DESI: a Tool for Decomposing Signal Transition Graphs *

B.Kangsah[1], W. Vogler[2], R. Wollowski[3], and J. Beister[1]
[1]*FB Elektro- und Informationstechnik, Technische Universität Kaiserlautern,*
{kangsah,beister}@rhrk.uni-kl.de
[2]*Institut für Informatik, Universität Augsburg, vogler@informatik.uni-augsburg.de*
[3]*Hasso-Plattner-Institut, ralf.wollowski@hpi.uni-potsdam.de*

1. Introduction

Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuit behaviour. As a first step in the indirect synthesis of a circuit corresponding to a given STG N, one usually constructs the reachability graph (e.g. when using the tool *petrify* [3]) or the step graph (e.g. using CASCADE [1]). A serious problem – state explosion – may occur when constructing such a graph: the number r of reachable states (markings) may become too large to be handled due to insufficient storage space or too long CPU times. To avoid state explosion, one could try to decompose the STG N into components C_i (and thus, the circuit into modules). The reachability graphs of the C_i, taken together, can be much smaller than r since r might be the product of their sizes. Even if this is not achieved, decomposition can reduce design effort and save circuit area. Where N may have to be synthesized by heuristic methods, its components C_i may even be handled by exact methods yielding optimal results. Decomposition can also be useful aside from size considerations: there are examples where N cannot be handled by a certain synthesis method, while its C_i can . It may also be possible to extract library elements; this is particularly valuable for arbiters, which are difficult to design.

The decomposition algorithm [7] offers significant improvements over others, e.g. [2], [5]. In particular, there is no restriction to live and safe free-choice nets or to marked graphs, and labels are not required to occur only once. Also, a formal proof based on a formal correctness criterion has been given. Further improvement of [7] is given in [6], which allows more STGs for real applications to be decomposed, and gives better results (smaller components).

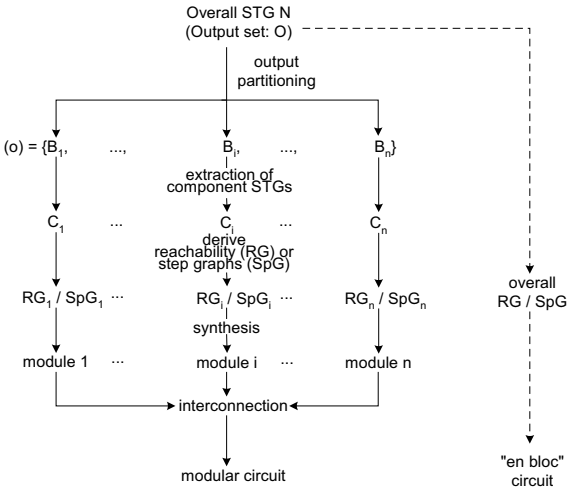

Figure 1. Modular design flow

2. DESI

DESI (**DE**composer of **SI**gnal Transition Graphs) is implemented based on the algorithm [6] [7] which starts with a given partition of the set of output variables: each C_i is responsible for one block of the partition. The C_is are then extracted from the STG by transition contraction and deletion of redundant places [1] [7], as well as deletion of loop-only and duplicate λ-transitions [6], care being taken to keep only the relevant input signals, which may be global inputs or outputs of other components. Step by step examples of STG decomposition can be seen in http://www.eit.uni-kl.de/beister/eng/projects/deco_examples/main_examples.html.

DESI is suggested to be applied in modular design. In our case, the modular design starts from an STG as specification. The STG is decomposed into components based on

* Work partially supported by DFG-project 'STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

1 Deletion of redundant places in DESI is restricted to deletion of loop-only and duplicate places.

the output partition. Each component is synthesized separately, then the resulting modules are interconnected to form a modular circuit. The overall design flow is shown in Fig. 1.

DESI is designed as part of CASCADE, which can forward results to other synthesis tools such as *petrify* and $3D$ [8]. Based on the modular design flow, DESI together with other tools completes the tool chain for a modular design (see Fig. 2).

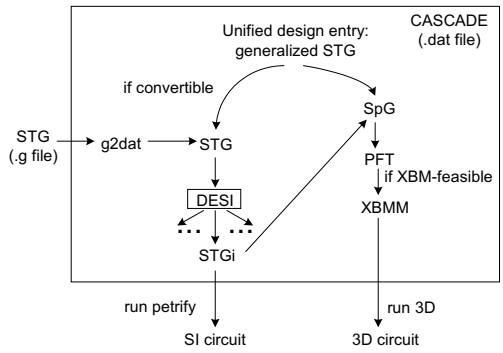

Figure 2. Modular design tool chain

DESI is an academic tool which may be downloaded from http://www.eit.uni-kl.de/beister/ger/projekte/projekte_wollo.html. Version 2 uses the algorithm from [7]. This version was presented in [4]. Version 3 is version 2 augmented by the algorithm from [6]. Also DESI with risky strategy (see [6] for more detail) included in these versions.

