Architectural Rule Checking for High-level Synthesis

Jie Gong, Chih-Tung Chen and Kayhan Küçükçakar

Unified Design System Laboratory
Motorola, Inc. MD EL714
2100 E. Elliot Road, Tempe, AZ 85284

Abstract

Verifying an implementation produced from high-level synthesis is a challenging problem due to many complex design tasks involved in the design process. In this paper, we present an architectural rule checking approach for high-level design verification. This technique detects and locates various design errors and verifies both the consistency and correctness of an implementation. Besides describing different rule suites, we also report a working environment for the architectural rule checking. Finally, we highlight the value of the proposed approach with a real-life design.

1. Introduction

Design verification, which is to find out whether the design produced is the one specified, is indispensable for both interactive and automatic high-level design. It is easy to accept that the interactive approach needs design verification since the interactivity opens a pathway for introducing errors into the design while enabling the utilization of designers' expertise during the design process. It is not obvious why a design synthesized automatically needs to be verified since one may argue that the tools should produce correct designs by construction. However, in reality, there is no such guarantee unless the tools can be formally validated. To validate a large software system such as a high-level synthesis system is still impractical, if not impossible.

Although comprehensive design verification is indispensable for both interactive and automatic high-level synthesis, little work has been done in this area. The only work related to error detection is reported in the interactive synthesis system RLEXT [1], in which some design properties are used to detect design violations after designer modifies the structure by adding or deleting components and interconnect. Our work differs from RLEXT in that RLEXT mainly focuses on detecting and fixing errors during resynthesis while our work extensively detects errors for various design aspects in the high-level synthesis process such as library component, behavioral specification, scheduling, datapath structure, binding, resource conflict and value-life time. The key contribution of our architectural rule checking approach is that it provides a step-wise design verification capability which can detect errors early and identify the locations of errors quickly.

2. Models and Rules

The infrastructure of the rule checking is shown in Figure 1. Design is captured in a global database through internal models. Control/data flow graph (CDFG) is used to represent the data and control dependencies residing in a behavioral specification. Timing graph (TG) is used to represent the execution sequence imposed by the scheduling on the operations in the CDFG. Structure graph (SG) is used to represent the datapath of the design. Library Component Model (LCM) is used to characterize the components used in the datapath. Design can be modified by a set of editors. The rule suites are used to verify design properties. Due to space limitation, we will just give an example rule for each rule suite to show what kind of checking each rule suite performs. For details on rules and models, please refer to [2].

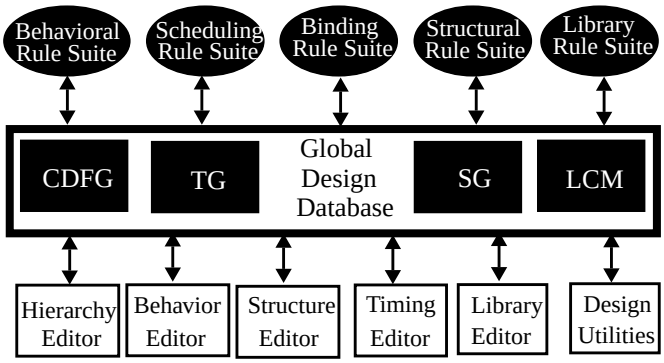

Figure 1. Infrastructure of the rule checking

2.1 Behavioral Rule Suite

The behavioral rule suite is to check the consistency of a behavioral specification. A sample rule would be: *All variables used in the behavior must be defined before their use*.

2.2 Scheduling Rule Suite

Scheduling is a task which assigns time steps to each operation in the CDFG. A schedule is correct if its TG is consistent with its CDFG. A sample rule would be: *All*

dependencies specified in the CDFG are not violated by the schedule in the TG.

2.3 Library Rule Suite

The library rule suite is to check the consistency of each library component. A sample rule would be: *Data, control and clock pins have no intersection.*

2.4 Structural Rule Suite

The rules in the structural rule suite are used to check the consistency of the structure (i.e. datapath) of the design. A sample rule would be: *For each input pin of module, there is no more than one driver.*

2.5 Binding Rule Suite

Binding is a task which maps behavior to the structure and there are three types of bindings. Operation binding is to map operations in CDFG to the functional units in SG. Value binding is to map variables in CDFG to storage modules in SG. Path binding is to map value transfers in CDFG to paths in SG.

