

# THE BEST OF ICCAD

20 Years of Excellence in  
Computer-Aided Design

Edited by  
Andreas Kuehlmann

Springer Science+Business Media, LLC

---

# THE BEST OF ICCAD

# THE BEST OF ICCAD

20 Years of Excellence in Computer-Aided Design

Edited by

ANDREAS KUEHLMANN

Cadence Berkeley Labs  
Berkeley, CA, USA



Springer Science+Business Media, LLC

**Library of Congress Cataloging-in-Publication**

Title:                   **THE BEST OF ICCAD**  
                          20 Years of Excellence in Computer-Aided Design  
Edited by:             Andreas Kuehlmann  
ISBN 978-1-4613-5007-1    ISBN 978-1-4615-0292-0 (eBook)  
DOI 10.1007/978-1-4615-0292-0

---

Copyright © 2003 by Springer Science+Business Media New York  
Originally published by Kluwer Academic Publishers in 2003  
Softcover reprint of the hardcover 1st edition 2003

Cover photo displays the IBM Power4 microprocessor chip with 174 million transistors.  
Courtesy of International Business Machines Corporation. Unauthorized use not permitted.  
Photo by Thomas Way.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photo-copying, microfilming, recording, or otherwise, without the prior written permission of the publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

Permissions for books published in the USA: [permissions@wkap.com](mailto:permissions@wkap.com)  
Permissions for books published in Europe: [permissions@wkap.nl](mailto:permissions@wkap.nl)  
*Printed on acid-free paper.*

Cover design by Marc Palmer: <http://www.sigildesign.com>.

# Contents

Foreword	xii
Preface	xiii

## Part I Functional Verification

Formal Methods for Functional Verification <i>Randal E. Bryant and James H. Kukula</i>	3
Automating the Diagnosis and the Rectification of Design Errors with PRIAM <i>Jean Christophe Madre, Olivier Coudert and Jean Paul Billon</i>	17
Functional Comparison of Logic Designs for VLSI Circuits <i>C. Leonard Berman and Louise H. Trevillyan</i>	29
A Unified Framework for the Formal Verification of Sequential Circuits <i>Olivier Coudert and Jean Christophe Madre</i>	39
Dynamic Variable Ordering for Ordered Binary Decision Diagrams <i>R. Rudell</i>	51
Verification of Large Synthesized Designs <i>Daniel Brand</i>	65
GRASP—A New Search Algorithm for Satisfiability <i>João P. Marques Silva and Karem A. Sakallah</i>	73

## Part II System Design and Analysis

System Design and Analysis Overview <i>Hugo De Man and Jan Rabaey</i>	93
An Efficient Microcode-Compiler for Custom DSP-Processors <i>Gert Goossens, Jan Rabaey, Joos Vandewalle and Hugo De Man</i>	107

HYPER-LP: A System for Power Minimization Using Architectural Transformations <i>Anantha P. Chandrakasan, Miodrag Potkonjak, Jan Rabaey and Robert W. Brodersen</i>	117
Power Analysis of Embedded Software: First Step Towards Software Power Minimization <i>Vivek Tiwari, Sharad Malik and Andrew Wolfe</i>	129
A Methodology for Correct-by-Construction Latency Insensitive Design <i>Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha and Alberto L. Sangiovanni-Vincentelli</i>	143
Exploring Performance Tradeoffs for Clustered VLIW ASIPs <i>Margarida F. Jacome, Gustavo de Veciana and Viktor Lapinskii</i>	159

### **Part III Logic Synthesis**

Logic Synthesis Overview <i>Robert K. Brayton and John A. Darringer</i>	181
Multiple-Level Logic Optimization System <i>R. Brayton, E. Detjens, N. Phillips, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung and A. Sangiovanni-Vincentelli</i>	191
Exact Minimization of Multiple-Valued Functions for PLA Optimization <i>R. Rudell and A. Sangiovanni-Vincentelli</i>	205
Improved Logic Optimization Using Global-Flow Analysis <i>C. Leonard Berman and Louise H. Trevillyan</i>	217
A Method for Concurrent Decomposition and Factorization of Boolean Expressions <i>Jagadeesh Vasudevamurthy and Janusz Rajski</i>	227
An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs <i>Jason Cong and Yuzheng Ding</i>	235
Logic Decomposition during Technology Mapping <i>Eric Lehman, Yosinori Watanabe, Joel Grodstein and Heather Harkness</i>	249

### **Part IV Analog and Digital Circuit Design**

Highlights in Analog and Digital Circuit Design and Synthesis at ICCAD <i>Ramesh Harjani, Philippe Magarshack, Gerard Mas and Rob A. Rutenbar</i>	269
--	-----

An Interactive Device Characterization and Model Development System <i>Ebrahim Khalily, Peter H. Decher and Darrell A. Teegarden</i>	285
TILOS: A Posynomial Programming Approach to Transistor Sizing <i>J. P. Fishburn and A. E. Dunlop</i>	295
SPECS2: An Integrated Circuit Timing Simulator <i>Chandu Visweswariah and Ronald A. Rohrer</i>	303
Automatic Synthesis of Operational Amplifiers based on Analytic Circuit Models <i>Han Young Koh, Carlo H. Séquin and Paul R. Gray</i>	313
Analog Circuit Synthesis for Performance in OASYS <i>Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley</i>	325
Extraction of Gate-Level Models from Transistor Circuits by Four-Valued Symbolic Analysis <i>Randal E. Bryant</i>	337
Optimization of Custom MOS Circuits by Transistor Sizing <i>Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill and Chandu Visweswariah</i>	347

## Part V Physical Simulation and Analysis

Highlights in Physical Simulation and Analysis at ICCAD <i>Kenneth S. Kundert and Jacob White</i>	367
Nonlinear Simulation in the Frequency-Domain <i>Kenneth S. Kundert and Alberto Sangiovanni-Vincentelli</i>	383
Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation <i>Peter R. O'Brien and Thomas L. Savarino</i>	393
Efficient Techniques for Inductance Extraction of Complex 3-D Geometries <i>M. Kamon, M. J. Tsuk, C. Smithhisler and J. White</i>	403
Time-Domain Non-Monte Carlo Noise Simulation for Nonlinear Dynamic Circuits with Arbitrary Excitations <i>Alper Demir, Edward W.Y. Liu and Alberto L. Sangiovanni-Vincentelli</i>	413
PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm <i>Altan Odabasioglu, Mustafa Celik and Lawrence T. Pileggi</i>	433

Circuit Noise Evaluation by Padé Approximation Based Model-Reduction Techniques <i>Peter Feldmann and Roland W. Freund</i>	451
--	-----

## Part VI Physical Design

Physical Design Overview <i>Ernest S. Kuh and Chi-Ping Hsu</i>	467
Floorplan Design Using Annealing <i>Ralph H.J.M. Otten and Lukas P.P.P. van Ginneken</i>	479
GOALIE: A Space-Efficient System for VLSI Artwork Analysis <i>Thomas G. Szymanski and Christopher J. Van Wyk</i>	489
Gordian: A New Global Optimization/ Rectangle Dissection Method for Cell Placement <i>Jürgen M. Kleinhans, Georg Sigl and Frank M. Johannes</i>	499
Exact Zero Skew <i>Ren-Song Tsay</i>	509
Efficient Network Flow Based Min-Cut Balanced Partitioning <i>Honghua Hannah Yang and D. F. Wong</i>	521
Rectangle-Packng-Based Module Placement <i>Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake and Yoji Kajitani</i>	535

## Part VII Timing, Test and Manufacturing

Timing, Test and Manufacturing Overview <i>Karem A. Sakallah, Duncan M. (Hank) Walker and Sani R. Nassif</i>	551
A Methodology for Worst Case Design of Integrated Circuits <i>A. J. Strojwas, S. R. Nassif and S. W. Director</i>	563
Timing Analysis using Functional Relationships <i>Daniel Brand and Vijay S. Iyengar</i>	567
On the Design of Robust Multiple Fault Testable CMOS Combinational Logic Circuits <i>Sandip Kundu, Sudhakar M. Reddy and Niraj K. Jha</i>	575
Circuit Optimization Driven by Worst-Case Distances <i>Kurt J. Antreich and Helmut E. Graeb</i>	585

<b>Contents</b>	<b>ix</b>
Verifying Clock Schedules <i>Thomas G. Szymanski and Narendra Shenoy</i>	597
Efficient Implementation of Retiming <i>Narendra Shenoy and Richard Rudell</i>	615
 <b>Part VIII Industry Viewpoints</b>	
A Cadence Perspective on ICCAD <i>Louis K. Scheffer</i>	633
ICCAD and Fujitsu <i>Hiromu Hayashi</i>	639
ICCAD's Impact in IBM <i>John A. Darringer, Leon Stok and Louise H. Trevillyan</i>	645
Magma and ICCAD <i>Michel Berkelaar</i>	653
Designers Face Critical Challenges and Discontinuities of Analog/Mixed Signal Design and Physical Verification <i>Walden C. Rhines</i>	659
NEC and ICCAD - EDA partners in success <i>P. Ashar, S. Chakradhar, A. Gupta, J. Henkel, A. Raghunathan and K. Wakabayashi</i>	663
The Strong Mutual Impact between Philips Research and the ICCAD <i>Emile Aarts and Frans Theeuwen</i>	675
Contributions from the “Best of ICCAD” to Synopsys <i>Raul Camposano, Ahsan Bootehsaz, Debasish Chowdhury, Brent Gregory, Jim Kukula, Narendra Shenoy and Tom Williams</i>	683
ICCAD and Xilinx <i>Rajeev Jayaraman</i>	689
 <b>Index</b>	697
 <b>Author Index</b>	705
 <b>Reference Index</b>	707

# Foreword

In 2002, the International Conference on Computer Aided Design (ICCAD) celebrates its 20th anniversary. This book commemorates contributions made by ICCAD to the broad field of design automation during that time. The foundation of ICCAD in 1982 coincided with the growth of Large Scale Integration. The sharply increased functionality of board-level circuits led to a major demand for more powerful Electronic Design Automation (EDA) tools. At the same time, LSI grew quickly and advanced circuit integration became widely available. This, in turn, required new tools, using sophisticated modeling, analysis and optimization algorithms in order to manage the evermore complex design processes. Not surprisingly, during the same period, a number of start-up companies began to commercialize EDA solutions, complementing various existing in-house efforts. The overall increased interest in Design Automation (DA) required a new forum for the emerging community of EDA professionals; one which would be focused on the publication of high-quality research results and provide a structure for the exchange of ideas on a broad scale.

Many of the original ICCAD volunteers were also members of CANDE (Computer-Aided Network Design), a workshop of the IEEE Circuits and System Society. In fact, it was at a CANDE workshop that Bill McCalla suggested the creation of a conference for the EDA professional. (Bill later developed the name). Throughout the years, CANDE has provided an active meeting place for its members, disclosing and discussing important new developments, but lacking a wide forum for formal paper publications and general participation. To address this need in turn, two conferences were founded - ICCAD and the International Conference on Computer Design (ICCD). ICCAD was largely focused on the algorithmic core of CAD, whereas ICCD was mainly oriented to design and CAD applications.

From its inception, ICCAD was structured as a high-quality conference with a superb review process, thereby emphasizing original contributions with significant theoretical and practical impact. The technical program and executive committees were based on a rotating involvement of key experts from different CAD areas from Asia, Europe, and the American Continent and provided the backbone for the international and multi-disciplinary character of ICCAD. From the beginning, the conference home was established in the heart of Silicon Valley, which was one of the main centers of many emerging CAD activities and entrepreneurs and close to key commercial and university research groups. In fact, an early guiding light for the Conference was Professor Don Pederson from UC Berkeley. ICCAD was initially co-sponsored by the IEEE Circuits and Systems Society and the IEEE Computer Society. In 1992, the ACM Special Interest Group on Design Automation joined as a co-sponsor.

After its foundation, ICCAD quickly became a core conference in the CAD area and a prime choice for high-quality paper publications. The resulting large number of manuscript submissions combined with a stringent selection process resulted in a high standard for its papers. Furthermore, ICCAD's predictable presence in Silicon Valley in November and its primary focus on scientific exchange made ICCAD a key event for EDA novices and experts alike. It became a place where students, colleagues and friends from industry and academia would meet, discuss their work and learn from each other. Soon, regular peripheral meetings and events were organized around the ICCAD schedule, reflecting the importance of the conference to the EDA community.

Over 2,200 papers have been published at ICCAD during the past 20 years, many of them presenting important innovations that have made their way into products or resulted in a long trail of rich follow-up research. For example, many original publications in automatic layout generation first appeared in ICCAD. Similarly, much of the early work on logic synthesis and optimization was presented at ICCAD. Numerous other landmark papers in physical design, synthesis, circuit and system design and analysis, functional verification, statistical modeling and optimization as well as digital testing, timing and manufacturing analysis have been published in ICCAD, a selection of which are presented in this book.

