Monarch: A Platform for Logic Optimization using ATPG/Diagnosis-based Design Rewiring

J. Brandon Liu Magdy S. Abadir
SPS, Motorola
Austin, TX
{Brandon.Liu, M.Abadir}@motorola.com

Robert Chang Andreas Veneris
Dept. of ECE and CS
University of Toronto
{rchang, veneris}@eecg.utoronto.ca

Abstract

In a typical VLSI design cycle, technology-dependent logic optimization may occur after the physical synthesis to satisfy various design constraints in area, power, timing, and testability. Recently, it is proposed in [7] an ATPG-based design rewiring methodology that achieves significant performance gains in benchmark circuits that are already optimized by formal techniques. This case study describes an application of this technique as a logic optimization platform for Motorola high-performance designs: *Monarch*. The flow, which consists of EDA vendor tools and in-house software, allows the design error diagnosis and correction techniques of [7] to be applied to gate-level modules in high-performance cores. Experiments in timing optimization show that *Monarch* can improve the slack of a module that has been already optimized by tools from commercial EDA vendors.

1. Introduction

In logic optimization, the gate-level netlist from high-level synthesis tools is locally re-synthesized to achieve certain design constraints, which are often a combination of area, power, timing and testability. Technology-dependent logic optimization operates on designs that have been mapped to a technology library characterized at physical level. Automatic test pattern generation (ATPG)-based optimization techniques have demonstrated their effectiveness in area minimization, power reduction, performance optimization, routing, and design for testability [1,3,4,5].

ATPG/Diagnosis-based [7] design rewiring (ADDR) combines some existing techniques in design error diagnosis and correction and ATPG, and complements them with new theorems and heuristics. A general flow for ADDR is given in Figure 1. It operates on a flattened gate-level combinational or full-scan sequential netlist. First, an arbitrary design error is injected, which is commonly the removal of a wire. Then using an ATPG tool, a set of test vectors are generated which can detect the error. The third step is to use a simulation-based design error diagnosis and correction algorithm such as the one in [6], which returns a set of corrections for the vectors used. Lastly, these corrections are formally verified against the original netlist. Multiple corrections may exist due to the redundancies in the design and the method takes advantage of such corrections to improve performance. The complete ADDR process may be repeated multiple times over the newly generated netlist. In theory, ADDR allows unlimited type and amount of structural logic transformations in the circuit [7].

ADDR can serve as the core engine in a logic optimization flow by making technology-dependent choices during error injection and netlist selection (the highlighted blocks in Figure 1). Given an optimization constraint, the selection of error and correction can greedily bring the design closer to satisfying the constraint. This case study describes

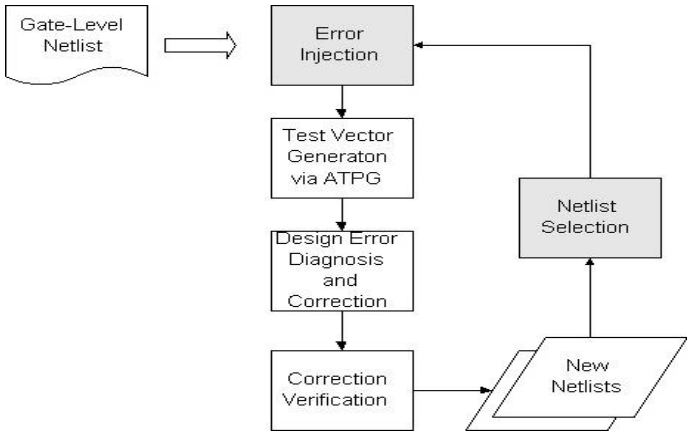

Figure 1 General flow for ADDR.

one such platform developed at Motorola Somerset design center using vendor and in-house tools: *Monarch*. It allows ADDR-based logic optimization, which was only available to non-hierarchical public-domain benchmark designs previously, to run on hierarchical gate-level modules in high-performance microprocessor cores. Experiments on timing optimization show improvement over that obtained by vendor tools.

The rest of the paper is organized as follows. The next section outlines the implementation of *Monarch*. Section 3 describes the use of *Monarch* in a typical physical synthesis flow. Experiment results are given in Section 4 and conclusions in Section 5.

2. Implementation

Monarch implements all the theorems and heuristics summarized in [6]. It leverages the proven correctness and efficiency of vendor tools by using them whenever possible. Its core diagnosis and correction engine, which is named *Chrysalis*, is developed in-house. The schematic flow of *Monarch* is presented in Figure 2. Here, it is tailored to timing optimization.

Monarch accepts a hierarchical gate-level combinational or full-scan sequential verilog netlist. Physical characterizations such as timing, power and placement are accessible in standard libraries. The design first undergoes static timing analysis. The critical path is identified for each clock domain and the flow continues if there is any negative slack (Slack is the difference between the actual and expected signal arrival time. A negative slack means a violation of some timing requirement.).