A consistent binding is one in which all three types of bindings are consistent. A correct binding is a consistent binding in which there is no resource conflict and all values are alive when they are needed. A sample rule would be: *Value life times of variables bound to same storage module should not overlap.*

3. A Working Environment

The proposed architectural rule checking approach is implemented in an interactive behavioral synthesis system called Matisse[3]. At any stage of the design process, the designer can invoke the rule checking tool to check the current design.

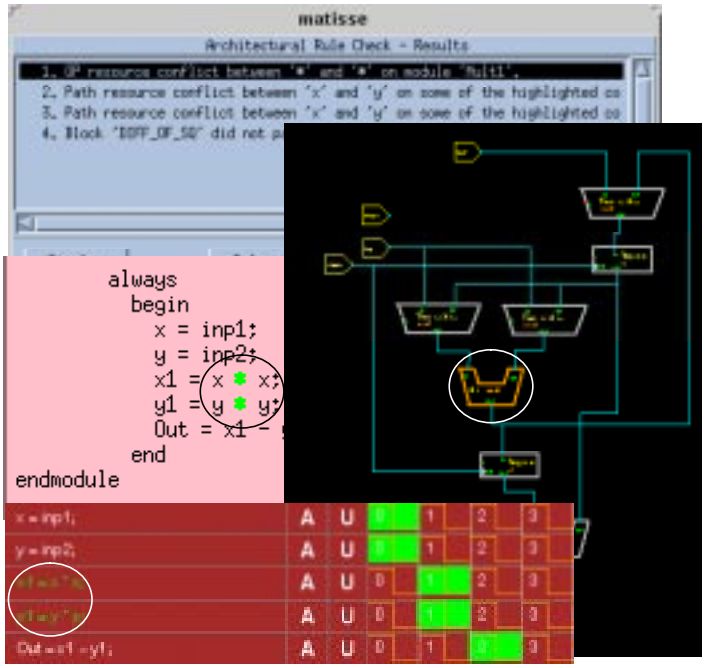

Figure 2. Display of Rule Checking Result

Figure 2 shows an example of the rule checking where the error or warning messages are displayed in a checking result window. The designer can then select a message and the corresponding design objects will be displayed in related editors. For example, in Figure 2, a resource conflict error is detected. After the designer selects the resource conflict error message, the behavioral editor will highlight the related operations which have the resource conflict and the structural editor will highlight the resource to which those operations are mapped and the timing editor will highlight their schedules. With this graphic display of where the error is, designers can correct design errors quickly. After any change to the design, designers can either run the rule checking again or continue displaying other reported errors and incrementally repairing them.

4. Results

The value of our architectural rule checking was manifested by the redesign of the Motorola 68HC11 microcontroller. It is very difficult, if not impossible, for designers to find out some of the errors detected by the rule checking. For example, in 68HC11 design, it took about 11 million path comparisons to find out some of the path conflicts in the design. Without the rule checking capability, a designer has to rely on the simulation to find out these errors, which is a difficult task. The rule checking also has the benefit of pinpointing the location of an error, and not just its existence. With the behavioral, timing and structural editor windows open, the rule checking highlights the behavioral tokens, statements or structural resources that cause the problem. This allows errors to be corrected quickly.

5. Conclusions

We have demonstrated an architectural rule-checking approach for high-level synthesis. It consists of a set of rule suites for statically checking problems existing in the behavior, schedule, structure, library components and bindings. The rule-checking mechanism is formal since it is built on well-defined design models. It is also extendable since additional rules can be added to the rule suites. The actual use of the rule checking capability provided up to an order of magnitude reduction in the design debugging time.

6. References

[1] D.W. Knapp. Manual Rescheduling and Incremental Repair of Register-Level Datapaths. *International Conference on Computer-Aided Design*, 1989.

[2] J. Gong, C. T. Chen and K. Küçükçakar. Multi-dimensional Rule Checking for High-level Design Verification. *IEEE International High Level Design Validation and Test Workshop*, 1997.

[3] K. Küçükçakar, C. T. Chen, J. Gong, W. Phillipsen and T. E. Tkacik. An Architectural Design Tool and Methodology for Commodity IC Design. To appear in *IEEE Design and Test of Computers*.