# Orchestrated Multi-Level Information Flow Analysis to Understand SoCs *

Görschwin Fey
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
fey@informatik.uni-bremen.de

## ABSTRACT

Complex *Systems on Chip* are developed by large design teams integrating various different blocks. Typically, no single person in the design team understands all details of such a design. Integrating new designers into the team as well as debugging failures or performance problems becomes a time-consuming cost-generating threat to the overall project.

We envision tool support for these critical steps. The paths of information flow are automatically extracted and explanations for certain behavior are derived by reasoning engines. Then, the designer interactively explores the design within this environment.

## Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

## General Terms

Design, Productivity

## 1. INTRODUCTION

Complex *Systems on Chip* can only be created by large design teams. These teams assemble third party blocks, communication blocks, newly designed sophisticated blocks for application specific operations, and customized module interfaces. Consequently, no single person in the team knows all details about the entire design. In this situation tool support is required not only for complex computation intensive tasks like synthesis or routing, but also for work that typically needs human intuition and interaction.

Recent examples for such tool support have been proposed, e.g., for debugging or verification. For debugging in-field failures, data is gathered and potential reasons for the failure are pinpointed [5]. In design debugging, failing traces are explained to the designer by relating the failure to very similar passing traces [4]. If formal verification succeeds, the partial specifications in terms of formal properties may be hard to understand. These partial specifications are therefore lifted to higher level specifications that capture the *design intent* [1, 7]. Finally, a formal specification may not be available. But even from simulation traces properties [6] or transaction level descriptions [3] are semi-automatically extracted. All of these approaches provide innovative solutions to speed up tasks that otherwise require time consuming manual interaction. The designer still has to be in the loop as human intuition and creativity cannot be replaced in the design flow.

We propose to go a step further with such tool support. *Orchestrated Multi-level Information flow analysis* (OMI[1]) familiarizes a designer with a complex SoC and supports her in debugging failures. *Paths of information flow* are automatically extracted and *explanations* for the activation of certain paths are given on demand. For this purpose analysis techniques from testbench creation, formal verification, or compiler construction are orchestrated. The analysis takes multiple design levels from a global component view down to module descriptions in *Hardware Description Language* (HDL) into account. Descriptions on architectural or transaction level can be utilized as additional data sources. The designer sees one path of information flow on the component level at the beginning that was extracted from a testbench. Starting at this view she interactively navigates along this path where she may decent into the design to understand interfaces or branching conditions on a detailed level. Alternatively, the designer may ask OMI whether some path from one module to another exists. OMI explains how to activate such a path.

## 2. ORCHESTRATED MULTI-LEVEL INFORMATION FLOW ANALYSIS

OMI relies on various analysis techniques whose results are integrated into an abstract view of the SoC design. Figure 1 shows an overview of the framework. The envisioned analysis components of OMI are described in the following. The interactive navigation is considered afterwards.

### 2.1 Structural and Functional Paths

The simplest analysis step is the extraction of structural paths on the component level. These paths are extracted from the global interconnect structure of the system, e.g., buses, buffered communication via dedicated memory, and specialized point-to-point connections. Due to their large number, structural paths are not explicitly represented, but represented as an abstracted communication graph where each communication device – sender, recipient, or channel – is represented as a node. Edges between nodes denote communication lines.

*This work has been funded in part by the German Academic Exchange Service (DAAD, grant no. D/09/02091) and in part by the German Research Foundation (DFG, grant no. FE 797/6-1).


*DAC 2011,* June 5 - 10, 2011, San Diego, California, USA.


[1]"Omi" is a German word for "granny".

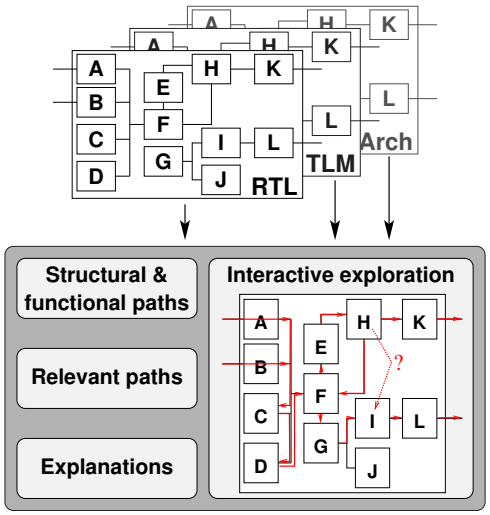


**Figure 1: Overview of OMI**

On the module level control flow and data flow are extracted by standard techniques known from compiler construction. This information is extracted on demand for each module that becomes relevant during the interactive exploration.

