CDR: A Rewriting Based Tool to Design FPLA Circuits

Zahir Maazouzi, Nirina Andrianarivelo, Wadoud Bousdira, and Jacques Chabin

Laboratoire d'Informatique Fondamentale d'Orléans 45067 Orléans Cedex 02 (Fr.)
{maazouzi, andria, bousdira, chabin}@lifo.univ-orleans.fr

Abstract. A rewriting based method to design circuits on FPLA electronic devices is presented. It is an improvement of our previous work. In comparison with this latter, the number of boolean vectors generated during the design process is reduced. This is done thanks to new forms of rewriting rules denoting new interesting properties on boolean vectors, associated to boolean products. Only boolean products which are implicants of the circuit to design are computed. Thus, this new design process is more efficient than the previous one.

1 Introduction

FPLA (Field Programmable Logic Array) devices are the core of complex circuits as random logic circuits, interface logic, and other applications that require decoding of device inputs. The aim of this work is to design logical circuits for such kind of devices.

It is the first rewriting based method dealing with designing circuits on FPLA chips [1]. It is a correct and complete method, in the sense that, if solutions of the design exist, they will be deduced by our method within a finite period of time. And if no solution exists, then within a finite period of time, a signal of failure is also displayed. Let us note that theoretically, neither simulation nor validation of the computed solutions is needed. Thanks to the formalism of constraint we used, all the properties to be satisfied by the solution are specified in the constraints of the rewrite rules, and they have to be considered in all the steps of the process. Such properties could be conditions on the layout of the logical gates in the device, for example. This approach has been implemented in our prototype CDR (Circuit Design by Rewriting).

New properties of boolean vectors representing boolean products are stated. To the best of our knowledge, we are the first to present all those properties. They will be used to redefine our new system of constrained rewriting rules.

2 Boolean Vectors

We deal with basic notions on boolean algebra [4, 9, 8, 5]. The novelty of this approach consists of handling the boolean functions in the sum of products normal form by a vectorial representation, unlike the classical ones where an

J.A. Campbell and E. Roanes-Lozano (Eds.): AISC 2000, LNAI 1930, pp. 219–222, 2001.

algebraic representations is used. With this representation, we prove interesting new properties that will be used to perform the design. It is independent of any design method, which can be used to improve the classical ones.

A boolean vector $\overrightarrow{v} = (v_1, v_2, \ldots, v_n)$ is any n-uple of values all in the set $\{0,\ 1\}$. Let $\overrightarrow{\mathbb{E}}^n$ be the set of all boolean vectors of size 2^n. For example, $\overrightarrow{\mathbb{E}}^2$ ={(0000), (0001), (0010), (0011), (0100), (0101), (0110), (0111), (1000), (1001), (1010), (1011), (1100), (1101), (1110), (1111)}. n is the dimension of a vector of $\overrightarrow{\mathbb{E}}^n$, and 2^n its size. Let us now define the following notions in order to introduce these new properties :

- the split operation : $\forall n \geq 2$

$$Split : \begin{cases} \overrightarrow{\mathbb{E}}^n \rightarrow \overrightarrow{\mathbb{E}}^2 \times \overrightarrow{\mathbb{E}}^2 \times \ldots \times \overrightarrow{\mathbb{E}}^2 \\ \overrightarrow{v} \quad \mapsto ((v_1, v_2, v_3, v_4), \ldots, (v_{2^n-3}, v_{2^n-2}, v_{2^n-1}, v_{2^n})) \end{cases}$$

- Let $\overrightarrow{v} = (v_1, \ldots, v_{2^n})$ and $\overrightarrow{u} = (u_1, \ldots, u_{2^n})$ be two vectors in $\overrightarrow{\mathbb{E}}^n$:

$$\overrightarrow{v} \preceq \overrightarrow{u} \text{ iff } \forall i \in [1 \ldots 2^n],\ \text{if } u_i = 0 \text{ then } v_i = 0.$$

  Let us note that in the literature[8, 5], $\preceq$ is denoted *implication*. When $\overrightarrow{v} \preceq \overrightarrow{u}$, we say that $\overrightarrow{v}$ *implies* $\overrightarrow{u}$, we say also that $\overrightarrow{v}$ is an *implicant* of $\overrightarrow{u}$.

