A Logic Sharing Synthesis Tool for Mutually Exclusive Applications

Alp Kilic, Zied Marrakchi, Matthieu Tuna, Habib Mehrez

A Logic Sharing Synthesis Tool for Mutually Exclusive Applications

Alp Kilic, Zied Marrakchi[1], Matthieu Tuna[1] and Habib Mehrez
University of Pierre and Marie Curie, Paris VI, LIP6 Laboratory, France
[1] *Flexras Technologies, France*
{alp.kilic, habib.mehrez}@lip6.fr, {zied.marrakchi, matthieu.tuna}@flexras.com

***Abstract*— Multiple Context ASIC (mASIC) is a circuit grouping a set of designs (applications) which operates at mutually exclusive times. In this paper we propose to take this particularity into account when we run logic synthesis. The idea is to maximize logic resources sharing between designs to reduce the total resulting area. Once used on mASIC for a set of 5 benchmark designs, our synthesizing technique reduces area by 28% compared to the sum of the 5 individual ASIC areas.**

I. Introduction

Today electronic devices contain more and more features due to emergence of new embedded applications like telecom, digital television, automative and multimedia applications. These applications require on one hand hardware architectures with higher performances, but on the other hand the same architectures should be as small as possible and meet very tight power consumption constraints.

The interesting point which comes with feature-rich platforms is that lots of the features cannot be executed at the same time. Some of the applications are used in mutually exclusive way. It means they have no common outcomes. For exemple in mobile phones, we can not listen to the music while talking on the phone.

These applications can be implemented on three different hardware platforms:

- **CPU:** Applications can be implemented in software and can be executed in one CPU. While the resource sharing problem is met, these applications are usually computation-intensive, which prevents them from being executed by a general-purpose processor.
- **FPGA:** Moving from one application to another comes with a reconfiguration of the FPGA by loading the corresponding bitstream. It means that we can exploit efficiently resource sharing since our applications do not run at the same time. Otherwise, in this case we do not exploit the fact that we know the limited applications in advance and that we do not need an unlimited flexibility. That is why, while the resource sharing is met, an FPGA cannot compete with a dedicated hardware (ASIC) in terms of area, performance and power consumption [7].
- **ASIC:** Usually the preferred way to implement computation-intensive applications. As we pointed out, in the context of mutually-exclusive applications there is a silicon waste. To overcome this problem we tried to encapsulate (instantiate with a mux at the inputs and demux at the outputs) under the same top and connected to the same interface. The obtained circuit is synthesized and optimized using common tools (Cadence RTL compiler [4] or Synopsys Design Compiler [13]). Based on the experimentation shown in figure 8, it turns out that existing tools are not efficient to recognize the "mutually exclusive" pattern and do not optimize the area, based on resource sharing from different applications. We obtain a circuit with the sum of areas of the sub-circuits synthesized separately. We even used the same sub-circuit by instantiating it three times under the same top but we obtain a circuit with three times the area of the sub-circuit.

Actually there are no efficient tools that can overcome this kind of sharing problem and exploit the fact that applications can be mutually exclusives. Thus, we propose a logic sharing synthesis tool for mutually exclusive applications. The main idea is to achieve the best level of optimization in terms of area. We take the advantage of the possibility of resource sharing between applications knowing that resources cannot

[1]This work is partially funded by the ANR project ASTECAS.

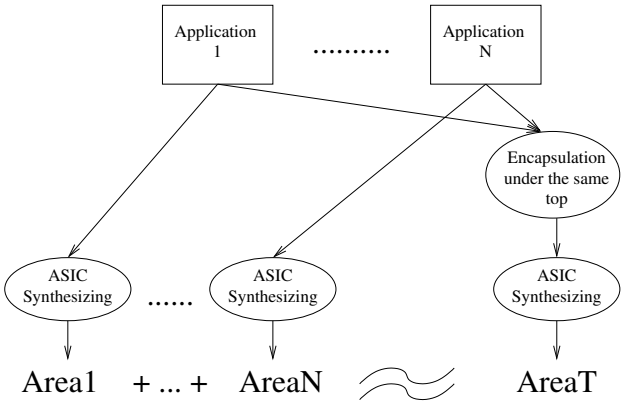


Fig. 1. Multiple-context ASIC synthesizing methods

be used at the same time. The new resulting architecture is a circuit which can run one application at a time. We call this new architecture a multi-Application Specific Integrated Circuit: mASIC.