John J. Golembeski, Chair 1st ICCAD (1983)

Paul B. Weil, Chair 3rd ICCAD (1985)

Ian E. Getreu, Chair 4th ICCAD (1986)

In memory of William J. McCalla, Chair 2nd ICCAD (1984)

# Preface

## About This Book

The year 2002 marks the 20th anniversary of the International Conference on Computer Aided Design. We decided to celebrate this event by re-publishing a selection of papers from among the best contributions presented in ICCAD based on their impact on research and applications. In addition to papers, a set of overview articles were solicited from leading researchers to comment on the historical context of the selected papers and to outline their impact on follow-up work. Furthermore, nine sponsoring companies, which have been actively involved in ICCAD, contributed special articles outlining the impact of ICCAD on their businesses.

The selection process for the papers was initiated with a period of public nominations during which the EDA community was invited to suggest landmark ICCAD publications to be included in this collection. Following this period, a selection committee completed the list of candidate papers, divided them into seven topic areas, reviewed the papers in corresponding subcommittees, and made the final selection based on a ranking and voting procedure. The committee was composed of key program committee members from all of the past ICCAD events, and was chaired by an international group of four leading EDA researchers.

During the public nomination phase we received 216 entries from 65 nominees. After the selection committee nominated additional papers, the candidate pool included a total of 141 distinct papers representing approximately 6.4 % of all papers published during the past 20 years of ICCAD. From these candidates 42 papers were selected for this collection. Clearly, this set is only a sample of excellent papers published in ICCAD, since many other important contributions could not be included due to space limitations.

The structure of the book reflects the partitioning of the selection process into topic areas. The papers of the individual areas are grouped into separate book parts which are introduced by corresponding overview articles. A special part on "Industry Viewpoints" includes the articles from the sponsoring companies. In addition to a subject and author index, a reference index lists the authors of papers referenced in the overview and industry articles at the end of the book.

The work on this project became an insightful reminder of how much progress has been made in EDA during the past 20 years and how many excellent contributions were published in ICCAD. We hope that the reader enjoys browsing through this historic collection and perceives the book as encouragement to crack the next wave of emerging EDA problems.

## Selection Committee

### Committee Co-Chairs:

Robert Brayton University of California at Berkeley, Berkeley, USA  
John Darringer IBM T. J. Watson Research Center, Yorktown Heights, USA  
Patrick Dewilde Delft University of Technology, Delft, The Netherlands  
Tatsuo Ohtsuki Waseda University, Tokyo, Japan

### Committee Members

Takahide Inoue	Member Program Committee ICCAD 1983
Paul B. Weil	Program Chair ICCAD 1984
Ian E. Getreu	Program Chair ICCAD 1985
Basant Chawla	Program Chair ICCAD 1986
Al Jimenez	Program Chair ICCAD 1987
Andrzej J. Strojwas	Program Chair ICCAD 1988
Alberto Sangiovanni-Vincentelli	Program Chair ICCAD 1989
Satoshi Goto	Program Chair ICCAD 1990
Louise Trevillyan	Program Chair ICCAD 1991
Michael Lightner	Program Chair ICCAD 1992
Jochen A.G. Jess	Program Chair ICCAD 1993
Richard Rudell	Program Chair ICCAD 1994
Rob A. Rutenbar	Program Chair ICCAD 1995
Ralph H.J.M. Otten	Program Chair ICCAD 1996
Hiroto Yasuura	Program Chair ICCAD 1997
Jacob White	Program Chair ICCAD 1998
Ellen Sentovich	Program Chair ICCAD 1999
Rolf Ernst	Program Chair ICCAD 2000
Lawrence T. Pileggi	Program Chair ICCAD 2001
Andreas Kuehlmann	Program Chair ICCAD 2002

## Reviewers for the Book Articles

Chuck Alpert	Andrew B. Kahng	Giovanni De Micheli
John Cohn	Timothy Kam	Hidetoshi Onodera
Anirudh Devgan	Kurt Keutzer	Sridevan Parameswaran
Rolf Ernst	Yuji Kukimoto	Joel R. Phillips
Masahiro Fujita	Wolfgang Kunz	Lawrence T. Pileggi
Georges Gielen	Luciano Lavagno	Carl Pixley
Soha Hassoun	Malgorzata Marek-Sadowska	Majid Sarrafzadeh

## Acknowledgments

On behalf of the ICCAD 2002 Executive Committee, the editor would like to thank the many people who have been involved in this unique project and contributed to its success. Particular thanks go to the selection committee for spending significant time and effort revisiting all past papers and attempting a fair selection process. We anticipated that asking the authors of the selected papers to reformat their manuscript for the book could be difficult, especially for early papers. However, the reformatting process was surprisingly smooth for which we would like to thank all participating contributors. We are grateful to the authors of the commentary articles for their effort to present balanced overviews of the areas and to the reviewers for providing helpful feedback. Many thanks go to Cathy Larimer for her skill in attending to the many administrative matters and Sue Pope for her excellent job in retyping papers. We would like to thank the team of MP Associates for their help on many fronts ranging from the support of the initial nomination process to the shipment of the printed books.

Our special gratitude goes to the sponsoring companies and to the individuals who organized their support and authored the industry articles. Without their help, this project would not have been possible. The sponsors include Cadence Design Systems, Inc., Fujitsu Laboratories Ltd, IBM Corp., Magma Design Automation, Inc., Mentor Graphics Corp., NEC Corp., Philips Research Laboratories, Synopsys, Inc., and Xilinx, Inc.

We would like to thank our conference sponsors, the IEEE Circuits and Systems Society, the ACM Special Interest Group on Design Automation, and the IEEE Computer Society for supporting this project and Kluwer Academic Publishers for providing editorial help and for publishing the book.

Andreas Kuehlmann  
Program Chair ICCAD 2002

*In those days spirits were brave, the stakes were high, men were real men, women were real women and small furry creatures from Alpha Centauri were real small furry creatures from Alpha Centauri ...*

Douglas Adams  
in “The Hitchhiker’s Guide to the Galaxy”

*... and ICCADs were real ICCADs.*

a past ICCAD Program Chair

## Part I

# FUNCTIONAL VERIFICATION

Formal Methods for Functional Verification <i>Randal E. Bryant and James H. Kukula</i>	3
Automating the Diagnosis and the Rectification of Design Errors with PRIAM ( <i>ICCAD 1989</i> ) <i>Jean Christophe Madre, Olivier Coudert and Jean Paul Billon</i>	17
Functional Comparison of Logic Designs For VLSI Circuits ( <i>ICCAD 1989</i> ) <i>C. Leonard Berman and Louise H. Trevillyan</i>	29
A Unified Framework for the Formal Verification of Sequential Circuits ( <i>ICCAD 1990</i> ) <i>Olivier Coudert and Jean Christophe Madre</i>	39
Dynamic Variable Ordering for Ordered Binary Decision Diagrams ( <i>ICCAD 1993</i> ) <i>R. Rudell</i>	51
Verification of Large Synthesized Designs ( <i>ICCAD 1993</i> ) <i>Daniel Brand</i>	65
GRASP—A New Search Algorithm for Satisfiability ( <i>ICCAD 1996</i> ) <i>João P. Marques Silva and Karem A. Sakallah</i>	73

# FORMAL METHODS FOR FUNCTIONAL VERIFICATION

Randal E. Bryant  
*Computer Science Department*  
*Carnegie Mellon University*  
*Pittsburgh, PA 15213*

James H. Kukula  
*Advanced Technology Group*  
*Synopsys, Inc.*  
*Hillsboro, OR 97124*

## Abstract

Formal hardware verification ranges from proving that two combinational circuits compute the same functions to the much more ambitious task of proving that a sequential circuit obeys some abstract property expressed in temporal logic. In tracing the history of work in this area, we find a few efforts in the 1970s and 1980s, with a big increase in verification capabilities the late 1980s up through today. The advent of efficient Boolean inference methods, starting with Binary Decision Diagrams (BDDs) and more recently with efficient Boolean satisfiability (SAT) checkers has provided the enabling technology for these advances.

## 1. Introduction

Functional hardware verification involves determining whether or not a logic design matches a specification of its intended behavior. Most commonly, the design consists of a combinational or sequential logic gate circuit (possibly derived from an RTL description), and so the analysis can be performed purely at the Boolean level. Furthermore, the circuit is generally assumed to be either fully combinational or fully synchronous, and hence the functionality can be verified without any consideration of the circuit timing.

In many applications, the specification is also given as a logic circuit. This form of verification is referred to as *equivalence checking*. For example, a designer might want to verify that some optimizations to a netlist did not alter its functionality. Even when verifying that a gate-level netlist implements a specification given in a hardware description language such as Verilog or VHDL, the first step is typically to expand the HDL description into gate-level form and then use this as the specification.

Equivalence checking can be further categorized as *combinational* or *sequential*. With combinational equivalence checking, the two circuits are acyclic, gate-level circuits, and the task is to determine whether they compute the same

Boolean functions. Note that combinational equivalence can be used to prove the equivalence of two sequential circuits, as long as these circuits use the same encodings of their states. In fact, the commercial equivalence checkers now in widespread use follow this approach. With sequential equivalence, we are given two sequential circuits that could be using totally different state encodings. The task is to determine whether the two circuits would ever differ in their output values when they are started in their respective initial states and run with some arbitrary input sequence.

Historically, and even to this day, the most common approach to functional verification is to perform extensive simulation. For equivalence checking, this simply involves simulating the two circuits over many patterns and seeing whether they ever produce different values. In principle, combinational circuits could be fully verified by this means, if we were willing to run enough simulation (typically exponential in the number of primary inputs). For sequential equivalence checking, there is no practical bound on the amount of simulation required to prove the two circuits will have identical behavior for all possible input sequences. In this commentary, we will focus instead on *formal* verification, where mathematically based techniques are used to prove equivalence or other properties for all possible input sequences.

Going beyond equivalence checking, a more ambitious task is to prove that a circuit satisfies some general requirement, such as that there should never be a deadlock, or that any bus request will eventually be granted. The most widely studied class of tools for this form of verification are *model checkers*, where the program determines (checks) whether the circuit (model) obeys a property given as a formula in some type of temporal logic. Such formulas can express properties that involve the behavior of the system over time, cases where we use English words such as “always” and “eventually.”

## 2. Early Work in Verification

Formal hardware design verification appears to have developed in the 1970’s from earlier work in hardware testing and in software verification [23]. Testing-based approaches were applied to equivalence checking. Roth initially proposed [59] unrolling sequential logic to perform bounded sequential equivalence checking. Later [60] he introduced the assumption of a tight correspondence between the registers of the circuits to be compared, thus reducing the sequential problem to a combinational one. Kawato et al. [40] developed similar equivalence checking methods around the same time.

Early formal hardware verification methods based on software methods [29] included symbolic simulation [66, 18, 24], user-defined inductive invariants [56] and inductive invariants derived from assertions [50]. These early software-

based methods generally required user interaction to guide expression simplification.

Automated formal methods continued to be developed [32, 55, 20], but before efficient BDD algorithms were introduced most work was based on impoverished data representations (e.g., sum-of-products), or inefficient search routines (variants of the DPLL [31, 30] method used for Boolean satisfiability). The main problem with these approaches was that they did not exploit commonality of structure. Consequently, either the data representations would blow up, or the search routines would run too long.

A notable exception is work at IBM on equivalence checking. They developed programs for internal use that could handle very large combinational circuits.

## 2.1 Early Equivalence Checking at IBM

An algorithmic breakthrough was achieved at IBM with the Differential Boolean Analyzer using Equivalent Sets of Partials (DBA/ESP) [64]. This algorithm provided the satisfiability engine for the equivalence checking tool which was widely used at IBM in the late 1970's through the 1980's. DBA/ESP was inspired by Shannon decomposition and also the method of bifurcations given in Hammer and Rudeanu's book [37], the practical potential of which was recognized by Al Brown. DBA is also closely related to the Binary Decision Diagrams introduced by Akers [2].

DBA proceeds by successive elimination of variables using Shannon decomposition. The key advances that made DBA practical were effective variable ordering heuristics and the ESP feature suggested by Gordon Smith, which detected shared subproblems so that redundant analysis could be avoided. Variable ordering was inspired by the “longest equation” and “most frequent unknown” heuristics of Hammer and Rudeanu. The discovery of shared subproblems was made feasible by a representation that allowed hashing and efficient structural isomorphism checking. Shannon decomposition by itself would generate a binary tree, but common subproblem recognition transforms the tree into a directed acyclic graph, in particular a BDD. Since common subproblem recognition is done before the subproblem is analyzed and based on structural isomorphism rather than functional equivalence, the resulting BDD is not fully reduced. However, satisfiability of the initial problem is immediately determined once the diagram construction is complete or a 1 terminal is reached.

The IBM equivalence checker also provided means to indicate correspondence between internal combinational signals, reducing the complexity of the problems that the DBA/ESP algorithm needed to solve. The possibility of false negatives due to cutting at internal equivalence points was noted, and the use of a manually specified don't care signal to circumvent the problem is described.

