Automatically Connecting Hardware Blocks via Light-Weight Matching Techniques *Extended Abstract*

Jan Malburg* Niklas Krafczyk* Goerschwin Fey*†

*Institute of Computer Science
University of Bremen
28359 Bremen, Germany
malburg@informatik.uni-bremen.de

†Institute of Space Systems
German Aerospace Center
28359 Bremen, Germany
Goerschwin.Fey@dlr.de

I. Introduction

Modern chip designs are composed out of several different blocks. Blocks are typically described in a *Hardware Description Language* (HDL). Often these blocks are from different developers or even third party blocks licensed from other companies and have to be assembled into a single chip. Writing the corresponding connections is a tedious task for a developer, as he needs to connect several hundreds or even thousands of different ports. Hence, automation is desirable.

Currently, tools for automatically connecting ports of different blocks require either exact name matching, e.g., Emacs Verilog-Mode [1] or require additional input, e.g., MKTREE[2], ShapeUp [3]. MKTREE requires a description of the intended connection and ShapeUp uses a specification of the different blocks in order to generate correct connections.

In this work we present a technique for connecting ports of different blocks, that neither requires exact name matching nor additional user input. The presented technique focuses on fast light-weight techniques based on similarities between strings.

II. Technique

The basic flow of our approach is shown in Figure 1. As input a list of Verilog modules, the source code of each module, and, optionally, the amount how often each module should be instantiated is used. First, a set of prohibiting heuristics is applied on the modules. The prohibiting heuristics mark connections as forbidden, either because they are very unlikely or would result in nonsynthesizable code.

Next, supporting heuristics are used to compute a likelihood for two ports to be connected. We use a name matching heuristic, an extended name matching heuristic and an event checking heuristic. The name matching heuristic is based on the similarity of the port-names. We assume that for most design the name of the port corresponds to the data which is sent over the port. The name matching heuristic can use three different string-similarity metrics: Jaro-distance [4], Levenshtein-distance [5] and longest-common-substring. The extended name matching heuristic not only considers port-names but also the names of the modules and their submodules. The event checking heuristic tries to find the clock and reset signals of a design and connect them correspondingly.

The last step in the computation is the application of a connection strategy. We implemented several different strategies of different complexity for choosing the connections between the ports. All connection strategies use a threshold value which decide if a connection should be created at all. Based on preliminary results the threshold value was set to two third of the maximal value of the supporting heuristics. The simplest is a greedy strategy, always creating the connection which has the highest value assigned by the supporting heuristics and no previously created connection prevents the connection. More complex connection strategies rate connections higher, if the modules which they connect are already connected by other connections or consider all possible connections of a port and prefer those connections for which the other port has only poorly ranked alternatives.

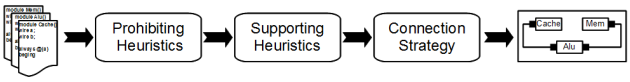

Figure 1. The basic flow of our approach

III. Evaluation

For evaluating our approach we use eight designs of different size, of different purpose, and from different authors. For the evaluation we removed the sub-module instantiations from the top-modules of the designs and then applied our approach to recreate the instantiations. We used Emacs Verilog-Mode as a baseline-comparison. For comparing both approaches we used a metric approximating the effort a developer saves using an automatic tool.

The evaluation shows that in most cases the event checking heuristic reduces the quality of the result. For several of the considered designs, the more advanced connection strategies are heavily affected by small changes to the supporting heuristic. However, the pure greedy approach achieves good result for all the designs. Further, the evaluation showed that the technique is most effective if the designs follow a clear naming conventions for the port names.

The best average results are computed by the combination of the name matching heuristic and the extended name matching heuristic both using the Levenshtein-distance together with the pure greedy connection strategy. This combination yields better results than Emacs Verilog-Mode in seven of eight design. This includes a design which is optimized towards Emacs Verilog-Mode. Further, for a design which Emacs Verilog-Mode is not able to create any connection our approach is able to create a perfect set of connections.

References

[1] W. Snyder, "Verilog-mode: Reducing the veri-tedium," in *Synopsys Users Group Conference*, San Jose, 2001.

[2] "MKTREE," accesss date: 12.09.2013. [Online]. Available: http://www.angelfire.com/biz/mktree/

[3] C. Neely, G. Brebner, and W. Shang, "ShapeUp: A High-Level Design Approach to Simplify Module Interconnection on FPGAs," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 141–148.

[4] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage." in *Survey Research Methods Section, American Statistical Association*, 1990, pp. 354–359.

[5] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.