Chapter 3

SYSTEM-LEVEL VERIFICATION

The parts of the proposed design and verification flow covered in this chapter are shown in Figure 3.1. As already mentioned, for modeling a system in this book the system description language SystemC is used. Thus, from the textual specification the initial system-level model is directly described in SystemC. Following the design methodology of SystemC the system-level model is very abstract and can be simulated at this high level of abstraction already by compiling the model into the executable specification. This allows for efficient design space exploration. After analyzing the results of a certain design direction the designer can go back and revise design decisions (for simplicity this loop is not shown in the figure). Since not all details have been modeled already, this can be accomplished with moderate costs. Also part of the design space exploration phase is to check hardware/software trade-offs. Hence, hardware/software partitioning is performed to meet the requirements of the specification. During the development of the system-level model verification is started.

In the middle of Figure 3.1 the dedicated verification techniques and the respective quality check that are proposed in this chapter for the system level are depicted. In the first part of this chapter constraint-based simulation is considered which overcomes the limitations of “pure” simulation. Constraint-based simulation is based on stimulus generation by constraint solving. The resulting stimuli will in particular cover corner case test scenarios which are usually hard to identify manually by the verification engineer. For SystemC the SystemC Verification (SCV) library [Sys03] offers constraint-based simulation. The underlying constraint-solver of the SCV library is based on formal methods. More precisely, Binary Decision Diagrams (BDDs) are used to represent the constraints. In Section 3.1 the scenario of constraint-based simulation in
general and thereafter in the context of SystemC using the SCV library is considered. Section 3.2 presents two improvements: First, new operators for the specification of SCV constraints are provided. Second, the uniform distribution among all solutions of a constraint is guaranteed for maximizing the chance of entering unexplored regions of the design state space. Both improvements have been published in [GED07]. In Figure 3.1 they are summarized in the verification task improved constraint-based simulation.

While specifying complex non-trivial constraints (and for example focusing on special scenarios), the verification engineer is faced with the problem of over-constraining, i.e. the overall constraint defined for a certain test scenario has no solution. In this case the root cause of the contradiction has to be identified and resolved. Given the complexity of constraints used to describe test scenarios, this can be a very time-consuming process. Thus, in Section 3.3 a fully automated approach for contradiction analysis is introduced. The method determines all “non-relevant” constraints and computes all reasons that lead to the over-constraining (see task contradiction debugging in the figure). The approach has been published in [GWSD08].

Finally, to ensure the resulting verification quality we investigate the question how thorough the design was tested. Hence, in Section 3.4 an approach to measure the quality of the testbench is presented. Dedicated code coverage techniques that have been developed for SystemC models are used for the analysis. As a result a coverage report is generated that shows all statements in the model that have not been executed during simulation (see task coverage check in Figure 3.1). The method has been published in [GPKD08].