The reminder of the paper is organized as follows. Section II decribes two different architectures with similar objectives and different approaches. In section III we propose our solution for the problem of the resource sharing and we introduce the Multi context ASIC concept. Section IV presents various experimental results using a set of applications. Finally section V presents the conclusion and future work.

II. Prior Work

There are different trade-offs between FPGAs and ASICs. They are explored by several research groups in [14] [12] [6]. FPGA vendors have also solutions to migrate FPGA-based applications to structured ASIC implementation. Altera provides Hard-Copy [2] and Xlinix provides EasyPath [15].

There are two different architectures that we are interested to in this work: configurable ASIC cores (cASIC) [5] and Application Specific FPGA(ASIF) [10]. Both architectures are desgined to execute a given set of applications at mutually exclusive times. One of the major difference between cASIC and ASIF is in the approach used to optimize their routing networks. As shown in figure 2, The starting point of cASIC is ASIC. It is generated using a constructive buttom-up "insertion" approach; flexibility is inserted through the addition of multiplexers and demultiplexers. On the other hand, an ASIF is generated using an iterative top-down "removal" technique; different circuits are mapped onto an FPGA, and flexibility is removed from the FPGA to support only the given set of circuits and to reduce its area.

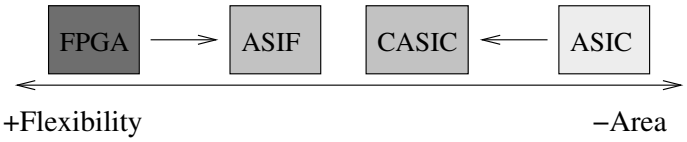


Fig. 2. ASIF and CASIC: Different start points but same objective

cASICs are intended as accelerator in domain-specific systems-on-a-chip. But they are not designed to replace the entire ASIC-only chip. cASICs implement only data-path circuits and thus supports full-word blocks only. Since the set of circuits supported by a cASIC are limited, cASICs are significantly smaller than an FPGA implementation. As hardware resources are shared between different netlists, cASICs are 2.2x smaller than a standart cell implementation of individual circuits.

The concept of an ASIF is similar to cASIC. It is an FPGA with reduced flexibility that can implement a set of application circuits which will operate at mutually exclusive times. These circuits are efficiently placed and routed on an FPGA to minimize total routing switches required by the architecture. Later all unused routing switches are removed from the FPGA to generate an ASIF. The disadvantage of ASIF is that it does not ensure unlimited flexibility although being a configurable device. The remaining flexibility has area/performance overhead that can be exploited in very limited cases.

Figure 3 illustrates the ASIF generation concept. When an FPGA-based product is in the final phase of its development cycle, and if the set of circuits to be mapped on the FPGA are known, it can be reduced for all the given set of circuits. An ASIF can yield considerable area and performance gains to an FPGA-based product by reducing it to a much smaller multiplexed circuit. Execution of different application circuits can be switched by loading their respective bitstream on ASIF.

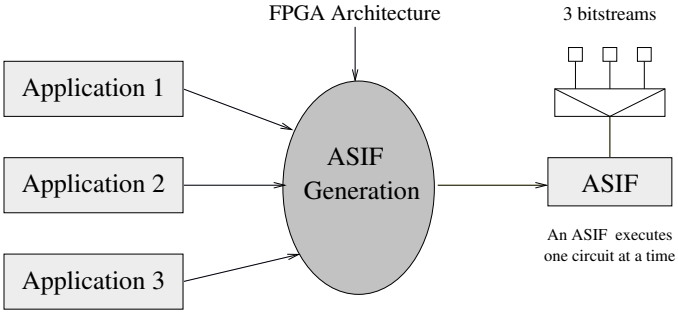


Fig. 3. An illustration of ASIF generation concept.

The biggest advantage of ASIF is to favor the logical sharing

between netlists. According to [10], ASIF uses efficient algorithms to place and route the set of given netlists. Efficient placement tries to place the instances of different netlists in such a way that minimum routing switches are required in an FPGA. Consequently, efficient routing increases the probability to connect the driver and receiver instances of these netlists by using the same routing wires. Also the efficient wire sharing favors different netlists to route their nets on an FPGA with maximum common routing paths and tries to minimize the total routing switches required in an ASIF. It is one of the most important process in ASIF generation because the main idea of ASIF is to remove unused switches after all netlists are efficiently routed on FPGA. Our work is based on ASIF and we are taking advantage of the existing resource sharing thanks to these two algorithms.