## 2.2 Relevant Paths

The communication graph represents possible paths of information flow. Only a small subset of these paths will be active in the SoC. Relevant paths of information flow are extracted from the running system. Tagged simulation is used for this purpose. Tags are added to data packets entering the system, these tags are carried through the system to follow the flow of information for the stimuli generated by the testbench. Note, that tagged simulation does not require descriptions of all components on the HDL level. Tagged simulation can be performed on any representation of the system that can be simulated and represents data accurately.

The result of tagged simulation is an overall picture of the information flow on a global component level and also on the module level (see the red arrows in Figure 1). This picture of the information flow is necessarily incomplete as the number of possible communication paths increases exponentially with the number of modules if powerful communication switches or bus structure are available. But commonly used paths for information flow in the system become visible. Also statistical knowledge is generated showing which paths are used most, which paths are only used at start-up etc. All this information is gathered before the designer starts to work with OMI.

## 2.3 Explanations

While seeing the paths of information flow already helps to understand the typical communication within the SoC, further information is required to understand why these paths are taken. For this purpose reasoning engines as applied in formal verification derive explanations for certain aspects of the system behavior. An explanation is defined as the minimal set of assignments that are required for a certain event to occur. Such a set of assignments may not be unique, but with respect to the assignments that occurred during tagged simulation, relevant candidate sets are extracted. Moreover, the traces seen during tagged simulation reduce the search space for the formal methods. As a result explanations can be calculated on-demand where needed by the designer. The explanations can only be derived from descriptions accessible to the reasoning engines. This holds, e.g., for HDL source, but also for parts of transaction level descriptions and other higher level descriptions. The following questions provide a few examples where explanations provide an answer:

- Why is data transferred from Module H to Module I (see dotted arrow in Figure 1)?
- Why is a certain control path activated in a module?
- What are the conditions to activate a communication interface?

## 2.4 Interactive Exploration

A designer hardly understands an SoC after being confronted with the above information in a static manner. Also when debugging failures, explanations are required at very specific spots within the design, new questions arise and have to be answered. Therefore OMI will be used interactively.

To get a first insight into the SoC the designer may select a certain path of information flow and then surf along this path through the design. Questions can be asked interactively, the above procedure provides explanations, e.g. by way of logic queries [2]. Alternatively, when debugging, the designer may choose an arbitrary point for starting the navigation and then trace backwards along the information flow to understand, e.g., that modules have been assembled in an unspecified way.

While navigating through the design, component level views, structural views on module level, and source views are linked. Zooming within the hierarchy is supported and the paths of information are visible in all views.

# 3. EXPECTED IMPACT

Not only new designers entering the team, but also those designing a proprietary block may use OMI to understand under which conditions their block operates. By this, the designer immediately answers these questions herself without waiting for a team meeting and without interrupting other team members. Also for debugging failures the explanatory features of OMI support a designer in debugging even parts of the SoC she is not familiar with.

As a consequence, OMI increases design productivity.

# 4. REFERENCES

[1] P. Basu, S. Das, A. Banerjee, P. Dasgupta, P. Chakrabarti, C. Mohan, L. Fix, and R. Armoni. Design-intent coverage: A new paradigm for formal property verification. *IEEE Trans. on CAD*, 25(10):1922 –1934, 2006.

[2] W. Chan. Temporal-logic queries. In *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, 2000.

[3] A. DeOrio, A. Bauserman, V. Bertacco, and B. Isaksen. Inferno: Streamlining verification with inferred semantics. *IEEE Trans. on CAD*, 28(5):728 –741, 2009.

[4] A. Groce and W. Visser. What went wrong: Explaining counterexamples. In *Model Checking of Software: SPIN Workshop*, number 2648 in Lecture Notes in Computer Science, pages 121–135, 2003.

[5] S.-B. Park, T. Hong, and S. Mitra. Post-silicon bug localization in processors using instruction footprint recording and analysis (IFRA). *IEEE Trans. on CAD*, 28:1545–1558, 2009.

[6] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke. Advanced verification by automatic property generation. *IET Computers and Digital Techniques*, 3(4):338–353, 2009.

[7] A. Sinha, P. Dasgupta, B. Pal, S. Das, P. Basu, and P. P. Chakrabarti. Design intent coverage revisited. *ACM Trans. on Design Automation of Electronic Systems*, 14:9:1–9:32, 2009.