ASGtools

Norman Kluge
Hasso-Plattner-Institut
University of Potsdam, Germany
norman.kluge@hpi.de

Ralf Wollowski
Hasso-Plattner-Institut
University of Potsdam, Germany
ralf.wollowski@hpi.de

Cornelius Bock
Hasso-Plattner-Institut
University of Potsdam, Germany
cornelius.bock@student.hpi.de

I. Introduction

BALSA [1] provides a design flow where asynchronous circuits are created from high-level specifications, but the syntax-driven translation often results in performance overhead. To improve this, we exploit the fact that bundled-data circuits can be divided into data and control path. Hence, tailored optimisation techniques can be applied to both paths separately. We have introduced such an approach in [2], [3]. In this abstract we present a tool suite implementing this flow.

II. The tools

Fig. 1 shows the overall design system. Just like the original BALSA flow, we start with a Balsa program which is transformed into a Breeze netlist with the Balsa compiler. A Breeze netlist specifies a network of handshake (HS)-components e.g. communicating via the 4-phase bundled data HS-protocol. This network can be visualised with ASGbreezeGui. Afterwards, a Verilog netlist is generated from this Breeze file. In the original design flow this is done by BALSA-NETLIST – we introduce ASGresyn, a new (resynthesis) tool which will do this step in an optimised manner. The Verilog netlist, its derived delay file and a testbench for the design (e.g. generated by ASGtestGen) are given to a simulator to perform tests.

A. ASGresyn

ASGresyn implements the *resyn*thesis procedure as described in [3], i.e. it generates an optimised Verilog netlist from a Breeze specification. Fig. 2 shows the internal architecture of this tool. The central operation is the splitting of the HS-components into control and data path. The Data Path Generator and the Merging Agent are integrated into the tool itself. All other agents are seperate programs: DesiJ, ASGlogic, and breeze2stg are developed in-house, while PCOMP [4], PETRIFY [5], and PUNF/MPSAT [6] are third party. ASGresyn orchestrates all programs by calling them depending on the configuration and maintaining the generated data.

B. DesiJ and breeze2stg

DesiJ decomposes large STGs into smaller ones to tackle state space explosion [7] using an adjusted STG decomposition algorithm [8]. Moreover, a part of DesiJ called breeze2stg is responsible for generating STGs for the control part of

This research was partly supported by DFG-project 'Optacon' VO 615/10-1 and WO 814/3-1.

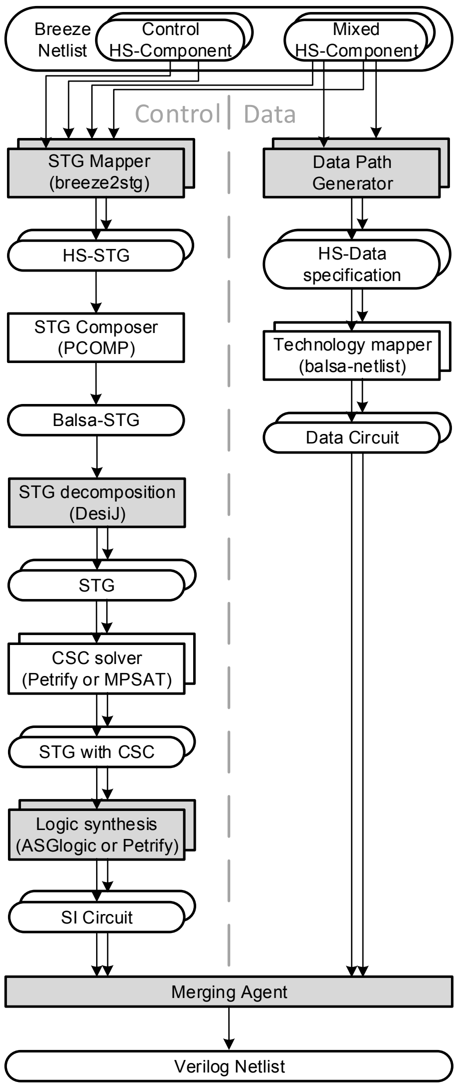

Fig. 2. ASGresyn architecture

a Breeze description [2]. For a more detailed description (excluding the latest features) on DesiJ see [9].

C. ASGlogic

ASGlogic is a *logic* synthesiser generating a Verilog implementation from an STG specification. The implementation of the state graph construction and equation derivation are based on the ideas from [10]. For technology mapping, we