3. Experimental results

The examples in the benchmark table are taken from a collection of benchmark examples that circulate in the STG community. In the experiment, *petrify* is used to synthesize the components. For each example in the table, we have the number r of reachable states, the area a resulting from synthesis and the computation time τ needed for an Intel Xeon 2.2 GHz with 1 GB memory. The lower-case literals are used for the original specification; the upper-case literals are used for the sum of all components resulting from decomposition. T_d is the time taken by DESI for decomposition, T_p is the time taken by petrify for synthesizing all the components. In the arbiter example, the ME-element found by DESI could not be synthesized by petrify (neither could the original specification).

From the table, one could see that DESI gives the best results if r is large. We obtained fewer reachable states, less area, and less computation time. Even for small examples, we mostly need less computation time. DESI overhead is small in most cases compared to the computation

name	original			all components together					ratio (%)			
	r	a	τ	R	A	T	T_d	T_p	R/r	A/a	T/τ	T_d/T_p
mread	8932	67	231.18	409	59*	6.31	0.15	6.16	4.57	88.06	2.73	2.44
stg-blunno'	1241	54	401.35	248	36*	2.95	1.27	1.68	19.98	66.67	0.74	75.6
FIFO	832	49	140.88	126	41*	3.12	0.45	2.67	15.14	83.67	0.32	16.85
locked2'	168	23*	6.91	82	29*	2.36	0.39	1.97	48.81	126.09	34.15	19.8
(risky)				44	--	--	0.35	--	26.19	--	--	--
pe-send-ifc'	117	50	2.07	100	62*	4.82	0.43	4.39	85.47	124	232.85	9.79
mux2'	101	68*	255.42	93	109*	50.26	0.47	49.75	92.08	160	19.68	0.94
LL Arbiter'	64	--	--	82	^	--	0.1	--	128.13	--	--	--
post-office'	62	28	22.3	72	32*	17.72	0.49	17.23	116.13	114.29	79.46	2.84
(risky)				67	--	--	0.43	--	108.06	--	--	--
nak-pa	58	18	0.19	54	18	0.38	0.26	0.12	93.1	100	136.84	216.67
adfast	44	17	1,34	38	15*	0.91	0.15	0.76	86.36	88.24	67.91	19.74
adc-yak	44	15	1.29	39	16*	0.64	0.14	0.50	88.64	106.67	49.61	28
NEI Arbiter'	42	--	--	33	^	--	0.09	--	78.57	--	--	--
tsend-csm'	36	38	9.43	41	35*	1.4	0.24	1.16	113.89	92.11	14.85	20.69
vmecon'	24	19	1.72	27	22*	0.57	0.13	0.44	112.5	115.79	33.14	29.55

' could be handled only by improved algorithm; i.e. DESI version 3 or above
* there is module which is not speed independent (petrify synthesis with "slow environment")
^ ME-element plus synthesizable component(s)

Figure 3. Benchmark table

time needed for synthesis. With the risky strategy, we got smaller number of R, but with the penalty that the components could not be synthesized by petrify.

Acknowledgement
Special thanks to V. Khomenko for his comments about DESI. Also to I. Blunno and J.Carmona for the benchmark examples.

References

[1] J. Beister, G. Eckstein, and R. Wollowski. Cascade: a tool kernel supporting a comprehensive design method for asynchronous controllers. In M. Nielsen, editor, *Applications and Theory of Petri Nets 2000*, Lect. Notes Comp. Sci. 1825, 445–454. Springer, 2000.

[2] T.-A. Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *Proc. International Conf. Computer Design (ICCD)*, pages 220–223. IEEE Computer Society Press, 1987.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *XI Conference on Design of Integrated Circuits and Systems*, Barcelona, Nov. 1996.

[4] B. Kangsah, R. Wollowski, W. Vogler, and J. Beister. DESI: A tool for decomposing signal transition graphs. In *3rd ACiD-WG Workshop*, 2003.

[5] A. Kondratyev, M. Kishinevsky, and A. Taubin. Synthesis method in self-timed design. Decompositional approach. In *IEEE Int. Conf. VLSI and CAD*, pages 324–327, 1993.

[6] W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. In *Application of Concurrency to System Design 2005*, To appear.

[7] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella, A. Yakovlev, and G. Rozenberg, editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, pages 152–190. Springer, 2002.

[8] K. Y. Yun, D. L. Dill, and S. M. Nowick. Synthesis of 3D asynchronous state machines. In *Proc. International Conf. Computer Design (ICCD)*, pages 346–350. IEEE Computer Society Press, Oct. 1992.