Chapter 1

Introduction

For years very large scale integrated circuits and the resulting digital systems have conquered a place in almost all areas of our life — even in security sensitive applications. Complex digital systems control airplanes, have been used in banks and on intensive-care units. Hence, the demand for error-free designs is more important than ever. In addition, economic reasons also underline this demand: the design and production process of present day VLSI-circuits is highly time- and cost-intensive. Moreover, it is nearly impossible to repair integrated circuits. Thus, it is desirable to detect design errors *early* in the design process using computer-aided tools and not just after producing the prototype chip. Circuits have become more complex regarding to their design and the tasks that they are designed for as the level of integration on an integrated circuit (IC or chip) itself has been increasing. While a handful of devices were integrated on the first circuits in the 1960s, circuits with over a million devices can be manufactured nowadays. With that, the probability increases that design errors remain undetected (e.g. the problems with the INTEL-Pentium-processor in 1995). In other words, it is not just more important to get error-free designs, but it also becomes an increasingly difficult task for a team of human designers to carry out a full design without errors. So, the development and improvement of verification tools that are able to prove the correctness of design of present day digital systems has obtained major significance.

Verification is the comparison of two models for consistency. Traditionally, checking the correctness of a system is done by simulation based methods. In such an approach, the designer has to create a complete set of test vectors which represents all possible inputs of the system. Then, the outputs of the design for each of these input vectors must be analyzed in order to guarantee the correctness of this design. It is obvious to see that this process is very CPU-time intensive and thus impractical for larger designs.

As a result, another kind of verification strategies have been becoming popular: strategies that use formal verification techniques. By using these techniques, the correctness of a design for all input combinations can be guaranteed.

Aspects of Formal Logic Verification

Nowadays, a digital system is designed with the help of computer–aided design tools that work at different, almost independent levels in a hierarchical manner. Such a process of designing usually starts with describing the system, which has to be designed, in an abstract model. On this model, an extensive simulation is made. Then it becomes the *golden specification* [20]. Starting with this golden specification, a detailed implementation is derived, passing through different design levels step by step. First a synthesizable behavioral Register-Transfer-Level (RTL) description is derived which describes the block structure behavior of the design. This description is then translated into the structural description describing the combinational logic of the system. From this the transistor netlist is derived which finally leads to the physical layout description.

To be able to catch bugs as early as possible in the above design process, it is important to verify the functionality of the design at *every* level of this process against the golden specification. At first glance, this so called *implementation verification* should play a minor role when computer-aided design tools are used. Those methods should provide *correctness by construction* [23]. However, because of the widespread use of synthesis tools this is not the case. While the synthesis algorithms have guaranteed properties of correctness, their software implementations cannot be guaranteed to be error–free. The same holds for the implementation of the operating system and the data bases. So there is a need of implementation verification methods that support synthesis tools.

This verification proceeds in two phases [20]. In the first phase, a Boolean network is extracted from the actual description (e.g., [7, 35]). Then, in a second phase, this Boolean network is verifyed by some formal verification methods against the original (golden) specification. It is this part of the verification process, that is the focus of this thesis. Moreover, we need to distinguish between the verification of *combinational* and *sequential* circuits. In combinational circuits, the outputs depend only on the current inputs, where in sequential circuits, the outputs depend not only on the current inputs but also on the past sequence of inputs.

The Combinational Permutation Equivalence Problem

In this thesis, we focus on a special problem in *combinational* logic verification. The tools used at the different levels in the design process may have their own naming conventions for the inputs or the outputs of the circuit which has to be designed. Then the input/output correspondence between the different descriptions of the circuit design gets lost. However, before we can verify the equivalence of the two Boolean networks, we need to know this correspondence. The main focus of this thesis is to determine such a lost correspondence for the inputs of two Boolean networks. This problem is $\mathcal{NP}$–hard. Thus, we need to consider techniques that are non-exhaustive and work well in practice. We will introduce such techniques and demonstrate their practical efficacy. Moreover, we shortly explain how they can be used to find a lost correspondence for the outputs of two Boolean networks, although a evaluation of this would be beyond the scope of this thesis.

Note that a solution for the permutation equivalence problem can be used in another part of circuit design as well. It also is important for Boolean matching. Boolean matching is the key operation

in technology mapping. It checks whether an element of a given library can be used to implement a part of a Boolean function. This can be formulated as checking the equivalence between a given Boolean function, called the *target function*, and the set of functions representing a library element. Often this is considered for any permutation of the input variables.

The Latch Correspondence Problem

Furthermore, we demonstrate how the techniques that we have used to handle the *combinational* permutation equivalence problem can be easily applied to a problem arising from *sequential* logic verification.

Verifying general sequential circuits is a very difficult task. The techniques that exist so far for verifying those circuits are not applicable to very large designs. So their use in a practical design methodology is limited. However, some sequential verification problems can be reduced to a combinational verification problem. One of these is the case when the corresponding latches (memory elements) in the two designs that we have to test for equivalence, are identified. Thus it is desirable to have a way to establish this correpondence between the latches of two given sequential circuits. More formally, we define the following problem of sequential logic verification (we call it the latch correspondence problem): given two sequential circuits, does there exist a correspondence between the latches of these two circuits, such that the combinational parts are equivalent using this latch correspondence?

It is easy to see that there is a connection between the combinational permutation equivalence problem and this latch correspondence problem. We show that this is indeed the case, underline the differences between the two problems, and demonstrate how we can apply the techniques of the combinational permutation equivalence problem to the latch correspondence problem.

Contents

After our introduction in **Chapter 1**, we discuss the background of this thesis in **Chapter 2**.

In **Chapter 3** and in **Chapter 4**, we describe the combinational permutation equivalence problem, explain why it is $\mathcal{NP}$–hard, and provide techniques to handle the problem. Moreover, an extensive analysis and evaluation of our practical experiments is made.

Chapter 5 describes, how the techniques used to handle the combinational permutation equivalence problem can be applied to the problem of finding a correspondence between the latches of two sequential circuits.

Finally, we give a summary of the techniques provided in Chapters 3, 4, and 5 and mention some ideas for future projects based on the results of this thesis in **Chapter 6**.