Don't care signals could also be used to avoid false negatives due to unreachable states, with support for validating the unreachable state assertion during simulation.

## 2.2 Binary Decision Diagrams

The idea of encoding a Boolean function as a graph of decisions was first proposed by Akers [2], based on an earlier encoding as a straight-line program by Lee[44]. Akers coined the term "Binary Decision Diagram" (BDD) and also explored some of their properties, but he did not provide an efficient method for building BDDs from circuits. Akers' default strategy would be exhaustive simulation to build a complete binary tree, followed by reduction operations to exploit subtree sharing.

In late 1983, Bryant was inspired by the way concurrent fault simulators evaluate the gates in a circuit for both the good and many faulty behaviors in a single pass through the circuit, using lists to encode the multiple different values at each gate. He thought of replacing the list representation with a tree to encode all possible input combinations and then realized the subtrees could be merged to form a directed acyclic graph. This led him to formulate the Reduced Ordered Binary Decision Diagram (ROBDD, but often simply referred to as "BDD") representation and to devise algorithms for performing operations on Boolean functions based on graph algorithms operating on BDDs. He first published these ideas in 1985 [12], with a more complete description in the well known 1986 paper [13] (submitted for publication in 1984).

The success of BDDs stems from an interrelated set of issues:

- The BDD data structure is based on a maximal sharing of substructure. They are not as prone to exponential blow up as are other representations.
- Boolean operations can be performed using simple graph algorithms. The complexity of these operations are polynomial in the graph sizes. They gain efficiency by exploiting the sharing within BDDs.
- They provide a single, homogeneous representation of the problem space. For example, with symbolic model checking BDDs are used to represent the system being modeled, and the sets of possible states of the system. By contrast, many EDA programs shift back and forth between many different data structures.
- By providing a general purpose Boolean manipulation engine, BDDs help application developers think in more abstract terms. Looking at earlier work, we can often see where the application developer muddles concerns about the problem with how the problem is represented.

## 2.3 The Effectiveness of BDDs for Design Automation

Although not among the selected papers for this volume, ICCAD papers by Malik et al. [48] and Fujita et al. [35] put BDDs on the map to the larger CAD community. They showed 1) that fairly elementary heuristics could select reasonably good variable orders for combinational circuits, and 2) that BDDs could then be constructed for large benchmark circuits enabling tasks to be performed (e.g., equivalence checking) that far exceeded what had been done before.

## 2.4 Dynamic Variable Ordering

Rudell's work [62] strongly reinforces the advantage of having a separate Boolean manipulation engine. Previously, others had shown that this engine could handle housecleaning tasks such as automatic garbage collection [52] and cache management [10]. Rudell showed that it could also handle the task of continuously improving the ordering to minimize storage requirements. While a program is running, the BDDs pointed to by the application keep changing in structure (without changing the underlying function being represented.) But, the user need not be concerned with this.

The work is also a masterpiece of careful engineering. Rudell recognized that the BDD transformations could be made without having the update any of the pointers from external sources into the BDD data structure. This allowed many BDD-based applications to be "retrofitted" to use dynamic variable ordering with only minor changes. In practice, dynamic variable ordering often greatly increases the runtime of BDD-based applications, but it can also enable successful completion in cases that would otherwise fail due to excessive memory requirements.

## 2.5 Combinational Equivalence Checking with BDDs

Researchers at Bull Research, headed by François Anceau, were "early adopters" of BDDs. They showed that BDDs could be used to perform equivalence checking of combinational circuits in an industrial setting in 1987, published in a paper [46] at DAC in 1988. (The earlier work at IBM was not widely known, and did not use conventional BDD algorithms.) Their ICCAD paper [47] showed that once you have good equivalence checking and a powerful Boolean engine, you could go beyond a basic Yes/No equivalence check and attempt to determine why the two circuits are not equivalent. It is based on looking for small variants of the circuit (e.g., inserting one inverter or changing one Nand to Nor) to see if the circuits could be made equivalent. They encode these variants symbolically, so that one run of the engine can test all candidate variations. Al-

though this particular application of BDDs has not had widespread use, the paper illustrates a general principle of using BDDs. By using symbolic encoding, one can often replace a long series of tests with a single symbolic evaluation.

Further developments [41, 38] have continued in diagnostic methods for combinational equivalence checking.

## 2.6 Sequential Verification

Sequential equivalence checking requires proving that two state machines have identical behavior. A common approach is to cast this as a reachability problem. First, a composite circuit is constructed consisting of the two original circuits, plus logic that indicates whether the two circuits have different output values. The task then becomes to determine whether, starting in a state where the two original circuits are in their initial states, the composite circuit can ever reach a state where the comparator circuit indicates that the two circuits have generated different outputs. If so, the circuits are inequivalent. This is equivalent to performing model checking on the composite circuit with the temporal logic query  $\text{EF } t$  (“is it ever possible for  $t$  to hold?”), where  $t$  is the output of the comparator circuit. Thus, we can see that sequential equivalence checking is a special case of model checking.

Model checking was first developed by Clarke and Emerson [21, 22] as a way to automatically verify properties of synchronization programs. They also coined the term “model checking.” The first implementations of model checkers used an *explicit state* representation, encoding each state as a node in a graph. For most hardware designs, the number of states is far too large (exponential in the number of state variables) to be represented in such a fashion, and hence model checkers were originally only applied to very small circuits.

The major breakthrough for the application of model checking to hardware design came with the advent of *symbolic model checking*, where both the circuit model and the set of reachable states are encoded implicitly, typically with BDDs.

The history of symbolic model checking and symbolic FSM equivalence is more difficult to trace, with a number of researchers coming up with similar ideas independently. It is generally acknowledged that Ken McMillan originated the idea of BDD-based symbolic model checking in 1987 and implemented one. But, he did not publish any papers about this work at the time. In 1989, Coudert, Berthet, and Madre presented two papers [26, 25] on using symbolic state machine traversal to verify the equivalence of two finite state machines. Bose and Fisher presented a paper [9] describing a symbolic model checker and its implementation. Bahnsen and Kukula [3] also sketched some ideas for BDD-based state traversal, but they did not have any implementation.

A number of advances in model checking were reported in 1990. Burch, Clarke, McMillan, and Dill presented their seminal papers [15, 14]. Their work was the most general of all, showing that they could symbolically evaluate formulas in a mu-calculus logic that can express many other forms of logic, including several different temporal logics. Coudert, Madre, and Berthet presented a paper [28] on a symbolic, computation-tree-logic (CTL) model checker. In the same conference, Pixley [57] presented a sequential equivalence checker, which followed a different approach than the reachability approach sketched above. Pixley’s method involved determining which pairs of states are equivalent to each other, eliminating the need to specify initial states. He also presented detailed algorithms for symbolic model checking, although his only experimental results were for equivalence checking.

The included paper by Coudert and Madre [27] was the first appearance in ICCAD of a paper on symbolic model checking. The subject of the paper was some refinements on how to perform the preimage computation more efficiently than had done before. Coudert and Madre’s model checker was based on a *function vector* approach, where the set of reachable states is encoded as the range of a set of Boolean functions. This approach has not proved as popular as the *characteristic function* approach, where the set of reachable states is encoded as a single Boolean function indicating whether or not a given state is reachable. The Coudert and Madre paper also introduced the operations *constrain* and *restrict*, which were later explored for other applications [45, 63, 1].

Since this early research on symbolic model checking, there has been a continuous stream of research on ways to improve efficiency. The most successful techniques exploit the modularity of circuits by representing and applying the transition relation in a partitioned form [16, 53].

### 3. SAT and ATPG Methods

Binary Decision Diagrams continue to serve as the foundation for many CAD algorithms. They provide a canonical representation, which is compact across a wide range of practical Boolean functions and efficiently supports a wide range of operations such as intersection, inversion, and quantification. In some situations, however, a less powerful approach can be more efficient. For example, some applications require finding only a subset of the solutions of a Boolean equation. Significant advances and fresh applications in dynamic search approaches such as SAT and automatic test pattern generation (ATPG) algorithms have paralleled those in BDDs.

#### 3.1 Efficient Search Space Pruning

GRASP [49] introduced a new generation of SAT solvers that were designed and tuned using EDA benchmarks. GRASP and its successors[54, 36] are based

on the same DPLL search procedure that has been known for decades, but they are much more efficient at pruning the search space to avoid fruitless search. In the case of GRASP, the main contribution was “conflict diagnosis,” where the solver analyzes the conditions leading to a dead end in the search and infers from this a general condition that will make the formula unsatisfiable. This analysis enables *nonchronological backtracking*, where the search engine backtracks through multiple levels of decisions. It also enables *clause learning*, where the SAT solver can add information about a failed search (in the form of a new clause) to its database and thereby avoid repeating a fruitless search. SAT solvers have now supplanted BDDs for many EDA applications where simple Boolean operations are required.

SAT checking has recently flourished as a research area. Each successive generation of SAT checker typically outperforms its predecessors by an order of magnitude, both in terms of the speed on existing benchmarks and the ability to handle larger problems. Much of the recent focus has been on efficiently organizing and maintaining the internal data structures, particularly the set of clauses. The CHAFF solver [54] demonstrated the value of using clever structures to minimize the number of clauses that need to be checked during constraint propagation. Other ideas seen in modern SAT checkers include: *restarts*, where the solver abandons the current search tree and starts a new one, as well as refined techniques for deciding which elements to discard from the clause set, which variable to split on next, and which value (0 or 1) to try first for a splitting variable [36].

One important application of SAT checkers has been to a limited form of model checking, known as a *bounded model checker*[6]. Bounded model checking involves running the circuit for a fixed number of steps from its initial state and determining whether it satisfies the specified temporal properties. This does not guarantee that the properties would then hold indefinitely, but it serves as a useful check, often quickly finding counterexample traces when they exist. The main advantage of bounded model checking is that it can be applied to very large circuits [7, 8].

### 3.2 Exploiting Similarity

Often two combinational circuits being compared for equivalence will have many functionally equivalent internal signals. This observation was already exploited quite early [33, 4] in the evolution of formal equivalence checkers and has since evolved into the most successful approach to improving performance and capacity.

The equivalence checker developed by Berman and Trevillyan [5] detects equivalent points within the two circuits and then partitions the circuits, treating these equivalence points as primary inputs and outputs of the subcircuits. The

key idea in this paper is the use of a weighted graph min-cut algorithm to minimize the number of subcircuit primary inputs, thus improving efficiency of the equivalence check. Treating internal equivalences as primary inputs introduces the possibility of false negatives. This paper outlines a method for eliminating false negatives by incrementally shifting cutpoints back toward the original primary inputs, using BDDs and the compose operation.

While [5] relies primarily on exhaustive simulation, Kunz [43] used ATPG methods to detect and exploit internal equivalences. This more powerful method can be used without introducing internal cutpoints, thus avoiding the false negative problem. Advances in this area continue to be made by many researchers, e.g., [58, 39, 51, 42, 17]. Sequential methods that exploit internal equivalences have also been developed [65, 34].

### 3.3 Leveraging Observability

To further improve the efficiency of combinational equivalence checking, one can extend the notion of internal equivalence to include signals that are different functions of the primary inputs but whose difference is unobservable at primary outputs or state registers. Cerny and Maura [19] introduced such a method that relied on BDD characteristic functions across full cuts of a circuit.

Brand [11] provided improved efficiency with an ATPG-based combinational equivalence checker. This paper introduced the concept of the joining two candidate equivalent signals with a “Miter” circuit, which indicates whether a difference in the signal values is observable at the primary outputs. The complexity of the equivalence check is reduced by repeatedly transforming this joined circuit based on detected equivalences. Broadening the class of equivalences detected, from signals which are identical functions of the primary inputs to signals whose functional difference is undetectable at the primary outputs, allows further simplifications of the problem so that ATPG can be used on larger problems without introducing internal cutpoints with their risk of false negatives.

## 4. Conclusions

The progress in formal verification from the late 1980s through today has been remarkable. The research community has developed and refined both the underlying symbolic manipulation engines as well as the verification tools that use these engines to reason about complex combinational and sequential circuits. In addition, a number of different equivalence and property checkers have become available commercially. Formal combinational equivalence checking tools have become robust enough to be incorporated routinely into industrial methodologies. Still, the needs of the electronics industry in terms of capacity, performance, and functionality far exceed the capabilities of current sequential verification tools, and even combinational checking occasionally fails due to in-

adequate capacity. We can anticipate that this part of the EDA community will continue to flourish in its ideas and importance.

## Acknowledgments

The authors wish to thank Al Brown and Gordon Smith for their help in outlining the early history of formal equivalence checking at IBM.

