INTEGRATION, the VLSI journal 53 (2016) 100–114

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Invited paper

A dynamic specification to automatically debug and correct various divider circuits

Mohammad Hashem Haghbayan, Bijan Alizadeh *

School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Received 17 February 2015
Received in revised form
6 December 2015
Accepted 9 December 2015
Available online 29 December 2015

Keywords:
Arithmetic and logic units
Debugging aids
Diagnostics
Verification
Formal models

ABSTRACT

This paper presents a formal technique to verify and debug division circuits on fixed point numbers. The proposed technique is based on a reverse-engineering mechanism of obtaining a high level model of the gate level implementation and also introducing an intermediate representation of the specification that makes equivalence checking between two models possible. The main advantage of this representation is the fact that the specification is dynamically updated according to the information obtained from the implementation. At the end, if two updated models are not equivalent, possible bugs can be localized and then corrected automatically by analyzing the difference, if possible. Experimental results show the robustness of the proposed technique in comparison with other contemporary methods in terms of the run time and also show that two orders of magnitude average speedup is obtained.

1. Introduction

As the size and complexity of digital systems increase continuously, design verification and debug are quickly becoming more important. From a verification point of view, one of the most difficult parts in such complicated designs is arithmetic datapaths and their components, such as multipliers and dividers. Most of hardware verification tools are based on bit-level methods like Binary Decision Diagrams (BDDs) or Satisfiability (SAT) solvers that are not scalable because they suffer from space and time explosion problems when dealing with large arithmetic circuits [1,2]. To alleviate this problem, word-level decision diagrams like Binary Moment Diagrams (BMDs), *BMDs and K*BMDs have been proposed [3]. However, these diagrams and decision procedures still suffer from the memory explosion when dealing with wide ranges of arithmetic operations due to the fact that functions are defined over binary variables as a bit vector rather than integer variables. Another methodology transforms arithmetic circuits into propositional logic formulas and then satisfiability tools, i.e., SAT or Sat-Modulo Theories (SMT) solvers, are employed to verify the validity of the formulas [11]. Although Boolean SAT-based methods are very efficient regarding memory requirement, they have not been very successful for designs containing large arithmetic units due to computational complexity. SMT-based verification techniques are also suffering from the same problem because when dealing with arithmetic circuits most SMT solvers resort to bit blasting and therefore they show the same performance limitations as pure SAT solvers [11]. In [24] the authors proposed a SAT-based solution for logic diagnosis of multiple faults or design errors in combinational and sequential circuits. Again the proposed method is not scalable in terms of time and memory space since SAT solver is used for the verification and diagnosis process. In addition, such decision diagrams and decision procedures are not able to handle division circuits which are one of the most important operations of arithmetic computations.

A well-known approach in verifying arithmetic circuits is to extract arithmetic operations from the gate-level implementation and then generate an arithmetic model to be compared with a high level specification [5,10]. For example this technique can check the equivalence of integer multipliers based on a bit level reverse-engineering approach. The main challenge is to efficiently extract an arithmetic bit level description of a circuit from a given gate-level arithmetic circuit. Reverse engineering could be considered as a very pragmatic approach to multiplier verification. As the number of possible architectures for a multiplier is limited one may incorporate a variety of architectures in the frontend of the equivalence checker and repeat the comparison for all of them. The verification approach in [5,10] benefits from an efficient reverse-engineering process in extracting a network of half-adders from the gate-level circuit, while there is no need of an exhaustive process to map carry signals.

The authors of [4] proposed a formal method to describe and verify arithmetic circuits using symbolic computer algebra. The

* Corresponding author.
E-mail addresses: h.haghbayan@ece.ut.ac.ir (M.H. Haghbayan), b.alizadeh@ut.ac.ir (B. Alizadeh).

http://dx.doi.org/10.1016/j.vlsi.2015.12.004
0167-9260/

main idea is to describe arithmetic circuits with integer equations in a hierarchical manner. These equations (potentially very large) are formally verified by formula manipulations based on Groebner basis [7]. The authors of [12] modeled a gate level implementation by polynomials over rings Z_2^n and then the normal forms are computed with respect to the Groebner basis using modern computer algebra algorithms. This model is complicated and is not scalable to practical designs. The authors of [13], take the advantage of Groebner basis theory to verify arithmetic circuits over rings Z_2^n. Instead of mapping each gate to a polynomial, they have significantly reduced the number of polynomials by finding fanout-free regions and representing the whole region by one single polynomial. Although these techniques seem to be robust for large arithmetic circuits, they are not able to handle dividers. One of our contributions in this paper is extending the debugging technique in [5,10] in order to verify and debug large arithmetic systems that contain dividers in addition to adders and multipliers.