III. Proposed Solution : multi context ASIC

We propose to combine multiple strategies to achieve the best level of mASIC optimization in terms of area. We take advantage of the possibility of resource sharing as done in FPGA and of predefined multiple functions as in ASIC. As shown in figure 4, the flow is done in three steps: ASIF Generation to get a circuit placed and routed derived from a FPGA with reduced flexibility, a customization process to remove the remaining flexibility and finally ASIC synthesizing with a commercial tool. Finally, compared to a simple ASIC synthesis flow (right branch in figure 4), multiple ASIC flow proposes, in addition, logical sharing to reduce circuit area.

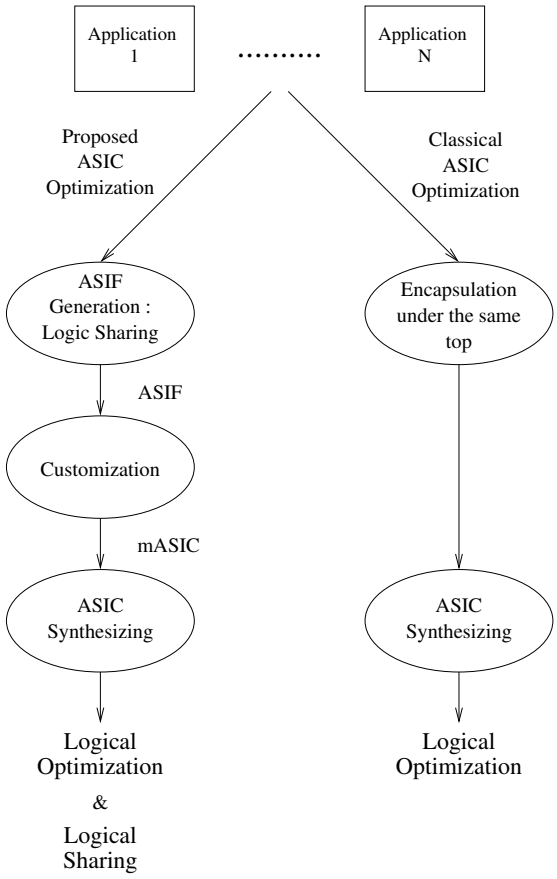


Fig. 4. Multiple context ASIC synthesizing method.

A. ASIF Generation Flow

First, a software flow transforms behavioural descriptions in Verilog or VHDL to their respective netlists, for mapping to the heterogeneous FPGA. These HDL descriptions are first synthesized with Cadence RTL Compiler to obtain a structural netlist composed of standard cell library instances and Hard Block (HB) instances in verilog. "flxVeriBlif" tool converts verilog netlists with hard blocks to BLIF [1] file format. Later "fixblif" removes all instances of hard blocks and passes the remaining netlists to SIS [11] for technology mapping (synthesis into Look-Up Table format). Dependence between HBs and the remaining netlist is preserved by adding temporary input and output pins to the main netlist. After SIS, and later after packing and conversion of the netlist to NET format through T-VPACK [8], "fixnet" adds all the removed HBs into netlist. It also removes all previously added temporary inputs and outputs. The generated netlits (in .NET format) include CLBs, HBs and IO (Inputs and Outputs) instances which are interconnected through signals called NETS. Then these netlists are placed and routed on the target FPGA architecture defined with a enough logic blocks number to handle the given set of netlists. The placer uses the simulated annealing algortihm [3] to place CLBs/HBs/IO instances of a netlist on their respective blocks of the FPGA. The router uses a pathfinder algorithm [9] to route the netlits using FPGA routing resources. As explained in section II, ASIF uses a modified version of these algorithms : Efficient Placement and Efficient Wire Sharing (described in [10]) to favor the logical sharing between netlists. The ASIF generation flow is shown in figure 5

Finally, the ASIF netlist is generated by removing all routing resources of the FPGA architecture that remain unused by the netlists. Once generated, the ASIF netlist is customized by removing the remaining flexibility to obtain a smaller circuit.