## References

- [1] M. D. Aagaard, R. B. Jones, and C.-J. H. Seger. Formal verification using parametric representations of Boolean constraints. In *Proc. Design Automation Conference*, pages 402–407, 1999.
- [2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27:509–516, 1978.
- [3] R. J. Bahnsen and J. H. Kukula. Technique for verifying finite state machines. *IBM Technical Disclosure Bulletin*, 32(3A):166–169, 1989.
- [4] L. Berman. On logic comparison. In *Proc. Design Automation Conference*, pages 854–861, 1981.
- [5] L. Berman and L. Trevillyan. Functional comparison of logic designs for VLSI circuits. In *Proc. of the International Conference on Computer Aided Design*, pages 456–459, 1989.
- [6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1579, pages 193–207. Springer, 1999.
- [7] A. Biere, E. M. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a Power PC microprocessor using symbolic model checking without BDDs. In *Computer-Aided Verification*, Lecture Notes in Computer Science 1633. Springer-Verlag, pages 60–71, 1999.
- [8] P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *Computer-Aided Verification*, Lecture Notes in Computer Science 2102. Springer-Verlag, pages 454–464, 2001.
- [9] S. Bose and A. L. Fisher. Automatic verification of synchronous circuits using symbolic logic simulation and temporal logic. In *Proc. IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, pages 759–764, 1989.
- [10] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proc. Design Automation Conference*, pages 40–45, 1990.
- [11] D. Brand. Verification of large synthesized designs. In *Proc. of the International Conference on Computer Aided Design*, pages 534–537, 1993.
- [12] R. E. Bryant. Symbolic manipulation of Boolean functions using a graphical representation. In *22nd Design Automation Conference*, pages 688–694, June 1985.
- [13] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [14] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *Proc. Design Automation Conference*, pages 46–51, 1990.

- [15] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking:  $10^{20}$  states and beyond. In *Proc. Symposium on Logic in Computer Science*, pages 1–33, 1990.
- [16] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proc. Design Automation Conference*, pages 403–407, 1991.
- [17] J. R. Burch and V. Singhal. Tight integration of combinational verification methods. In *Proc. of the International Conference on Computer Aided Design*, pages 570–576, 1998.
- [18] W. Carter, W. Joyner, Jr., and D. Brand. Symbolic simulation for correct machine design. In *Proc. Design Automation Conference*, pages 280–286, 1979.
- [19] E. Cerny, and C. Mauras. Tautology checking using cross-controllability and cross-observability relations. In *Proc. of the International Conference on Computer Aided Design*, pages 34–37, 1990.
- [20] M. S. Chandrasekhar, J. P. Privitera, and K. W. Conradt. Application of term rewriting techniques to hardware design verification. In *Proc. Design Automation Conference*, pages 277–282, 1987.
- [21] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs Workshop*, Lecture Notes in Computer Science 131. Springer-Verlag, 1981.
- [22] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *10th Annual ACM Symposium on Principles of Programming Languages*, 1983.
- [23] W. E. Cory and W. M. vanCleemput. Developments in verification of design correctness: a tutorial. In *Proc. Design Automation Conference*, pages 156–164, 1980.
- [24] W. E. Cory. Symbolic simulation for functional verification with ADLIB and SDL. In *Proc. Design Automation Conference*, pages 82–89, 1981.
- [25] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using Boolean functional vectors. In *Proc. IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, 1989.
- [26] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems: International Workshop Proceedings*, in Lecture Notes in Computer Science 407, pages 365–373. Springer, 1989.
- [27] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proc. of the International Conference on Computer Aided Design*, pages 126–129, 1990.
- [28] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In *Computer Aided Verification*, pages 75–84, 1990.
- [29] J. Darringer. The application of program verification techniques to hardware verification. In *Proc. Design Automation Conference*, pages 375–381, 1979.
- [30] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
- [31] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

- [32] S. Devadas, H. K. T. Ma, and A. L. Sangiovanni-Vincentelli. On the verification of sequential machines at differing levels of abstraction. In *Proc. Design Automation Conference*, pages 271–276, 1987.
- [33] W. E. Donath and H. Ofek. Automatic identification of equivalence points for Boolean logic verification. *IBM Technical Disclosure Bulletin*, 18:2700–2703, 1976.
- [34] C. A. J. van Eijk. Sequential equivalence checking without state space traversal. In *Proc. Design Automation and Test in Europe*, pages 618–623, 1998.
- [35] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *Proc. of the International Conference on Computer Aided Design*, pages 2–5, 1988.
- [36] E. Goldberg, Y. Novikov. *BerkMin: A Fast and Robust SAT Solver* In *Design Automation and Test in Europe*, pages 142–149, 2002.
- [37] P. L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer Verlag, Berlin, New York, 1968.
- [38] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng. Error correction based on verification techniques. In *Proc. Design Automation Conference*, pages 258–261, 1996.
- [39] J. Jain, R. Mukherjee, and M. Fujita. Advanced verification techniques based on learning. In *Proc. Design Automation Conference*, pages 420–426, 1995.
- [40] N. Kawato, T. Saito, F. Maruyama, and T. Uehara. Design and verification of large-scale computers using DDL. In *Proc. Design Automation Conference*, pages 360–366, 1979.
- [41] A. Kuehlmann, D. Cheng, A. Srinivasan, and D. LaPotin. Error diagnosis for transistor-level verification. In *Proc. Design Automation Conference*, pages 218–224, 1994.
- [42] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proc. Design Automation Conference*, pages 263–268, 1997.
- [43] W. Kunz. HANNIBAL: an efficient tools for logic verification based on recursive learning. In *Proc. of the International Conference on Computer Aided Design*, pages 538–543, 1993.
- [44] C. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, July 1959.
- [45] B. Lin, H. J. Touati, and A. R. Newton. Don't care minimization of multi-level sequential logic networks. In *Proc. of the International Conference on Computer Aided Design*, pages 414–417, 1990.
- [46] J. C. Madre and J. P. Billon. Proving circuit correctness using formal comparison between expected and extracted behavior. In *Proc. Design Automation Conference*, pages 205–210, 1988.
- [47] J. C. Madre, O. Coudert, and J. P. Billon. Automating the diagnosis and the rectification of design errors with PRIAM. In *Proc. of the International Conference on Computer Aided Design*, pages 30–33, 1989.
- [48] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. of the International Conference on Computer Aided Design*, pages 6–9, 1988.
- [49] J. P. Marques Silva and K. A. Sakallah. GRASP—a new search algorithm for satisfiability. In *Proc. of the International Conference on Computer Aided Design*, pages 220–227, 1996.
- [50] F. Maruyama, T. Uehara, N. Kawato, and T. Saito. A verification technique for hardware designs. In *Proc. Design Automation Conference*, pages 832–840, 1982.

- [51] Y. Matsunaga. An efficient equivalence checker for combinational gates. In *Proc. Design Automation Conference*, pages 629–634, 1996.
- [52] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In *Proc. Design Automation Conference*, pages 52–57, 1990.
- [53] I.-H. Moon, J. Kukula, K. Ravi, and F. Somenzi. To split or conjoin: the question in image computation. In *Proc. Design Automation Conference*, pages 23–28, 2000.
- [54] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proc. Design Automation Conference*, pages 530–535, 2001.
- [55] G. Odawara, M. Tomita, O. Okuzawa, T. Ohta, and Z. Zhuang. A logic verifier based on Boolean comparison. In *Proc. Design Automation Conference*, pages 208–214, 1986.
- [56] V. Pitchumani and E. Stabler. A formal method for computer design verification. In *Proc. Design Automation Conference*, pages 809–814, 1982.
- [57] C. Pixley. A computational theory and implementation of sequential hardware equivalence. In *Computer Aided Verification*, pages 293–320, 1990.
- [58] S. Reddy, W. Kunz, and D. Pradhan. Novel verification framework combining structural and OBDD methods in a synthesis environment. In *Proc. Design Automation Conference*, pages 414–419, 1995.
- [59] J. P. Roth. Verify: an algorithm to verify a computer design. *IBM Technical Disclosure Bulletin*, 15:2646–2648, 1973.
- [60] J. P. Roth. Hardware verification. *IEEE Transactions on Computers*, C-26:1292–1294, 1977.
- [61] J. P. Roth. *Computer Logic, Testing, and Verification*. Computer Science Press, Potomac, 1980.
- [62] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of the International Conference on Computer Aided Design*, pages 42–47, 1993.
- [63] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton. Heuristic minimization of BDDs using don’t cares. In *Proc. Design Automation Conference*, pages 225–231, 1994.
- [64] G. Smith, R. Bahnsen, and H. Halliwell. Boolean comparison of hardware and flowcharts. *IBM Journal of Research and Development*, 26:106–116, 1982.
- [65] D. Stoffel and W. Kunz. Record and play: a structural fixed point iteration for sequential circuit verification. In *Proc. of the International Conference on Computer Aided Design*, pages 394–399, 1997.
- [66] T. Wagner. Verification of hardware designs thru symbolic manipulation. In *Proc. International Symposium on Design Automation and Microprocessors*, pages 50–53, 1977.

# AUTOMATING THE DIAGNOSIS AND THE RECTIFICATION OF DESIGN ERRORS WITH PRIAM

Jean Christophe Madre<sup>1</sup>, Olivier Coudert<sup>2</sup> and Jean Paul Billon<sup>3</sup>

*Bull Corporate Research Center P.C. 58B1*

*68, Route de Versailles*

*78430 Louveciennes, France*

## Abstract

This paper presents the original extensions brought to PRIAM to automate both the diagnosis and the rectification of the design errors detected by this tool. PRIAM is an industrial automated formal verifier used to check the functional correctness of digital circuits of up to 20000 transistors. These extensions implement a new approach to diagnosis based on Boolean equation solving. In particular, no enumeration of the faulty patterns is necessary to find out the incorrect gates in the circuit. The diagnosis system can handle any circuit that can be verified by PRIAM.

## 1. Introduction

PRIAM is an automated formal verifier of digital circuits now used by industrial designers [6]. It checks the functional correctness of digital circuits with respect to their specifications. The behavioural specifications as well as the circuit descriptions are written in the hardware description language LDS. PRIAM formally proves the equivalence or the implication between LDS programs and thus establishes the correctness of the described circuits.

When PRIAM detects an error during the verification of a circuit, it provides the circuit designer with a description of the error: its kind, the related variable, and the equation of the set of input patterns that make the error observable. However, until now, no help was given to the designer to diagnose the error, that is to find out the reasons of the incorrect behaviour of the circuit. Except for very simple errors, this diagnosis task generally takes several hours to which must be added several hours for the rectification.

This paper presents the diagnosis and rectification system we have developed for the debugging of gate level circuit descriptions. This system is based on the equation solving facility of the automated propositional theorem prover of PRIAM. The diagnosis and rectification are pure formal methods. The diagnosis method overcomes the defaults of the methods based on enumeration of

---

Authors are currently with <sup>1</sup>Synopsys, <sup>2</sup>Monterey Design Systems and <sup>3</sup>SchlumbergerSema.

the faulty patterns. For instance, under the single fault assumption, each gate connected to some incorrect output is analyzed once to determine whether it is responsible for the incorrect value at this output.

The paper is divided in 4 parts. Part 2 presents the verifier PRIAM and its underlying propositional prover. Part 3 describes the diagnosis method. Part 4 explains how the rectification system determines whether the possibly incorrect gates of the circuit can be modified to make the circuit function well. Part 5 gives the fundamental results on theorem proving and Boolean equation solving that underly the diagnosis system.

## 2. The Formal Verifier PRIAM

PRIAM is a formal verifier of functional correctness of digital circuits. This tool has been integrated in the CAD system used by the circuit designers at Bull. Table 1 gives some verification times for industrial circuits. Note that none of these circuits could have been verified by simulation because of their very large numbers of inputs.

Within BULL's design methodology, the behaviour of synchronous circuits is described with the hardware description language LDS. Descriptions are at the cycle level. They are written in a procedural way like VHDL processes [11]. This means that the behavioural specification of a circuit written in LDS describes *how* its output and its transition functions are computed. All the storage elements of the circuit must be declared, and there must be no loop without an included storage element in the circuit.

Circuit	#Inputs	#Outputs	#Trans.	Time
Operabc	160	50	4100	9 mn
Addmul	100	45	5400	15 mn
Tdat	297	192	8400	12 mn
Scd5	663	591	17000	20 mn

Table 1. Verification Times with PRIAM (BULL DPX5000).

PRIAM uses symbolic execution of the LDS description of the circuit to compute its output and transition functions. Symbolic execution consists in executing a program  $\mathcal{P}$  with symbolic instead of logical values assigned to its inputs [4]. Since the output and transition functions of the circuit can be partial functions, PRIAM manipulates *partial Boolean functions* that are represented by *contexted values*. A contexted value  $(C_f, V_f)$  is a couple of Boolean expressions where  $C_f$  represents the domain of definition of the function  $f$  and  $V_f$  represents the function  $f$  on its domain of definition.

All along the symbolic execution of a LDS program  $\mathcal{P}$ , PRIAM has to make proofs in order to establish that the program  $\mathcal{P}$  is correct with respect to LDS se-

mantics [6]. PRIAM is built on a powerful propositional prover that makes these proofs simple to perform. This prover is based on a new canonical representation of propositional formulae called the typed decision graphs (TDG) [2, 6].