Monarch iteratively tries to shorten the critical path by cutting it at various places (i.e. removing a wire from the path) and find, if it exists, an alternative correction that results in a greater slack. A new verilog netlist is prepared with the injected error by using a netlister. Test vectors are then generated with an ATPG tool. Instead of generating specific vectors for the injected error, a vector set with 100% stuck-at fault coverage is generated for the original (error-free) design. It is shown in [7] that such a vector set prunes efficiently false corrections by simulation.

The verilog testbench file from the ATPG tool is suitably modified to contain a number of Verilog PLI (IEEE standard 1364-2001) calls, which serves as an interface between any commercial verilog simulator and our diagnosis and correction engine: *Chrysalis*. The former simulates the testbench while the latter extracts the circuit structure and

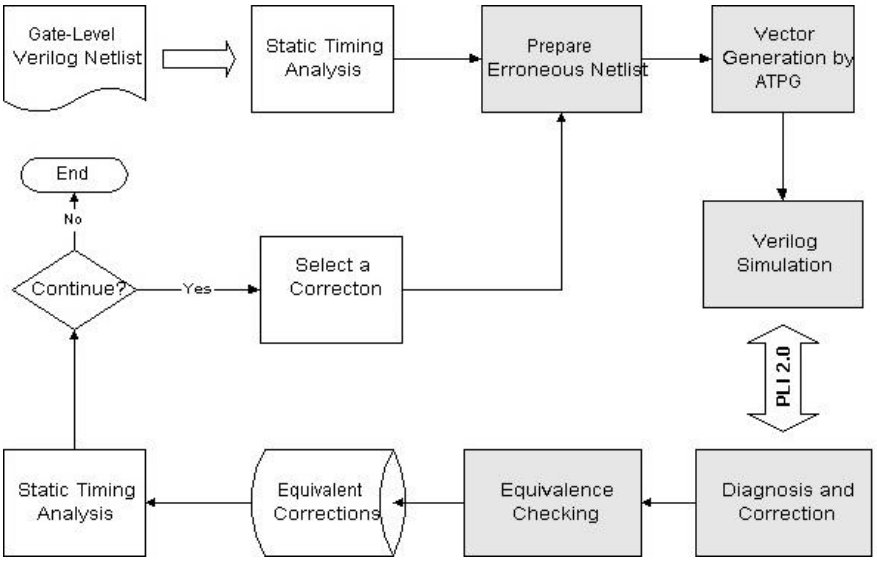

Figure 2 Detailed implementation of the *Monarch* flow for timing.

simulation results through the PLI interface, and proposes alternative corrections. *Chrysalis* augments the techniques in [6] by an X-simulation-based method for capturing sensitized paths described in [2] to efficiently prune the wires returned by diagnosis.

Corrections delivered to formal verification have already passed two stages of simulation in *Chrysalis*. First, a zero-gate-delay parallel-vector simulator verifies the combinational portion of the design. This quickly eliminates the majority of the false corrections. If their number is still high, the remaining corrections undergoes full error simulation with the help of the commercial simulator and PLI calls. The passing corrections are verified for its logic equivalence against the original netlist with a formal equivalence checker.

Static timing analysis is run on all the equivalent corrections to extract the new critical path. The flow can be continued greedily by preparing a netlist with the error and correction that have the highest slack and repeat the steps above.

The highlighted blocks in Figure 2 constitute the ADDR flow, which can be used as the core component for optimization flows targeting other constraints such as power and routability. In the current implementation, the diagnosis and correction block makes use of arrival and departure information from timing analysis to intelligently look for corrections that can more likely increase the slack. This approach improves its efficiency by eliminating unpromising corrections early. A similar approach can be taken in *Monarch* for other optimization settings.

3. Usage

A typical physical synthesis engine accepts the RTL description of a design and timing constraints among others, and goes through iterative steps of gate synthesis, placement optimization, sizing optimization, legalization, etc. to produce a technology-mapped gate-level netlist and its physical-level information. The netlist is re-timed with the wire delay extracted from layout. Manual hacking of the netlist often ensues if the constraints are violated.

Figure 3 depicts the use of *Monarch* potentially to replace the manual effort. It takes the technology-mapped netlist

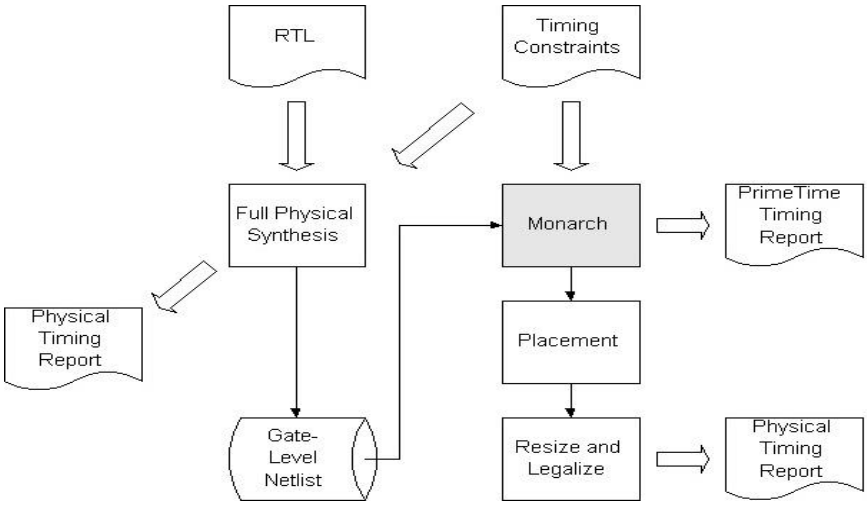

Figure 3 Monarch in a physical synthesis flow.

and the extracted wire delay (not shown), and improves the worst slack with its *Chrysalis* engine. The resultant netlist goes through an incremental physical synthesis flow, which consists of placement, resizing and legalization, to produce the final netlist and its physical level data. A designer can then verify that the changes are acceptable and take further steps if the timing characteristics are still not satisfactory.