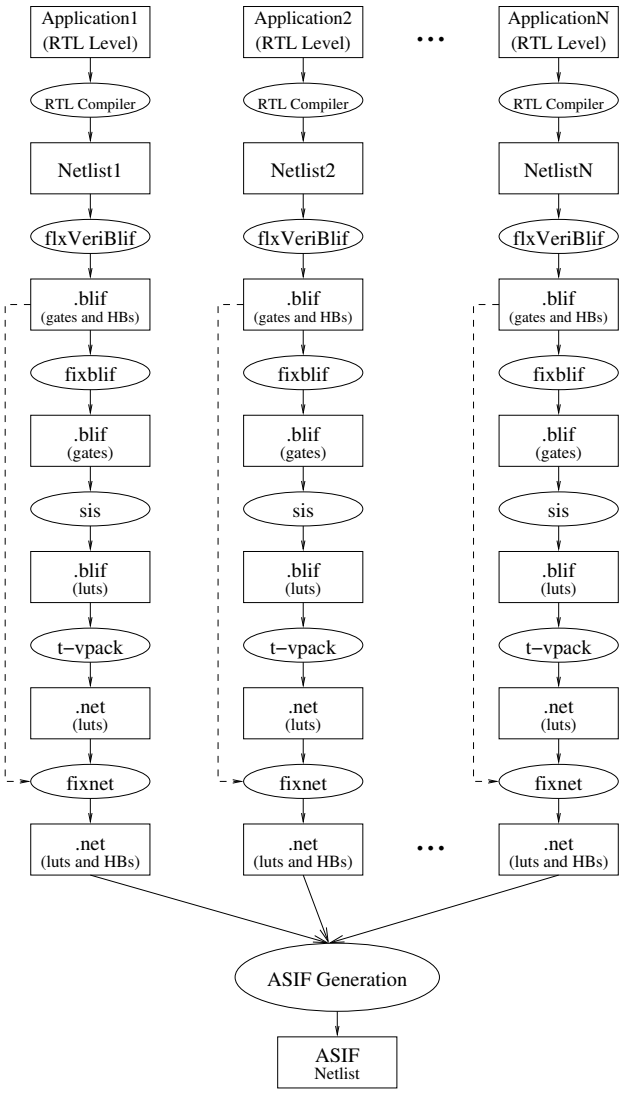


Fig. 5. ASIF generation flow.

B. Customazation method

In a traditional FPGA architecture, logic blocks resources occupy only 10 to 20% area of an FPGA. The remaining 80 to 90% of FPGA area is occupied by routing resources. Since an ASIF is generated by removing unused routing resources of an FPGA, the logic area percentage in an ASIF is higher than an FPGA. It means that the optimization of logic resources may bring major area advantages.

ASIF uses SRAM cells, like most FPGAs, to control pass transistors on routing connections, multiplexers and LUTs on CLBs. It contains limited flexibility. But this flexibility cannot really be exploited since configuration tools cannot guaranty it. Different set of application netlists, mapped on an ASIF, program the SRAM bits of a LUT differently. In this step we propose to replace all the remaining memory points to optimize both logic and routing resources. In figure 6 we show how to transform an SRAM into a multiplexer for 3 different netlists. As we know the bitsreams of all netlists in advance, we can replace every memory points by a multiplexer that takes hard coded bitstream as an input. Then we can choose which netlist to use by varying the "select_netlist" signal.

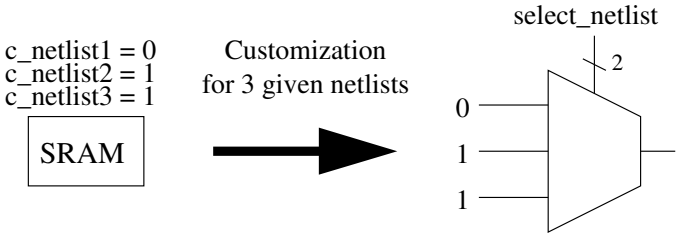


Fig. 6. Transformation of an SRAM into a multiplexer.