The formal comparison between two LDS programs  $\mathcal{P}_s$  and  $\mathcal{P}_r$  is performed through two steps. First PRIAM symbolically executes the programs  $\mathcal{P}_s$  and  $\mathcal{P}_r$  to compute the contexted values of each output  $o$  of the programs. We note these contexted values  $(C_o(\mathcal{P}_s), V_o(\mathcal{P}_s))$  and  $(C_o(\mathcal{P}_r), V_o(\mathcal{P}_r))$  respectively. PRIAM then compares these contexted values according to the selected comparison criterion. If  $\mathcal{P}_r$  must be proved to be implied by  $\mathcal{P}_s$ , which is the mostly used comparison criterion between LDS programs, the formulae to be proved valid for each output  $o$  are the following:

$$C_o(\mathcal{P}_s) \Rightarrow C_o(\mathcal{P}_r), \text{ and } C_o(\mathcal{P}_s) \Rightarrow (V_o(\mathcal{P}_s) \Leftrightarrow V_o(\mathcal{P}_r)).$$

When PRIAM proves that these formulae do not hold for some output of the programs  $\mathcal{P}_s$  and  $\mathcal{P}_r$ , then the circuit is declared to be functionally incorrect. PRIAM then provides the designer with a description of the error: the associated output, the error kind (incorrect context or incorrect value), and the equation of the set of all the faulty patterns. However until now PRIAM did not provide the designer with any help to find the causes of this error.

Next section shows that the tedious diagnosis task can be automated. It then shows that under some conditions, the diagnosis system can also provide the designer with possible rectifications of the circuit.

### 3. Automating The Diagnosis of Design Errors

This section shows that diagnosing design errors is a simpler problem than diagnosing faults in real circuits. It then presents the diagnosis method and it gives experimental results.

Diagnosing design errors is quite different from diagnosing faults in real circuits. Many different kinds of errors can be introduced during the fabrication of a circuit that are difficult to model correctly [7]. On the other hand, when dealing with design errors, only one fault needs to be modelled: a circuit does not function well because some of its gates are incorrect. This does not mean that these gates are incorrectly implemented but rather that they are misused in the logical network. For this reason when several outputs of a circuit are declared incorrect by PRIAM the errors are considered independent and analysed separately.

Different methods have been proposed to diagnose errors in circuits whose structure is known [9, 10]. All are based on enumeration of the faulty patterns, which are the input patterns that make the error observable. In the worst case, all faulty patterns must be enumerated to determine the set of possibly incorrect gates. Moreover, none of these methods directly supports automated rectifica-

tion. The diagnosis method proposed here eliminates this enumeration. Under the single fault assumption, a one-pass process over the gates of the circuit produces the exact set of gates that can be held responsible for the error detected by PRIAM. This set is empty if and only if the single fault assumption does not hold.

### 3.1 The Diagnosis Method

For the sake of clarity this section presents the diagnosis on a pure combinational circuit, that is a circuit without any transistors used as switches. We consider a combinational circuit  $C$  with  $n$  inputs noted  $i_1, \dots, i_n$ . This circuit has got several outputs and at least one of these outputs  $o$  has an incorrect value. This means that the specified Boolean function  $f_s$  and the Boolean function  $f_r$  produced by the circuit at this output  $o$  are not equivalent:  $f_r \neq f_s$ .

The first step in the diagnosis consists in computing the set  $S$  of gates in the circuit that are directly or indirectly connected to the output  $o$ . Indeed only these gates can be responsible for the error. Following [7], we call this set  $S$  of gates the *coverage cone* of the output  $o$ . It contains in general a small part of all the gates of the circuit. It is computed, like in [7], by an algorithm that traverses the circuit  $C$  from the output  $o$  to the inputs  $i_1, \dots, i_n$ .

The second step in the diagnosis is to determine which gates in the coverage cone  $S$  are really responsible for the incorrect value of the variable  $o$ . Under the single fault assumption, this problem is the same as determining whether the behaviour of any gate  $G$  in  $S$  can be modified in such a way that the output function  $f_r$  becomes equivalent to the expected function  $f_s$ . If we note  $[o_1 \dots o_p]$  the outputs of the gate  $G$ , the diagnosis problems comes down to finding a  $p$ -tuple of Boolean functions  $[f'_{o1} \dots f'_{op}]$  that should be produced by the gate  $G$  so that  $f_r = f_s$ .

In order to find the functions  $[f'_{o1} \dots f'_{op}]$ , we create a vector of Boolean variables noted  $[z_1 \dots z_p]$ , and we propagate them in the circuit  $C$  in place of the functions  $[f_{o1} \dots f_{op}]$  actually produced by the gate  $G$ . This propagation, which uses PRIAM's symbolic execution mechanism, produces the function  $f'_r(i_1, \dots, i_n, z_1, \dots, z_p)$  at the output  $o$ . Thanks to the free variables  $z_1, \dots$ , and  $z_p$ , the Boolean function  $f'_r$  represents all the Boolean functions that can be obtained at the output  $o$  by modifying the behaviour of the gate  $G$ . In order for the functions  $f_s$  and  $f'_r$  to be equivalent, we must find, for each set of values that can be assigned to the inputs of the circuit, the values of the variables  $z_1, \dots, z_p$  such that:  $f_s(i_1, \dots, i_n) = f'_r(i_1, \dots, i_n, z_1, \dots, z_p)$ . This is expressed by the following theorem.

**Theorem 1** *The gate G is possibly responsible for the incorrect value at the output o if and only if the following formula is valid:*

$$(C) \forall i_1 \cdots i_n, \exists z_1 \cdots z_p, f'_r(i_1, \dots, i_n, z_1, \dots, z_p) = f_s(i_1, \dots, i_n).$$

Section 5 explains how this non trivial theorem can be automatically proved by the propositional prover of PRIAM. It also shows that the proof procedure can give us the functional values of the variables  $z_1, \dots, z_p$ . These functional values represent all the functions that could be produced by the gate in order for the circuit to be correct. These functional values are used by the rectification system presented in Section 4 to provide the designer with the correct equation of the gate G.

### 3.2 Experimental Results

The performance of the diagnosis method mainly depends on the time needed to propagate the vector  $[z_1 \cdots z_p]$  to the output  $o$ . The variables  $z_1, \dots, z_p$  replace possibly complex Boolean functions so their propagation in place of these functions can be expected to require less time. This is clearly shown in Table 2. It guarantees that any circuit that can be verified with PRIAM can be handled by the diagnosis system.

Circuit/Output	#Stats	#Cone	CPU Time	#Faults
Add32/s15	269	53	48 s	3
Alu32/s14	174	51	192 s	4
Alu32/s0	174	89	660 s	12

Table 2. Diagnosis times for 32 bit circuits (BULL DPX5000).

Table 2 gives the diagnosis times for some industrial circuits. It gives the number of statements (“#Stats”) in the LDS program describing the circuit, the number of statements in the coverage cone of the incorrect output (“#Cone”), the total time (“CPU Time”) needed for the diagnosis, and finally the number of possibly incorrect statements (or gates) (“#Faults”). Note that the circuit *Add32* is described at the gate level and that the circuit *Alu32* is built out of standard cells.

The diagnosis procedure given above uses the single fault assumption. Experience shows that experienced designers make very few faults, so this assumption holds in many cases. When it does not hold, the diagnosis system will be unable to find a gate that can be held responsible for the incorrect behavior of the circuit. Nevertheless the diagnosis procedure can be applied to multiple faults, except that a tuple of gates of the set  $S$  instead of only one gate must be considered at each step. We are then faced with the usual combinatorial explosion in the search.

## 4. Rectifying Design Errors

Section 3 has presented a procedure to find the gates that can be responsible for the incorrect value of some output of a circuit. We now show that this procedure gives us enough information to determine which of these gates can be rectified in order for the output value to become correct.

The key idea here is that the formula (C) given in Section 3.1 can be considered as a theorem to be proved, but also as an equation to be solved. Solving this equation with the procedure given in Section 5 provides us with the functional values  $[f'_{o1} \dots f'_{op}]$  of the variables  $[z_1 \dots z_p]$ . These functions are the Boolean functions that *should* be produced by the gate  $G$  under analysis for the output function  $f_r$  to be correct. Note that there can exist more than one solution to the Boolean equation (C). The solver then introduces a finite number of parameters in the functions. This means that  $[f'_{o1} \dots f'_{op}]$  are higher order functions. They represent the set  $\mathcal{F}$  of all different vectors of output functions of the gate  $G$  for which the value of the output  $o$  is correct.

The problem we address here is to determine whether it is possible to rectify the gate  $G$ , *without changing the global structure of the circuit*. If this is possible, then the cost of rectifying the design error is minimal because the structure of the circuit is not modified. We note  $[f_1 \dots f_m]$  the tuple of Boolean functions taken as inputs by the gate  $G$ . The gate  $G$  produces the output functions  $[f'_{o1} \dots f'_{op}]$ . This means that  $G$  implements  $p$  compositions noted  $h_1, \dots$ , and  $h_p$  such that  $h_i(f_1, \dots, f_m) = f_{oi}$ . To rectify the gate  $G$  is to find  $p$  compositions  $h'_1, \dots$ , and  $h'_p$  that produce correct functions at the gate's outputs. This is expressed by the following theorem.

**Theorem 2** *The gate  $G$  can be rectified if and only if the following formula is valid:*

$$(R) \exists h'_1 \dots h'_p, [h'_1(f_1, \dots, f_m) \dots h'_p(f_1, \dots, f_m)] \in \mathcal{F}.$$

The resolution procedure of this equation is given in section 5.2. It returns, if they exist, all the functions  $[h'_1 \dots h'_p]$  that compose the input functions of the gate  $G$  in such a way that the circuit  $C$  produces a correct function at the output  $o$ . When several functions exist, the designer has to choose between them the best one according to some criteria.

The rectification problem is essentially combinatorial. When the gate  $G$  has  $m$  inputs and  $p$  outputs, the resolution procedure given in Section 5.2 has to deal with  $(p \times 2^m)$  Boolean variables. This combinatorial explosion restricts the rectification method to be applied on circuits whose gates have a small number of inputs and outputs. When this is not the case, for instance when complex standard cells are used in the circuit, the rectification problem looks very much like the synthesis one and the same problems are encountered.

## 4.1 Experimental Results

The CPU time needed to rectify a gate directly depends on its number of inputs and the complexity of its input functions. No rectification is attempted when the number of inputs is larger than 10. Experience shows that for the 32 bit circuits whose diagnosis times are given in Table 2, these CPU times are less than 5 seconds. None of the 3 possibly incorrect gates of the 32 bit adder *Add32* can be rectified. For the 32 bit ALU *Alu32*, the system finds that only one gate can be rectified in order for the output *s14* to become correct, and only one gate for the output *s0*. Eventually these two gates are the same one.

## 5. The Boolean Equation Solver

We have presented in [6] the propositional theorem prover underlying PRIAM. This prover is based on a new canonical form of the propositional formulae that we have called the Shannon typed canonical form. Formulae in this form are represented by graphs called Typed Decision Graphs (TDG). In PRIAM the prover is mainly used as a rewriting system to reduce propositional formulae to their canonical form. In this section we show that it can also be used to prove quantified formulae valid. We then show that proving a quantified formula valid is similar to solving a Boolean equation and we give in Section 5.2 several resolution procedures.

### 5.1 Validity of a Quantified Expression

We consider here a higher-order logic with a finite domain of interpretation. Since the domain of interpretation is finite, any closed formula in this logic can be rewritten into a closed formula whose variables are propositional variables [5]. For instance, the formula  $(\forall x_1 x_2 \exists x_3 (x_1 \vee x_3) \Leftrightarrow (x_2 \vee x_3))$  is such a quantified formula. A term is a formula without any quantifier. We note  $f_{/x=y}$  the formula obtained by substituting each occurrence of the variable  $x$  with the term  $y$ .

The validity of any closed quantified formula is inductively defined in the following way:

- the formula *True* is valid and the formula *False* is not valid.
- the closed formula  $(\forall x f)$  is valid if and only if both the formulae  $f_{/x=False}$  and  $f_{/x=True}$  are valid.
- the closed formula  $(\exists x f)$  is valid if and only if at least one of the formulae  $f_{/x=False}$  and  $f_{/x=True}$  is valid.

