Chapter 3

EQUIVALENCE CHECKING OF ARITHMETIC CIRCUITS

Dominik Stoffel
Dept. of Electrical and Computer Engineering, University of Kaiserslautern, Germany
stoffel@eit.uni-kl.de

Evgeny Karibaev
Dept. of Electrical and Computer Engineering, University of Kaiserslautern, Germany
karibaev@eit.uni-kl.de

Irina Kufareva
Department of Radio Physiks, Tomsk State University, Tomsk, Russia
irene@molsoft.com

Wolfgang Kunz
Dept. of Electrical and Computer Engineering, University of Kaiserslautern, Germany
kunz@eit.uni-kl.de

Abstract Although equivalence checking technology has matured greatly during the last few years and designs with millions of gates can be handled, some specific problems remain to be difficult. Formal verification of arithmetic circuits, especially if multiplication is involved, is one of these problems. In this chapter we analyze origin and nature of this problem. We then review the most important research contributions targeting equivalence checking of arithmetics. Specifically, we outline techniques exploiting arithmetic functional properties and techniques based on binary decision diagrams, both on the bit and word levels. We also report on our own experiments with Multiplicative BinaryMoment Diagrams (*BMDs). Finally, we introduce a new pragmatic approach to equivalence checking of arithmetic circuits. It extracts from a gate-level netlist an arithmetic bit-level representation of the circuit. Verification is carried out on this representation using a simple arithmetic calculus rather than a Boolean one. We show experimental results for successfully equivalence checking a large number of industrial multipliers as well as other circuits implementing more complex arithmetic expressions.

Keywords: Equivalence checking, arithmetic verification, datapath verification, multiplier verification, decision diagrams, arithmetic bit level

R. Drechsler (ed.), Advanced Formal Verification, 77-123.
1. Introduction

Modern circuit design flows increasingly employ formal verification techniques in order to ensure quality and reduce time-to-market by avoiding bug-related design iterations. Equivalence checking has become a regular step in the flow. Modern tools for equivalence checking are capable of verifying designs with millions of gates in very short times.

The great success of equivalence checking technology is due to numerous research advancements in this field during the past decade. An important idea on which a typical equivalence checker is based is to exploit structural similarity between the two circuit models being compared [1, 2]. Structurally similar circuits contain a lot internal nodes implementing equivalent circuit functions. These internal equivalences, sometimes called cut points [3] can be used to efficiently break the verification problem down into smaller ones as has been explored by several researchers [4, 2, 5, 3, 1, 6].

In many applications of equivalence checking, the two circuit models being compared exhibit a great amount of structural similarity. For example, during logic synthesis, the transformations of a gate netlist into another gate netlist are of fairly local scope so that many equivalent internal functions remain. However, in some cases, structural similarity is not given. An important example that occurs often in practice is the verification of arithmetic circuits. The problem occurs when an RTL-level (register transfer level) specification of a circuit must be compared against a gate-level implementation, e.g., when verifying the correctness of a logic synthesis step. Figure 3.1 illustrates this case. The verification engines in a typical equivalence checker all operate on gate-level circuit models. Hence, in order to compare an RTL specification with a gate-level implementation, the frontend of the verification tool first has to generate a gate-level representation of the specification. The process is similar to the logic synthesis step that produced the implementation. The two gate netlists can then be compared by the backend engines to verify equivalence or produce a counterexample. When the design contains arithmetic functions, this approach is bound to fail. The problem is that the two gate netlists hardly contain any structural similarity at all. The reason for this lies in the great flexibility when implementing arithmetic functions.

In general, arithmetic functions in digital circuits, such as addition, subtraction, multiplication and division, are implemented using addition as the base function. Subtracting a number $X$ in two’s complement notation from a number $Y$, for example, is implemented by inverting all bits of $X$, adding 1, and adding $Y$. Multiplication is also based on