1999 IEEE International Conference on Computer Design (ICCD)*, pages 46–50, 1999.