The proof procedure called *valid* given in Figure 1 is a direct implementation of this inductive definition. The formula to be proved valid is  $(Q_1 x_1 \cdots Q_n x_n t)$ ,

```

type vertex = record
    index : integer; low, high : tdg;
end;
type tdg = record
    tag : '-', '+'; node; vertex;
end;
var True, False : tdg;

function valid(t : tdg) : tdg;
    var v : vertex;
    begin
        if (t = True) or (t = False) then return t;
        v = t.node;
        if ( $\forall$ -quantified v.root) then
            if (t.tag = '+') then
                return And(valid(v.low), valid(v.high));
            else
                return And(valid(Not(v.low)), valid(Not(v.high)));
        else
            if (t.tag = '-') then
                return Or(valid(v.low), valid(v.high));
            else
                return Or(valid(Not(v.low)), valid(Not(v.high)));
    end;

```

Figure 1. Proof Procedure of a quantified closed formula.

where  $Q_1, \dots, Q_n$  are quantifiers and  $t$  is a term. The function **valid** takes as input the Shannon tree of the term  $t$  built with the order  $x_1 < \dots < x_n$ . The proof procedure traverses this tree and determines, at each step, whether the subterm represented by some vertex in the tree is valid. The function returns *True* if the term is valid, else it returns *False*.

When all the quantifiers of the formula  $(Q_1x_1 \dots Q_nx_nt)$  are identical, the canonicity of the representation makes the proof trivial. For instance, the formula  $(\forall x_1 \dots x_nt)$  is valid if and only if the term  $t$  is a tautology. In the same way, the formula  $(\exists x_1 \dots x_nt)$  is valid if and only if the term  $t$  is not identically equal to *False*. These remarks are used to optimize the proof procedure given in Figure 1. In a more general way, there is a proof procedure [3] that does not require the TDG of the term  $t$  to be built with the order  $x_1 < \dots < x_n$ .

## 5.2 Solving Equations

Any closed formula  $(Q_1x_1 \dots Q_nx_nt)$  can be seen as an equation whose unknown variables are its existentially quantified variables. From the theoretical point of view, solving the equation “ $t = \text{True}$ ” is the same as finding Skolem’s

functions of the existentially quantified variables [3]. We give here the resolution procedure for an equation that has only unknown variable. The general resolution procedure can be found in [3].

Consider the equation “ $t(x_1, \dots, x_n, y) = \text{True}$ ” with only one unknown variable  $y$ . We note tree <sub>$t$</sub>  the Shannon decomposition tree of the term  $t$  with the order  $y < x_1 < \dots < x_n$ . This tree can be written  $((\neg y) \wedge L) \vee (y \wedge H)$ , where  $L$  and  $H$  are both in canonical form. Then two cases must be considered:

- the formula  $(L \vee H)$  is not a tautology. Then for some interpretation of the variables  $x_1, \dots, x_n$ , both the formulae  $L$  and  $H$  evaluate to *False*, and the term  $t$  also evaluates to *False*. It is thus not possible to assign a value to  $y$  such that  $t$  evaluates to *True*. This means that the equation has no solution.
- the formula  $(L \vee H)$  is a tautology. Then for any interpretation of the variables  $x_1, \dots, x_n$ , either the formulae  $L$  or the formula  $H$  evaluates to *True*. In any case, it is possible to assign a value to  $y$  such that  $t$  evaluates to *True*. When both  $L$  and  $H$  evaluate to *True* (this can happen if the formula  $(L \wedge H)$  is not an antilogy), then any value can be assigned to  $y$ . The solution of the equation is:  $y = (\neg L) \vee (H \wedge v)$ , where  $v$  is a new free variable.

## Solving a Functional Equation

This section explains how the rectification system solves the functional equation (R) presented in Section 4.3. Consider a set of Boolean functions  $f_1, \dots, f_m$ , and  $f$  whose variables are  $i_1, \dots, i_n$ . The problem we address here is to find a Boolean function  $h$  such that:

$$(F1) \quad h(f_1, \dots, f_m) \equiv f.$$

There are  $2^N$ , where  $N = 2^m$ , Boolean functions with  $m$  inputs  $v_1, \dots, v_m$ . All these functions can be represented by a single TDG  $F_m$  built out of the variables  $v_1, \dots, v_m$  and  $N = 2^m$  new Boolean variables that we note  $h_1, \dots, h_N$ . This TDG has  $2^{(m+1)} - 1$  vertices. Any interpretation of the variables  $h_1, \dots, h_N$  defines one and only one of these Boolean functions. Figure 2.a shows  $F_2$ .

Starting from the TDG  $F_m$ , we can compute the TDG  $F_C$  that represents the set of Boolean functions that can be obtained by composing the Boolean functions  $f_1, \dots$ , and  $f_m$ . This TDG is obtained by substituting the TDGs of the functions  $f_1, \dots, f_m$  to the variables  $v_1, \dots, v_m$  in the TDG  $F_m$ . The variables occurring in  $F_C$  are the inputs variables  $i_1, \dots, i_n$ , and the variables  $h_1, \dots, h_N$ . Finally, finding a function  $h$  that is a solution of (F1) is the same as finding a tuple  $(h_1, \dots, h_N)$  that is solution of the equation:

$$(F2) \quad \exists h_1 \dots h_N \forall i_1 \dots i_n F_C(h_1, \dots, h_N, i_1, \dots, i_n) = f(i_1, \dots, i_n).$$

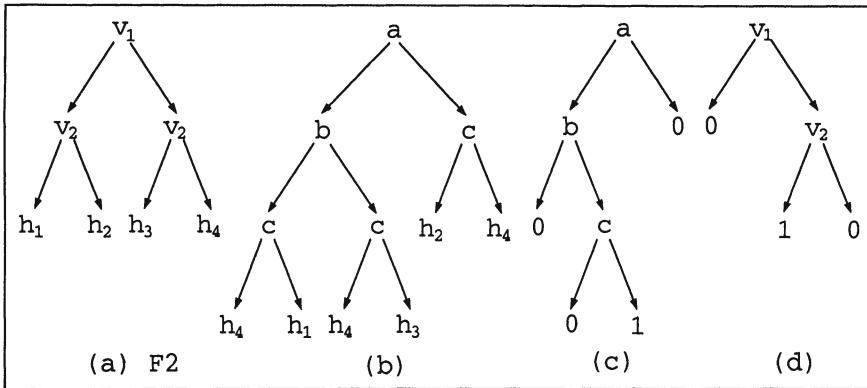


Figure 2. Resolution of a functional equation.

When this equation has solutions, the resolution procedure returns the TDGs representing the values of  $(h_1, \dots, h_m)$ . If there is only one solution to the equation then all these TDGs are equal to either *True* or *False*. However if there are several solutions then some parameters (at the most  $N$ ) occur in the TDGs. These TDGs are then substituted back in the TDG  $F_m$  and we obtain the TDG that represents all the Boolean functions that are solutions of the functional equation (F1).

Figure 2 shows the application of this resolution procedure to the case where  $f_1 = ((\neg a) \wedge b) \vee (a \oplus (\neg c))$ ,  $f_2 = a \vee (\neg c)$ , and the function to be obtained is  $f = ((\neg a) \wedge b \wedge c)$ . Figure 2.b shows the TDG  $F_C$  that represents all the functions that can be obtained by composing the functions  $f_1$  and  $f_2$ . Figure 2.c shows the TDG  $TDG_f$  of the function  $f$ . In order for the TDGs  $F_C$  and  $TDG_f$  to be isomorph we must have  $h_1 = h_2 = h_4 = False$  and  $h_3 = True$ . Figure 2.d is the TDG of the function  $h = v_1 \wedge (\neg v_2)$  that is the only solution of the functional equation.

## 6. Conclusion

This paper has presented the procedures we have developed to automate the diagnosis and the rectification of the design errors detected by the formal verifier PRIAM. The original diagnosis method is based on algorithms for proving quantified propositional formulae and for solving Boolean equations. These algorithms operate on formulae in Shannon typed canonical form that are represented with Typed Decision Graphs. The diagnosis procedure can be applied on any circuit that can be verified by PRIAM. When the diagnosis system has formally proved that some gate can be held responsible for the incorrect behaviour of the circuit, the rectification system is called. It determines whether the cir-

cuit can be rectified without changing its structure, and when this is possible, it provides the designers with the rectified equations of the gate.

## References

- [1] S. B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol C-27, N°6, 1978.
- [2] R. E. Bryant, "Graph-Based Algorithms for Boolean Functions Manipulation", *IEEE Transactions on Computers*, Vol C-35, N°8, pp. 677–692, août 1986.
- [3] O. Coudert, J. C. Madre, "Higher-Order Logics over Finite Domains of Interpretation: Proof and Resolution Procedures", *BULL Research Report*, 1989.
- [4] J. A. Darringer, "The Application of Program Verification Techniques to Hardware Verification", in Proc. of the *16th ACM/IEEE Design Automation Conference*, 1979.
- [5] S. C. Kleene, *Mathematical Logic*, John Wiley and Sons, NY, 1967.
- [6] J. C. Madre, J. P. Billon, "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour", in Proc. of the *25th ACM/IEEE Design Automation Conference*, Anaheim CA, USA, juillet 1988.
- [7] M. Marzouki, B. Courtois, "Debugging Integrated Circuits: AI Can Help!", in Proc. of the *First European Test Conference*, 1989.
- [8] L. C. Paulson, "Natural Deduction As Higher-Order Resolution", in *The Journal of Logic Programming*, N° 3, 1986.
- [9] R. Reiter, "A Theory of Diagnosis from First Principles", *Artificial Intelligence N°32*, Elsevier Science Publishers, 1987.
- [10] H. Simonis, M. Dincbas, "Using Logic Programming for Fault Diagnosis in Digital Circuits", *ECRC Technical Report TR-LP-18*, 1986.
- [11] "IEEE Standard VHDL Reference Manual", IEEE Std 1076-1987. The Institute of Electrical and Electronic Engineers, Inc., New York, 1988.

# FUNCTIONAL COMPARISON OF LOGIC DESIGNS FOR VLSI CIRCUITS

C. Leonard Berman and Louise H. Trevillyan

*IBM T.J. Watson Research Center*

*Yorktown Heights, NY 10598*

## Abstract

Determining whether or not two circuits are functionally equivalent is of fundamental importance in many phases of the design of computer logic. We describe a new method for circuit equivalence which proceeds by reducing the question of whether two circuits are equivalent to a number of more easily answered questions concerning the equivalence of smaller, related circuits. This method can be used to extend the power of any given equivalence-checking algorithm. We report the results of experiments evaluating our technique.

## 1. Introduction

Deciding whether two logic designs are functionally equivalent is a procedure with many applications in the design of computer logic. Unfortunately, the problem of determining functional equivalence of combinational circuits is known to be co-NP complete [4]. This suggests that it will be difficult to develop algorithms which solve this problem efficiently in all cases. In this paper, we present a method which reduces the question of the equivalence of two circuits to a number of questions about the equivalence of smaller circuits. Since the performance of equivalence checkers frequently degrades exponentially with the size of the circuits to be checked, this method can extend the power of an equivalence checker and can lead to vastly improved run times. We report the results of some experiments run to evaluate the method.

Until now, all equivalence checkers have used flat logic models for the circuits they are trying to compare. The possibility of developing “decomposition based” checkers, which prove equivalences between internal signals in the circuits being compared and using these internal equivalences to establish equivalence between the entire circuit, has been suggested [1, 6] however, no practical method for using these internal points has been offered. Our technique supplies this missing piece. We automatically discover a decomposition that facilitates equivalence checking. We do this by employing the min/cut algorithm [7] in a way which permits us to see intermediate equivalence points in the logic to break the full equivalence problem into a number of smaller, more manageable problems in much the same way that the use of lemmas allows us to shorten and simplify the proof of mathematical theorems.

Our experiments suggest that our technique has potential; however, there are limitations. One limitation is that, since our method depends on establishing internal equivalences, it will not work if the two circuits being compared have no equivalent internal signals. We have found that in real applications, such as design verification, there are generally enough corresponding signals to enable our method to achieve significant speedup. We realize that in certain applications this may not be the case.

Another limitation is inherent in any technique that tries to decide equivalence by decomposition. Since the basic step in a procedure of this type is to identify internal equivalences and then to proceed with an equivalence proof treating these internal equivalences as independent inputs, we may fail to recognize that certain functions are identical. We call this failure to recognize identity a *false negative*. In the final section of this paper, we describe an unimplemented method developed with R. E. Bryant, which may address this problem.

Despite these two limitations, the experiments that we report suggest that this technique has significant potential for extending the power of known equivalence checkers.

The paper is organized as follows. In section 2.1, we introduce some required terminology. In 2.2 - 2.4 the different parts of the algorithm are described. In Section 3, we illustrate the algorithm with an example and report our experimental results. Section 4 contains an approach to the problem of false negatives, and Section 5 is a summary.

## 2. Description of the Algorithm

In this section, we present a method which reduces the question of the equivalence of two circuits to a number of questions about the equivalence of smaller circuits.

Our method is based on maintaining two data structures, **PEF-list** and **EQ-list**. During the algorithm, **EQ-list** always contains a list of pairs of signals known to be equivalent. The input to the algorithm includes a list of correspondences between the inputs to the two circuits being tested; this information is used to initialize **EQ-list**. Eventually, if the circuits are equivalent, the corresponding output pairs will be placed in **EQ-list**. **PEF-list** contains a list of pairs of signals which may be equivalent.