4. Experiment

We ran the *Monarch* flow for timing on five full-scan sequential modules of a high-performance microprocessor core. The experiment procedure is similar to that in Figure 3. Static timing analysis was performed with Synopsys PrimeTime™,[1] and Synopsys Physical Compiler™,[2] was used as the physical synthesis tool. *Chrysalis* was configured to perform rewiring alone: the three permitted operations were 1) remove a connection; 2) make a new connection; 3) library cell type change (e.g. 2-input nand to 3-input nand, 2-input nor to inverter).

Ckt	Size	Worst Slack		Slack Improv.
		Original	Monarch	
1	2481	-500.17	-462.99	7.6
2	1482	-10.46	-7.43	29
3	7841	-36.61	-20.85	44
4	4484	-428.42	-380.06	11
5	3621	-231.92	-220.02	5.1
Avg				**19.3**

Table 1 PrimeTime timing result.

Two sets of timing reports were obtained for comparison: from PrimeTime™,[1] after Monarch and from Physical Compiler™,[2]. Physical data such as core area, average wire length, etc. were also extracted and compared.

Table 1 summarizes the results from PrimeTime™,[1]. The column "size" is the number of combinational primitives in the circuit. The next two columns shows the worst slack before and after *Monarch*. The last column has the percentage improvement in slacks. *Monarch* introduced between 2 and 10 rewiring changes for each of these circuits. The last row of the table shows

[1] PrimeTime™ is a registered trademark of Synopsys, Inc.

[2] Physical Compiler™ is a registered trademark of Synopsys, Inc.

Ckt	Original		Monarch		Slack Improv (%)
	Average wire length	Slack	Average wire length	Slack	
1	154.3	-490.6	159.7	-459.0	8.4
2	182.4	-7.45	178.8	-6.50	12.8
3	222.7	-38.21	222.0	-27.43	28.2
4	236.1	-418.8	234.0	-376.1	10.2
5	139.0	-494.6	138.0	-474.2	4.1
Avg	**186.9**		**186.5**		**12.7**

Table 2 Timing result after physical synthesis.

the average slack improvement, which is 19.3 %.

Physical-level timing and layout information are summarized in Table 2. "Average wire length" refers to the average length of nets in the final layout. That the change in this length is miniscule suggests that the gain in slack does not come at the cost of significant routing difficulties. The die area, which is not given in the table, remains the same. Slack improvement presented here, 12.7 % on average, is more accurate because realistic wire delays are incorporated in the computation.

Although the improvement in slacks is not large enough to eliminate timing violation, we believe that *Monarch* is a new addition to the arsenal of tools for timing optimization. It differs from conventional techniques such as buffer insertion and gate resizing and complements them in achieving timing closure. With the incorporation of more error/correction models in the *Chrysalis* engine, a greater improvement is expected at a larger impact to the eventual physical characteristics. More experiments are needed to study the potential tradeoff.

5. Conclusion

*Monarc*h, a versatile platform for gate-level logic optimization, is presented in this paper. It uses the ADDR algorithm for rewiring and 154.3+consists of EDA vendor and in-house tools. When it is used on several modules in a high-performance microprocessor core, *Monarc*h improves their timing over that achieved by vendor tools.

Bibliography

1. Chang, S. C., K. T. Cheng, N. S. Woo and M. Marek-Sadowska, "Post-layout logic restructuring using alternative wires," IEEE TCAD, vol. 16, Jun. 1997.
2. Liu, J. B., A.Veneris and H.Takahashi, "Incremental diagnosis of multiple open interconnects," in IEEE Int'l Test Conference 2002.
3. Rohfleisch, B. A. Lolbl, and B. Wurth, "Reducing power dissipation after technology mapping by structural transformations," in Proc. DAC conference 1996.
4. Steiz, G. B. M. Riess, B. Rohfleisch, and F. M. Johannes, "Performance optimization by interacting netlist transformation and placement," IEEE TCAD, vol. 19, Mar. 2000.
5. Veneris, A., M.Amiri and I.Ting, "Design rewiring for power minimization," ISCAS conference 2002.
6. Veneris, A., J. B.Liu, M.Amiri and M.S.Abadir, ``Incremental diagnosis and debugging of multiple faults and errors," IEEE DATE Conference 2002.
7. Veneris, A. and M.S.Abadir, ``Design rewiring using ATPG", IEEE TCAD, vol. 21, no,12, pp.1469-1479, Dec 2002.