There has been considerable interest in using theorem proving techniques to verify arithmetic circuits such as multipliers and dividers [8,14,15]. In these techniques, gate level implementation and high level specification should be translated into or described in an appropriate specification language which is usually equation-based. Then, a proof by induction is attempted to check the equivalence between two models that needs user guidance. Symbolic Trajectory Evaluation (STE) based formal verification approach has been widely used at Intel in the past for various microprocessor designs to formally verify data-path designs [20]. The authors of [21] implement industrial level tools tied together in the ACL2 theorem prover and focus on hardware validation. Although such techniques are scalable enough to be able to verify large arithmetic circuits, their main drawback is to balance efficiency with generality since a single proof strategy is being applied to all theorems. In other words, the main problem with theorem proving techniques is the lack of expertize and documentation. It takes a considerably long time to learn theorem proving techniques and also there is a strong need for libraries of specifications to be established, and more automated tools and approaches.

In this work, we assume that a gate-level implementation as well as a high level specification of a complex arithmetic datapath on fixed point numbers is given. Fig. 1 shows the basic idea behind the proposed debugging technique. As can be seen in the figure, from the specification, expected arithmetic operations are extracted and represented using functional bit level adder (FBLA) representation. FBLA is an extension of the BLA in [5,10] that will be explained in Section 3. Based on such a model, adders and other controlling logics are extracted from the gate-level implementation and then represented using logical bit level adder (LBLA) as will be discussed in Section 4. During the extraction of high level components from the implementation, FBLA will be updated if needed. This dynamic updating mechanism allows the algorithm to prepare FBLA for various structures based on extracted adders or repetitive logic cones. Then, the equivalence between the logics extracted from the implementation and expected arithmetic operations extracted from the specification are checked using a canonical mixed bit- and word-level decision diagram called Horner Expansion Diagram (HED) [16]. In the case of non-equivalent functionalities, error correction is performed by replacing buggy logics with expected functions from the functional bit level model. It should be noted that the expected functions in FBLA model are gradually updated during processing the implementation and cannot be specified independent of the implementation.

In addition to the above-mentioned methods, there are many methods for verification of dividers specially SRT dividers [1,17–19]. In general most of them cannot support optimization techniques and in some cases the run time is very high.

In summary, the main contributions of this paper in comparison with our previous works [5,10,23] are as follows:

- An algorithm tightly integrating verification and debugging of fast array dividers and sequential dividers even in the case of speeding up the remainder update stage (quotient selector logic) in the implementation where the design makes use of a sum and carry representation for the remainder, as will be discussed in Section 4.
- Dynamically representing the specification that can extend our proposed debugging algorithm in such a way that can be applied to optimized arithmetic circuits while the techniques in [5,10] are not able to handle such designs, as will be discussed in Section 5.
- Formally describing a wide range of arithmetic circuits including fast array dividers as well as sequential dividers by proposing functional bit-level and logical bit-level representations while the technique in [23] is just applicable to restoring and non-restoring dividers.
- Showing empirical results to prove that our proposed debugging technique enables us to verify and debug large industrial arithmetic circuits within practical time.

The rest of the paper is organized as follows: Section 2 presents the background on division arithmetic. In Section 3 basic representations are introduced. Section 4 presents the proposed debugging technique for different types of division circuits. In Section 5, we explain how to address various optimization issues. Section 6 demonstrates experimental results and Section 7 concludes the paper and presents the future works.

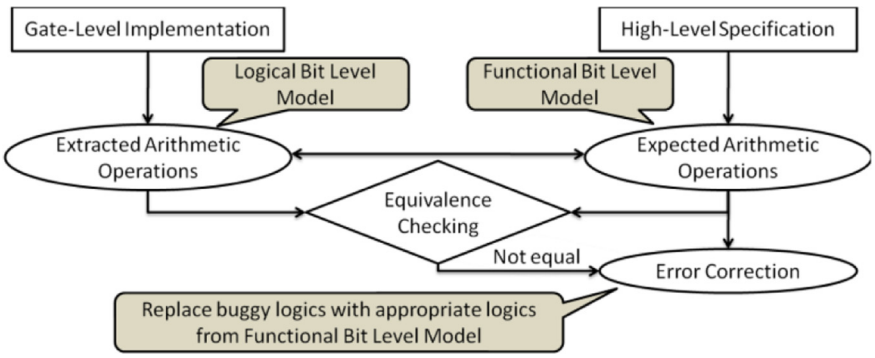

Fig. 1. Basic idea of the proposed debugging and correction techniques.