The equivalence checking process is broken into three cooperating processes (which are described in detail in the remainder of this section):

- 1 BC: A base checker which can check equivalence of signal pairs produced by “small” circuits.
- 2 PEF: A potential equivalence finder which builds **PEF-list**.

- 3 CDP: A circuit-decomposition process which maintains **EQ-list**. This process uses the *current EQ-list* to determine whether elements of **PEF-list** can be added to **EQ-list**.

Pseudo-code for the algorithm is shown in Figure 1. Our main result is a technique which uses the min-cut algorithm [7] to perform step 3.

## 2.1 Terminology

Circuits are represented as directed acyclic graphs (dags). The nodes of the dag are of three types: INPUT, OUTPUT, or combinational. Combinational nodes are labeled with a description of the function that they perform. We use the terms gate, node and box interchangeably to refer to the nodes of our dag, and the terms wire or signal to refer to the set of edges beginning at a node. Equivalence of synchronous logic (using the same state encoding) can be accommodated in our model by expanding the memory elements in a suitable manner and treating their inputs and outputs as circuit outputs and inputs respectively. If  $s$  and  $t$  are signals, we write  $t \rightarrow s$  if an edge of  $t$  has as a sink the node at which the edges of  $s$  begin. A path from  $t$  to  $s$  is a sequence of signals,  $j_1, \dots, j_r$ , such that  $t \rightarrow j_1 \rightarrow \dots \rightarrow j_r \rightarrow s$ .

The algorithm we present compares two circuits which we call the “reference model” and the “comparison model”. The algorithm begins with a list of corresponding signals from the two models. These signals are called *initial equivalence points*. An assignment to the inputs of the two circuits is consistent if it assigns the same values to corresponding inputs in the two circuits. A pair of signals,  $(s, s')$  is an (*actual*) *equivalence* if  $s$  and  $s'$  receive the same values for all consistent assignments to initial equivalences. In this case, we say that  $s$  and  $s'$  are *functionally equivalent* and they are called *equivalence points*.

Let  $s_1$  and  $s_2 : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $F_1$  and  $F_2 : \{0, 1\}^m \rightarrow \{0, 1\}$ , and  $g : \{0, 1\}^m \rightarrow \{0, 1\}^n$  be such that  $s_1 = F_1 \circ g$  and  $s_2 = F_2 \circ g$ . The triple  $(F_1, F_2, g)$  is called a *simultaneous decomposition* of  $s_1$  and  $s_2$ , and  $m$  is the *size* of the decomposition. The following simple result, which we state without proof, is the basis for our equivalence reduction technique:

**Theorem:** If  $(F_1, F_2, g)$  is a simultaneous decomposition of  $s_1$  and  $s_2$ , then  $(F_1 = F_2) \Rightarrow s_1 = s_2$ .

## 2.2 Potential Equivalence Finder (PEF)

The potential-equivalence finding process produces a list, **PEF-list**, of signal pairs. Every pair of signals  $s$  and  $s'$  which carry identical functions should be on the list, and ideally, we would like to include only pairs of equivalent signals from the circuits. Of course, this is finding all equivalent signals, so we use an efficient “signal signature” to eliminate pairs which can easily be shown

### Min-cut Equivalence Algorithm

```

INPUT: 2 circuits and a list of corresponding inputs
0: Initialize EQ-list to pairs of corresponding signals
1: Call PEF to build PEF-list /* see Section 2.2 */
2: Do while ( PEF-list is not empty)
    2.1 Take the first pair (s,s') from PEF-list
    /* Use the base checker, BC, if possible */
    2.2: If s and s' are “simple enough” call BC(s,s')
        If s ≡ s', put (s,s') on EQ-list
    /* IF BC cannot work, call DCP (section 2.4) */
    2.3: If s and s' are not “simple enough”, call CDP(s,s')
    Od;

```

*Figure 1.* Min-cut Equivalence Algorithm.

to be non-equivalent. This reduces the number of non-equivalent pairs which will later be tested for equivalence. The method we use computes signatures by generating a small number (e.g. 512) of consistent random assignments to the initial equivalences and simulating the circuits for these values. This produces a vector of values at each signal which we use as the signal’s signature. Signatures are hashed and all pairs with the same signature are considered potential equivalences and are placed on **PEF-list**. Finally, PEF orders the elements of the **PEF-list** in breadth-first order beginning with inputs.

### 2.3 Base Checker (BC)

Our decomposition method will work with any equivalence checker. In our experimental implementation, we used an exhaustive simulation checker which could check circuits with up to 30 inputs.

### 2.4 Circuit Decomposition Process (CDP)

The circuit decomposition process performs the bulk of the work. CDP is an iterative process which tries to add the elements of **PEF-list** to **EQ-list**. At each stage, CDP does the following:

- 1 Takes the next pair (s,s') from **PEF-list**. These are called the current signals.
- 2 If s and s' depend on fewer than 10 inputs (any small number could be used), BC is used to check whether they are equivalent.

- 3 If  $s$  and  $s'$  depend on more than 10 inputs, derive a simultaneous decomposition  $(F_1, F_2, g)$  for  $s$  and  $s'$ . (The technique used to derive the decomposition is described below). If  $\text{size}(F_1, F_2, g) \leq 30$ , use BC to determine whether  $F_1 \equiv F_2$ . If so, add  $(s, s')$  to **EQ-list**.

Since **PEF-list** has been ordered in a breadth-first manner, when a pair is checked, “earlier pairs” have already been checked for equivalence and are used to derive the decomposition. IF CDP adds an output signal pair to the list, we have shown that the outputs are equivalent.

We will now describe how the simultaneous decomposition of  $(s, s')$  is derived. First, for signal  $s$ , a directed acyclic graph is formed. The nodes of the graph include a source node, a sink node, a node for signal  $s$  and one node,  $\text{node}(t)$ , for every signal,  $t$ , which is on **EQ-list** and from which there is a path to  $s$ . The edges include one from the source to each of the initial equivalence points, one from  $s$  to the sink and one from  $\text{node(sig1)}$  to  $\text{node(sig2)}$  if there is a path from  $\text{sig1}$  to  $\text{sig2}$  in the circuit which avoids all signals currently on **EQ-list**. A similar dag is built for signal  $s'$ , and the source, sink and corresponding equivalent nodes *not including nodes corresponding to signals known to be equivalent to the current signals* from the two graphs are merged. (That is, if  $(t, v)$  is on **EQ-list** and neither  $t$  nor  $v$  is known to be equivalent to either of the current signals, then  $\text{node}(t)$  and  $\text{node}(v)$  may be merged). Finally, all nodes in the combined graphs which correspond to equivalence points *not including nodes corresponding to signals known to be equivalent to the current signals* are split into an in-half and an out-half. All edges which were incident to the original unsplit node are made incident to the in-half, all edges which began at the unsplit node are moved so they begin at the out-half, and an edge is added from the in-half to the out-half of each split node. All edges are given infinite weight except the edges from the in-half to the out-half of the split nodes. These edges are given weight 1.

The resulting graph represents how the equivalence points in the two cones of logic that feed  $s$  and  $s'$  interact to influence these signals. The min/cut algorithm is now applied to this graph. Observe that a cut separating the sink and source nodes corresponds to a set of equivalences which determine both  $s$  and  $s'$ , and therefore to a simultaneous decomposition,  $(F_1, F_2, g)$ , of  $s$  and  $s'$ . Because of the properties of the min-cut, this is, in some sense, the smallest simultaneous decomposition of  $s$  and  $s'$ .

### 3. Example and Experimental Results

#### 3.1 Example of the Algorithm

We will illustrate the algorithm using the circuits shown in Figure 2. For the sake of illustration, assume that our base checker can handle functions of at most 2 inputs.

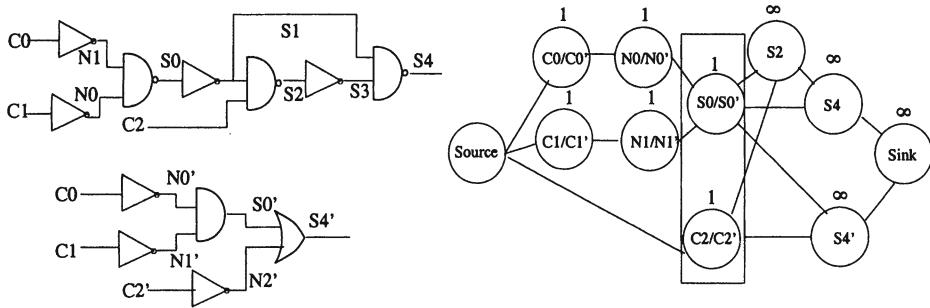


Figure 2. An example of the Comparison Algorithm.

We are given that  $C_0$  is equivalent to  $C_0'$ ,  $C_1$  to  $C_1'$ , and  $C_2$  to  $C_2'$ , so we begin by initializing

$$\text{EQ-list} = ((C_0, C_0'), (C_1, C_1'), (C_2, C_2'))$$

### Random simulation determines

$$\text{PEF-list} = ((C_0, C_0'), (C_1, C_1'), (C_2, C_2'), (N_0, N_0'), (N_1, N_1'), (S_0, S_0'), (S_2, S_4'), (S_4, S_4')).$$

The program begins at  $(N_0, N_0')$  and finds that  $N_0$  and  $N_0'$ ,  $N_1$  and  $N_1'$ , and  $S_0$  and  $S_0'$  are equivalent using the base checker. The later potential equivalence pairs  $(S_2, S_4')$  and  $(S_4, S_4')$  both depend on 3 inputs and, since we are assuming our base checker is limited to 2 inputs, we must use the CDP process. The processing of both pairs is similar, and so we will use  $(S_4, S_4')$  for illustration since it more fully exercises our algorithm. We will therefore assume that we already verified that  $(S_2, S_4')$  is an actual equivalence and that we are now processing the potential equivalence  $(S_4, S_4')$ .

The signals in the cone of  $S_4$  which are currently known to be equivalence points are  $C_0$ ,  $C_1$ ,  $N_0$ ,  $N_1$ ,  $S_0$ ,  $C_2$  and  $S_2$ . The signals in the cone of  $S_4'$  which are currently known to be equivalence points are  $C_0'$ ,  $C_1'$ ,  $C_2'$ ,  $N_0'$ ,  $N_1'$  and  $S_0'$ . First, a graph is built using signals,  $t$ , on **EQ-list** from which  $S_4$  can be reached. A similar graph is made for the comparison model, and they are merged together to yield the graph shown in Figure 2. To simplify the diagram, instead of splitting nodes corresponding to equivalence points, we have labeled the node with the weight which would be assigned to the inserted edge. Nodes which cannot be split are labeled infinity. Note that the node corresponding to  $S_2$  is neither merged nor split because it is known to be equivalent to the current signals. Applying the min-cut algorithm results in a cut consisting of the nodes representing the  $S_0/S_0'$  and  $C_2/C_2'$  equivalence sets. This is indicated by the box in Figure 2. The min-cut has reduced the size of the input set that must be handled from 3 to 2. The program now finds that  $S_4$  and  $S_4'$  are equivalent using the base checker, treating  $S_0/S_0'$  and  $C_2/C_2'$  as independent inputs. Ap-

plying our main theorem permits us to conclude that S4 and S4' are equivalent as functions of the initial equivalences.

### 3.2 Experimental Results

This procedure was implemented in PL/I as part of the Logic Synthesis System [5]. As BC, we used an exhaustive simulation equivalence checker limited to 30 inputs, as described in 2.3. Our test cases included three pieces of IBM logic as well as the ISCAS test set. In all cases, the original specification was compared to the implementation obtained by running LSS with a target of unrestricted NAND gates. The algorithm works best on circuit pairs which have a great deal of structural similarity, and it would be interesting to evaluate the method by comparing implementations designed using different methodologies. We observe, however, that in the context of industrial design, the experiments described here are typical examples of the tasks an equivalence algorithm might be called upon to perform.

<i>Model Name</i>	Equiv. Output Signals	Max Cone S/T	Max Cutset Size	Pairs
Sim 1	66/66	140	19	-
Sim 2	463/477	100	55	-
Sim 3	145/186	97	24	-
C432	2/7	27/36	27	85
C499	32/32	41/41	13	288
C880	26/26	45/45	12	157
C1355	32/32	41/41	10	754
C1908	22/25	33/33	11	1192
C2670	50/140	122	48	-
C5315	122/123	67/67	13	1188

Table 1. Functional Comparison Results (explained in Section 3).

The results are summarized in Table 1. Column 1 identifies the design that was run. Column 2 shows the total number of outputs and how many of them were proved equivalent. Column 3 shows the largest number of inputs in the signature of any output (T) and of any successfully compared output (S). (Where only one figure is given, the number is the number of inputs in the cone of the largest successfully compared output.) Column 4 shows the largest cut set in any successfully compared output. Column 5 shows the number of pairs for which a decomposition was used by CDP to show equivalence of two signals.