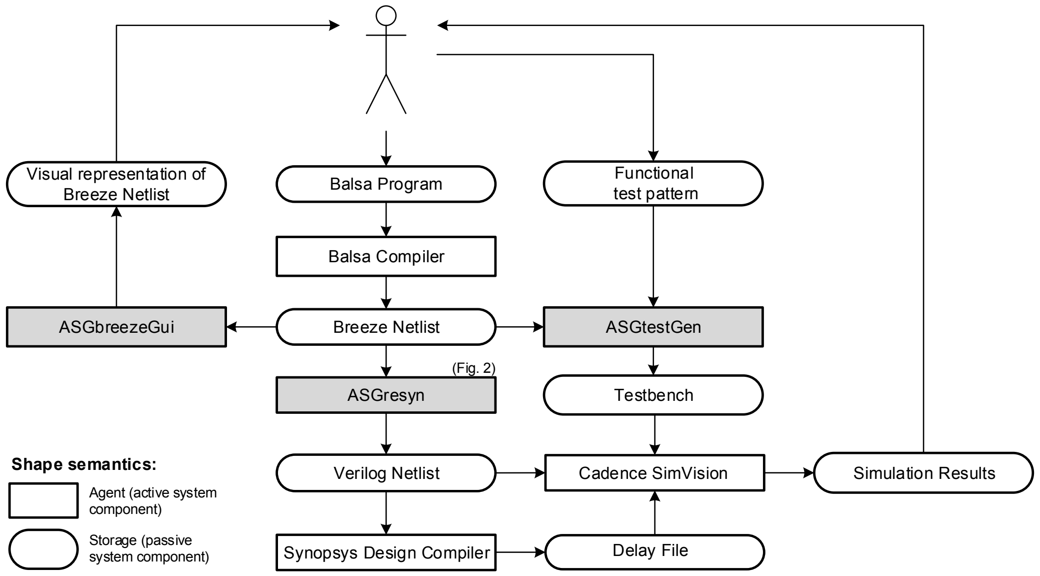

Fig. 1. Tool landscape

implemented methods presented in [11]. However, we are still working on enhancing the technology decomposition algorithm. As a main feature, ASGlogic provides a proper reset insertion mechanism. Note that ESPRESSO [12] is used for logic minimisation and CSC solving is delegated to PETRIFY or, alternatively, PUNF/MPSAT.

D. *ASGtestGen*

ASGTestGen produces test patterns from a Breeze netlist to verify implementations of it. It generates stimuli targeting maximal test coverage. Thus even unfamiliar Balsa programs can be verified. This work is still in progress.

E. *ASGbreezeGui*

ASGbreezeGui provides a graphical representation of a Breeze netlist with support for hierarchical designs.

III. Download and install

All ASGtools are open source and can be downloaded on GitHub: https://github.com/hpiasg. All required external tools are already included in the download and there is no need to manipulate environment variables manually. The tools are running on a Java VM and most of them are using Maven as build tool.

Acknowledgements

We thank the IHP GmbH (Frankfurt/Oder) for providing the design environment and the design kits, in particular Milos Krstic and Steffen Zeidler for their helpful advice. Best thanks to Walter Vogler and Stanislavs Golubcovs for fruitful cooperation making this work possible.

References

[1] A. Bardsley and D. A. Edwards, “The Balsa asynchronous circuit synthesis system,” in *Forum on Design Languages*, Sep. 2000.

[2] S. Golubcovs, W. Vogler, and N. Kluge, “STG-based resynthesis for balsa circuits,” in *Proceedings of the 2013 13th International Conference on Application of Concurrency to System Design*, ser. ACSD ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 140–149.

[3] N. Kluge and R. Wollowski, “Optimising bundled-data balsa circuits,” to appear in Asynchronous Circuits and Systems (ASYNC), 2016 22nd IEEE International Symposium on, May 2016.

[4] A. Alekseyev, V. Khomenko, A. Mokhov, D. Wist, and A. Yakovlev, “Improved parallel composition of labelled Petri nets,” in *Proceedings of the 2011 Eleventh International Conference on Application of Concurrency to System Design*, ser. ACSD ’11, 2011, pp. 131–140.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers,” *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.

[6] V. Khomenko, M. Koutny, and A. Yakovlev, “Detecting state coding conflicts in stg unfoldings using sat,” in *Application of Concurrency to System Design, 2003. Proceedings. Third International Conference on*, June 2003, pp. 51–60.

[7] W. Vogler and R. Wollowski, *Concurrency and Hardware Design: Advances in Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, ch. Decomposition in Asynchronous Circuit Design, pp. 152–190.

[8] S. Golubcovs and W. Vogler, “Decomposing Balsa-STGs (working notes),” Institute of Computer Science, University of Augsburg, Tech. Rep., 2014.

[9] M. Schaefer, D. Wist, and R. Wollowski, “Desij–enabling decomposition-based synthesis of complex asynchronous controllers,” in *Application of Concurrency to System Design, 2009. ACSD ’09. Ninth International Conference on*, July 2009, pp. 186–190.

[10] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic synthesis of asynchronous controllers and interfaces*, ser. Advanced Microelectronics. Springer-Verlag, 2002.

[11] P. Siegel and G. De Micheli, “Decomposition methods for library binding of speed-independent asynchronous designs,” in *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD ’94, 1994, pp. 558–565.

[12] R. L. Rudell, “Multiple-valued logic minimization for pla synthesis,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M86/65, 1986. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1986/734.html