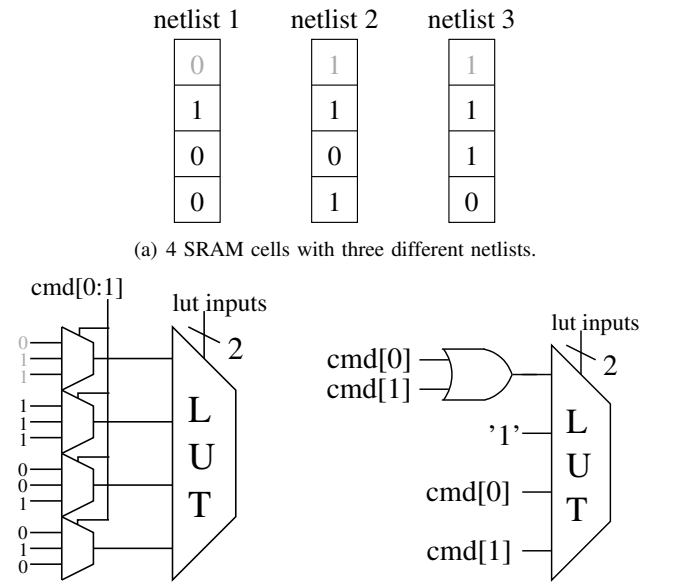
After the customization stage, using a common commercial synthesis tool like Cadence RTL Compiler or Synopsys Design Compiler, we map the mASIC on a given logic gate library. Through this process, these commercial tools can perform logic optimizations on multiplexors inserted in the customization stage. The main goal is to help the ASIC Synthesis tool to get the full advantage of its optimizations by shaping the input circuit.

In figure 7 we show an exmaple of a circuit containing 3 sub-circuits. In 7(a) we have three vectors of different bitstreams mapped on 4 SRAM cells of a LUT-2. Each vector corresponds to a netlist. As it can be seen in 7(b), we replace memory points with multiplexers with constants as inputs and controlled by an external input (cmd[0:1]) depending on the circuit to run. Those inputs are carried into the circuit interface and are used to select the application to run. Later the synthesis tool optimizes these multiplexers and delivers a smaller circuit by propagating constants (figure 7(c)).

IV. Results and Analysis

	#CLB	#ADD	#MULT
Netlist-1	2087	24	25
Netlist-2	921	12	13
Netlist-3	612	7	8
Netlist-4	145	3	4
Netlist-5	1342	18	0
Maximum	2087	24	25

TABLE I
5 NETLISTS USED IN EXPERIMENTS



(a) 4 SRAM cells with three different netlists.

(b) A 2-input LUT after customization.

(c) A 2-input LUT after synthesis.

Fig. 7. An mASIC optimization exemple for three netlists

A set of 5 circuits using CLBs (LUT-3), adders and multipliers are presented in Table I. These circuits are synthesized using the RTL Compiler [4]. To map netlists onto an ASIF, the synthesized netlists are handled through the software flow (described in Section 3.A) and converted into .NET format. The ASIF generation techniques are applied on these netlists to obtain an ASIF. Later ASIF is customized for given netlists in order to get the final netlist design "mASIC". Finally, ASIF and mASIC is compared with an ASIC obtained by the encapsulation of these 5 circuits under the same top (figure 8).

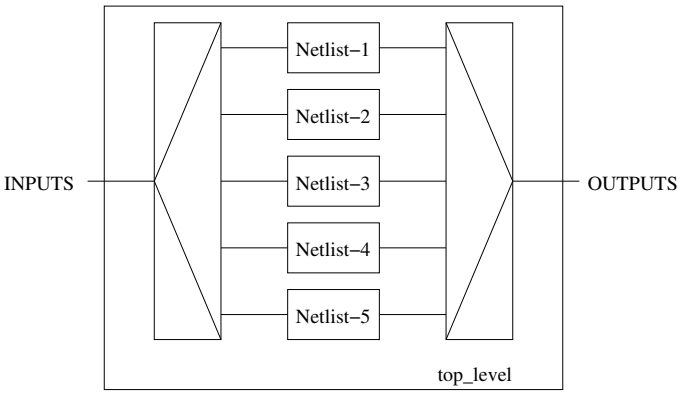


Fig. 8. Classical ASIC synthesis method.