In the same way, we note $\overrightarrow{\mathbb{P}}^n$ all the vectors of $\overrightarrow{\mathbb{E}}^n$ corresponding to products. We call them *product vectors* of dimension n. In order to distinguish between the two notions : vectorial-syntactic, we represent a syntactic form with a pattern p without arrow, and its vectorial value by $\overrightarrow{p}$.

2.1 Recursive Properties on Products

The following theorem is an interesting result on the boolean algebra in the sense that it is independent of any design method. It makes up the basis of our method.

Theorem 1. *Let be* $\overrightarrow{p} \in \overrightarrow{\mathbb{P}}^n$ *such that* $Split(\overrightarrow{p}) = (\overrightarrow{b}_1, \cdots, \overrightarrow{b}_k)$.

1. $\forall i, j \leq k$, *if* $\overrightarrow{b}_i \neq (0000)$ *and* $\overrightarrow{b}_j \neq (0000)$ *then* $\overrightarrow{b}_i = \overrightarrow{b}_j$.
2. $\forall i \leq k$, $\overrightarrow{b}_i \in \overrightarrow{\mathbb{P}}^2$.

3 The Rewriting Approach

We use the paradigm of conditional rewriting[3] to perform the circuit design. The formalism of constraints used here is also the usual one. For more details see [6], and [7]. All the notations used here are the same as in our previous work [2]. A constrained conditional rule is noted $\mathcal{C} \wedge L \Rightarrow s = t$, in which $\mathcal{C}$ is a list of constraints to be solved by using algorithms in the predefined algebras, or by unification, and L is a list of equations to be solved by rewriting techniques.

When the L part does not exist then we call it an equation. All the inference rules of the maximal-unit-strategy described in [2] are included in our system. The main inference rules we use are *deduction of new rules by Conditional-narrowing, and simplifying conditional rules and inequations*. Let us point out that this inference rule works between an inequation (a pure negative *conditional rule*) and an equation as well. As usual in refutation based methods, we infer in our system until the empty clause is generated. The proof of this empty clause yields the design of the circuit.

4 The Design Specification

The specification of the circuit is divided into three main parts. The axioms represent the inputs, called also equations. All the outputs are represented by a pure negative clause, and finally five conditional rules with the corresponding constraints specify the design of the FPLA and the chip specification. As usual in refutation based methods, CDR infers new clauses until the empty clause is generated. The proof of this empty clause yields the design of the circuit.

All the axioms are ground equations of the form $P\$(out(\overrightarrow{v}), 2) = tt$, where $\overrightarrow{v}$ is a product vector of dimension 2, the second argument of $P\$$.

There are 5 conditional rules specifying the design of the FPLA device. Three of those rules does deduce product vectors, and the remaining ones deduce the output sums. For example the first conditional product rule is as follows:

$$[r \neq 1 \wedge r < n-2] \wedge P\$(out(\overrightarrow{x}), 2) = tt \wedge P\$(out(t), r) = tt \Rightarrow P\$(out(ab(\overrightarrow{x}, t)), r+2) = tt$$

This rule exploits the theorem 1 in order to deduce recursively product vectors of upper dimension. The operator *ab* keeps the resulting product vector in a compact way without expanding it in its vectorial representation. It allows to reduce to memory consuming.

The following rule performs a sum between a product and an already deduced sum. This deduction is performed only through two conditions as shown in the constraint part.

$$[\overrightarrow{x} = simp(t, n) \wedge \overrightarrow{x} \not\preceq \overrightarrow{y} \wedge x \not\leq y] \wedge P\$(out(t), n) = tt \wedge S\$(out(\overrightarrow{y})) = tt \Rightarrow S\$(out(or(\overrightarrow{x}, \overrightarrow{y}))) = tt$$

The constraint $\overrightarrow{x} \not\preceq \overrightarrow{y}$ forbids redundant sums. For example, if x_i's are boolean variables, then the sum between $x_1x_2x_3$ and $x_1x_2 + x_2x_3'$ are not to be performed, because $x_1x_2x_3 \preceq x_1x_2$. The second constraint $x \not\leq y$ means that the syntactic representation of x must be syntactically smaller than y with respect to an arbitrary lexicographic order $<$, defined on the algebraic representation of products and which is extended over the sums. The vectors $\overrightarrow{x}$ and $\overrightarrow{y}$ take part of the predefined algebra, thus all the information corresponding to them are wired like integers and booleans. They are in the low hierarchy level [2]. For example, $x_1x_2 < x_2x_3 + x_1'x_3$ while $x_3 \not< x_2x_3 + x_1'x_3$. This constraint allows us to reduce the process in the search space avoiding the deduction of the same sum (i.e. the same semantic value) though syntactically different.

Finally, the outputs is represented by an inequation. All these outputs are the subterms of the variadic term $A\$(\dots)$. The inequation is as follows:

$$A\$(S\$(out(\overrightarrow{o}_1)),\ \dots,\ S\$(out(\overrightarrow{o}_k)) = tt \Longrightarrow A\$(tt,\dots,tt) \rightarrow tt$$

where $\overrightarrow{o}_1,\dots,\ \overrightarrow{o}_k$ are the column vectors of the truth table. Once all the subterms $S\$(\dots)$ in $A\$(\dots)$ are reduced to tt, the second standard rewrite rule reduces the negative clause to the nil one, thus we get the refutation.

5 Conclusion

In comparison with our previous work[1], thanks to those new properties combined with a goal-oriented deletion criteria, the number of clauses generated is tremendously reduced. This second version of CDR is more efficient than the first one. All the product vectors are stocked in a particular data structure as the BDDs[**?**]. The preliminary statistics show that the memory consuming is reduced to 30% with respect to the first version of CDR. Moreover, thanks to the constraints introduced in the product and sum rules, as presented above, only implicants of the outputs are deduced. The number of these deduced products is divided in some cases by half with respect to the previous version. Unfortunately, this saving is performed only for the product vectors, due to the main property stated in theorem 1. We think that additional improvements of our approach are still possible, especially for vectors sum deduction.

References

[1] N. Andrianarivelo, W. Bousdira, J. Chabin, and Z. Maazouzi. Designing FPLA combinational circuits by conditional rewriting. In H. Prade, editor, *John Wiley and Sons*, pages 373–377, Brighton, UK, 1998. 13th European Conference on Artificial Intelligence.

[2] N. Andrianarivelo, W. Bousdira, and J-M. Talbot. On theorem-proving in Horn theories with built-in algebras. In J. Calmet, J.A. Campbell, and J. Pfalzgraf, editors, *Lecture Notes in Computer Science*, volume 1138, pages 320–338, Steyr, Austria, 1996. Third International Conference on Artificial Intelligence and Symbolic Computation.

[3] N. Dershowitz and J-P. Jouannaud. Rewriting Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers North-Holland, 1990.

[4] P.R. Halmos. *Lectures Notes on Boolean Algebras*. Springer, Berlin, 1974.

[5] Randy H. Katz. *Contemporary logic design*. Benjamin Cummings/Addison Wesley Publishing Company, 1993.

[6] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with Symbolic Constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[7] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types*. PHD Thesis, Universitat Kaiserslautern, FB Informatik, West Germany, 1989.

[8] John F. Wakerly. *Digital Design Principles and Practices*. Prentice Hall International Editions, 1991.

[9] Ingo Wegener. *The Complexity of Boolean Functions*. J. Wiley and Sons, 1987.