Figure 9 compares ASIF, mASIC and ASIC with various number of applications. The X-axis presents the number of netlists; where 1 means only Netlist-1 is used; 2 means Netlist-1 and Netlist-2 are used, and so on. The Y-axis presents the total area in μm^2. In a classical ASIC sythesis method, RTL compiler cannot share resources between netlists. So the total resources in ASIC is equal to the sum of the resources of the sub-circuits. In ASIF, with the efficient placement and efficient wire sharing algorithms, some of the resources are shared. However it becomes bigger than an ASIC because the idea of being flexible is introduced by CLBs and configurable routing blocks. Later, it is customized to obtain an mASIC which is %96 smaller than ASIF. mASIC is also 28% smaller than ASIC in terms of total area thanks to both the customization and resource sharing.

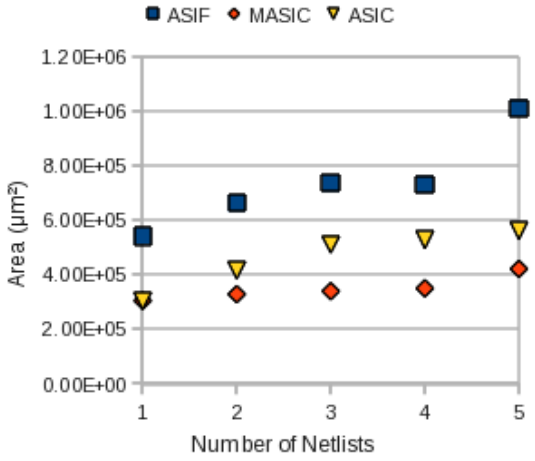


Fig. 9. Area comparaison between ASIC,mASIC and ASIF

V. Conclusion and Future Work

This paper has presented a new concept about mutli context ASIC (mASIC). An mASIC is an attempt to explore a new technique for asic synthesizing. It is a top-down approach starting from an FPGA to reach an ASIC which contains multiple netlists. First, it takes advantage of the possibility of resource sharing as done in FPGA by passing the netlists through the FPGA flow with efficient resource sharing algorithms. Then all the unused blocks of the FPGA are removed to obtain an ASIF. Later, the bitstreams of each netlist are hard coded by adding multiplexers and constants instead. At the end we obtain an mASIC that can execute the given netlists exclusively. Results illustrate that an mASIC for 5 netlists is %96 smaller than ASIF and %28 smaller than ASIC. In comparison to ASIF, mASIC achieves area advantage through the elimination of the flexibility since the multiple applications are known in advance. When compared to ASIC, mASIC can

share resources between netlists and remains smaller than it in terms of total circuit area.

There can be different directions for future work on mASICs. Firstly, current mASIC generation flow needs to be upgraded to support automatic validation in each step. This will allow to have other benchmarks more easily. Secondly, the bit values in LUTs can be reordered to get more similarities in input netlists. This will increase the number of constants and decrease the total area.

REFERENCES

[1] Berkeley logic interchange format (blif), 1996.

[2] Altera. http://www.altera.com/.

[3] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[4] Cadence. http://www.cadence.com/.

[5] Katherine Compton and Scott Hauck. Automatic design of area-efficient configurable asic cores. *IEEE Trans. Comput.*, 56:662–672, May 2007.

[6] easic. http://www.easic.com/.

[7] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 21–30, New York, NY, USA, 2006. ACM.

[8] Alexander (Sandy) Marquardt, Vaughn Betz, and Jonathan Rose. Using cluster-based logic blocks and timing-driven packing to improve fpga speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA '99, pages 37–46, New York, NY, USA, 1999. ACM.

[9] Larry Mcmurchie and Carl Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. pages 111–117, 1995.

[10] Husain Parvez, Zied Marrakchi, Alp Kilic, and Habib Mehrez. Application-specific fpga using heterogeneous logic blocks. *ACM Trans. Reconfigurable Technol. Syst.*, 4:24:1–24:14, August 2011.

[11] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

[12] Deepak D. Sherlekar. Design considerations for regular fabrics. In *Proceedings of the 2004 international symposium on Physical design*, ISPD '04, pages 97–102, New York, NY, USA, 2004. ACM.

[13] Synposys. http://www.synposys.com/.

[14] Kun-Cheng Wu and Yu-Wen Tsai. Structured asic, evolution or revolution? In *Proceedings of the 2004 international symposium on Physical design*, ISPD '04, pages 103–106, New York, NY, USA, 2004. ACM.

[15] Xilinx. http://www.xilinx.com/.