We note that in most cases, even though the two designs were structurally very different, the program was able to successfully decompose the logic and establish equivalences which depend on more than 30 inputs and which would have been impossible for the unaided base checker to establish. Note that in

approximately half of the cases our method allowed the base checker to establish complete equivalence between the two designs, and in all but one of the rest, it resulted in significant reductions in the number of inputs which had to be treated simultaneously.

#### 4. Avoiding False Negatives

As mentioned earlier, it is possible for the min-cut procedure to select a decomposition that causes two signals which are actually equal to appear to differ. In this section, we describe a method which may help avoid these “false negatives”. This method, which was developed in conversation with R.E. Bryant, is as yet unimplemented. We will, however, explain why we think it may be effective in addressing this problem.

In this section, we assume that we are using the reduced function graph method [3] as the base checker, and that the reader is familiar with this representation. Assume that we have processed part of the two circuits as described above, that we have established a number of internal equivalences, and that we have represented each of the equivalences as reduced function graphs whose variables are “earlier” internal equivalences and/or initial equivalences. This requires a total order on both initial equivalences and internal signals of the two circuits. Such an order can be computed using techniques described in [2]. Ordering initial equivalences is discussed in [8, 9].

Assume that  $(s, s')$  is a potential equivalence and that  $s$  and  $s'$  have simultaneous decomposition  $(F_1, F_2, g)$  and that  $F_1 \neq F_2$ . We wish to determine whether  $F_1 \circ g \equiv F_2 \circ g$ . In addition, for efficiency, we want to do this in a way which avoids representing the entire logic graphs as functions of the initial equivalences.

The algorithm we suggest has two parts. First, form the reduced function graph of  $H = F_1 \oplus F_2$ . Second, until no internal equivalences appears as variables in  $H$ , choose an internal equivalence which occurs as a variable in  $H$ , and expand this equivalence in terms of earlier equivalences. During step 2 of the algorithm, either the resulting function becomes 0, or all internal equivalences have been expanded and the result is an expression for the exclusive-or as a function of the real inputs, or the required functions grow too large and can no longer be represented as reduced function graphs. In the first case, we have shown equivalence, and in the second case, we have shown true non-equivalence. In the final case, we can draw no conclusion about the two functions.

If  $s$  and  $s'$  are actually equivalent, the dag representation of  $F_1 \oplus F_2$ , in terms of the internal equivalences may be comparatively small. As the equivalences are expanded and more information about the relationships among them is added, the exclusive-or must eventually assume its actual value which, if  $s = s'$ , is zero. These observations suggest that we may be able to compute with the exclusive-

or in this way, even in cases where we could not represent either  $s$  or  $s'$  directly in terms of the initial equivalences. There is, of course, no guarantee that the BDDs will remain small; however, we feel it is an interesting approach to the problem of false negatives in a “decomposition” equivalence checker.

## 5. Summary

We have described an algorithm for testing functional equivalence between two logic designs. Our method can augment the power of *any* given equivalence checker by automatically reducing the equivalence problem to a number of small related problems. Our experimental implementation demonstrated that our method could verify equivalence of circuits whose equivalence could not have been verified by the unaugmented algorithm.

The primary technical contribution is a technique for discovering internal equivalences and using them to show the equivalence of the outputs. Our method involves the use of signatures to reduce the number of potentially equivalent signals, and the use of the min-cut algorithm to reduce the original problem to related problems with fewer independent inputs. This algorithm has been implemented and has been shown to be effective on sample logic. The testing technique used in the implementation was exhaustive simulation, but other more efficient techniques could be used to increase the power and improve the run time of this procedure.

One danger with “decomposition” equivalence checkers is that an inconsistent test case may give a false negative. We have outlined an heuristic that could be employed to ameliorate this problem.

## Acknowledgments

We would like to thank R. E. Bryant for interesting conversations which led to the ideas reported in Section 4.

## References

- [1] C. Leonard Berman, “On Logic Comparison”,*Proc. 18th DAC*, Nashville, TN, 1981.
- [2] C. Leonard Berman, “Ordered Binary Decision Diagrams and Circuit Structure”,*Proc. of the ICCD*, Cambridge, MA, Oct 214, 1989.
- [3] Randal E. Bryant, “Graph Based Algorithms for Boolean Function Manipulation”,*IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677-691, 1986.
- [4] Steven A. Cook, “The complexity of theorem proving procedures”,*Proc 3rd ACM SIGACT*, pp. 151-158, 1971.
- [5] J.A. Darringer, D. Brand, J.V. Gerbi, W.H. Joyner and L.H. Trevillyan, “LSS: A System for Production Logic Synthesis”,*IBM Journal of Research and Development*, vol. 28, no. 5, pp. 537-545, Sept 1984.
- [6] W.E. Donath and H. Ofek, “Automatic Identification of Equivalence Points for Boolean Logic Verification”,*IBM Tech. Disclosure Bulletin*, vol. 18, pp.2700-2703, 1976

- [7] L.R. Ford,Jr. and D.R. Fulkerson,*Flows in Networks*,Princeton, NJ: Princeton University Press, 1962.
- [8] M. Fujita,H. Fujiwara and N. Kawato,"Evaluation and Improvements of Boolean Comparison Method based on Binary Decision Diagrams",*Proceedings of ICCAD*, Nov 1988.
- [9] Sharad Malik, Albert R. Wang, Robert K. Brayton and Alberto Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", *Proceedings of ICCAD*, Nov 1988.

# A UNIFIED FRAMEWORK FOR THE FORMAL VERIFICATION OF SEQUENTIAL CIRCUITS

Olivier Coudert<sup>1</sup> and Jean Christophe Madre<sup>2</sup>

*Bull Corporate Research Center PC 62A13*

*68, Route de Versailles*

*78430 Louveciennes, France*

## 1. Introduction

Hardware description languages (HDLs) dramatically change the way circuit designers work. These languages can be used to describe circuits at a very high level of abstraction, which allows the designers to specify the behavior of a circuit before realizing it. The validation of these specifications is currently done by executing them, which is very costly [2]. This cost motivates the research [3, 5, 7, 10] done on the automatic verification of temporal properties of finite state machines.

Once the design of the circuit is done, the problem is to verify that the resulting circuit is correct with respect to its specification. Until recently, this verification was done by simulating the circuit and its specification on the same input sequences and by comparing their output sequences. The verification method is very costly and incomplete because of the large number of input sequences to consider [2].

This paper presents a unified framework for the verification of synchronous circuits. Within this framework the two verification tasks presented above can be automatically performed using algorithms based on the same concepts. The first idea is to manipulate sets of states and sets of transitions instead of individual states and individual transitions. The second idea is to represent these sets by Boolean functions and to replace operations on sets with operations on Boolean functions.

Part 2 of the paper defines the two problems addressed here, and then it presents the verification algorithms. It shows that these algorithms use the standard set operations in addition to two specific operations called “*Pre*” and “*Img*”. Part 3 briefly explains why the basic set operations are very efficiently performed when sets are denoted by the Typed Decision Graphs of their characteristic functions. Part 4 presents the new Boolean operators “*Constrain*” and “*Restrict*”, and

---

Authors are currently with <sup>1</sup>Monterey Design Systems and <sup>2</sup>Synopsys.

the function “*Expand*” that support efficiently the “*Img*” and “*Pre*” operations. Part 5 gives experimental results and discusses them.

## 2. The Verification Algorithms

This section defines the model of sequential circuits that will be verified, and the two verification problems addressed here. Then it gives for both problems an algorithm based on set manipulations.

### 2.1 Definitions

For the sake of clarity we will consider in this paper that the sequential circuits that must be verified are deterministic Moore machines. Dealing with Mealy machines is done in a similar way. Moreover we assume that these machines are completely specified, which means that for any state of the machine, (1) the outputs are defined, and (2) for any input pattern, the next state of the machine is defined. This is not a limitation since, if the machine is incompletely specified, it is possible to add a dummy state in order to obtain a completely specified machine.

A deterministic Moore machine  $\mathcal{M}$  is defined by a 6-tuple  $(Y, I, O, \delta, \lambda, Init)$ .  $Y$  is the vector  $[y_1, \dots, y_m]$  of Boolean state variables of the machine: a *state* of  $\mathcal{M}$  is defined by the Boolean values of the variables  $y_1, \dots, y_m$ .  $I$  is the vector of  $n$  boolean inputs of the machine.  $O$  is the vector of  $k$  boolean outputs of the machine. The output function  $\lambda$  is a vector of  $k$  Boolean functions (one for each output) from the set  $\{0, 1\}^m$  into  $\{0, 1\}$ . The transition function  $\delta$  is a vector of  $m$  Boolean functions from  $\{0, 1\}^n$  into  $\{0, 1\}$ . Finally  $Init$  is the initial state of the machine.

The 6-tuple that defines a sequential circuit can be obtained either from its gate level description or from its functional description, using a symbolic execution process such as the one used in PRIAM [2].

### 2.2 Verification of Temporal Properties

The temporal formulas that the verification system takes as input are the state formulas of the computation tree logic CTL [7]. This logic is a formalism that was specifically developed to express properties of the states and the computation paths of finite state systems. The meaning of a state formula is relative to a state of machine, which is here defined by the values of its state variables. The 4 basic kinds of CTL state formulas are the following:

- (1)  $(y_1), \dots, (y_n)$  are state formulas. For any state  $s$ ,  $s \models y_j$  if and only if (iff) the value of the variable  $y_j$  is 1 in the state  $s$ .
- (2) If  $f$  and  $g$  are state formulas then so are the formulas  $(\neg f)$ ,  $(f \wedge g)$ ,  $(f \vee g)$ ,  $(f \Leftrightarrow g)$ , and  $(f \Rightarrow g)$ .

- (3) If  $f$  is a state formula, so are the formulas  $EX(f)$  and  $AX(f)$ :  $s \models EX(f)$  iff there exists at least one input pattern  $p$  such that  $\delta(s, p) \models f$ , and  $AX(f) =_{def} \neg EX(\neg f)$ .
- (4) If  $f$  and  $g$  are state formulas, so are the formulas  $E[f \cup g]$  and  $A[f \cup g]$ :  $s \models E[f \cup g]$  iff there exists at least one path  $(s_0, s_1, \dots)$  with  $s_0 = s$ , such that  $\exists i((s_i \models g) \wedge (\forall j(0 \leq j < i \Rightarrow s_j \models f)))$ , and  $s \models A[f \cup g]$  iff for all paths  $(s_0, s_1, \dots)$  such that  $s_0 = s$ , then  $\exists i((s_i \models g) \wedge (\forall j(0 \leq j < i \Rightarrow s_j \models f)))$ .

The first algorithms that have been proposed to verify automatically that some machine holds a temporal property [7] used traversal techniques of its state-transition graph, which had to be partially or entirely built. This limited the application of these algorithms to relatively small machines.

The verification algorithm used here takes as inputs a machine  $\mathcal{M} = (Y, I, O, \lambda, \delta, Init)$  and the temporal formula  $f$  to be verified. It recursively computes the set of states of  $\mathcal{M}$  that satisfy the formula  $f$  from the sets of states that satisfy its subformulas. At each step there are only 4 basic cases to consider that correspond to the 4 basic kinds of formulas given above. Once the set of states  $F$  that satisfy the whole formula is obtained, to check whether  $(s \models f)$  for some state  $s$  of  $\mathcal{M}$  comes down to checking whether  $s$  belongs to  $F$  [3].

The sets of states that satisfy formula of type (1) and (2) can be computed using the basic set operations. For instance, the set of states that satisfy the formula  $(y_1)$  is  $\{1\} \times \{0, 1\}^{m-1}$ ; if  $F$  and  $G$  are the sets of states that satisfy the formulas  $f$  and  $g$  respectively, then the set of states that satisfy the formula  $(f \vee g)$  is  $(F \cup G)$ .

The other kinds of formulas are treated with the “*Pre*” operation, either in one step ( $EX$  and  $AX$  formulas) or by fixed point algorithms ( $EU$  and  $AU$  formulas). By definition,

$$Pre(Q, A, B) = \{s | (s \in A) \wedge (Qp \delta(s, p) \in B)\},$$

Where  $Q$  is either the existential “ $\exists$ ” or the universal “ $\forall$ ” quantifier.  $Pre(Q, A, B)$  is the subset of states of  $A$  which have either at least one successor ( $Q = \exists$ ) or all their successors ( $Q = \forall$ ) in the set  $B$ . Let  $f$  be a formula and  $F$  be the set of states that satisfy  $f$ . The sets of states  $EX$  and  $AX$  that satisfy  $EX(f)$  and  $AX(f)$  respectively are defined by:

$$EX = Pre(\exists, \{0, 1\}^m, F) \quad (1)$$

$$AX = Pre(\forall, \{0, 1\}^m, F) \quad (2)$$

Let  $f$  and  $g$  be two formulas and  $F$  and  $G$  be the sets of states that satisfy  $f$  and  $g$  respectively. The sets of states  $EU$  and  $AU$  that satisfy the formulas  $E[f \cup g]$

and  $A[f \text{ } U \text{ } g]$  respectively are the limits of the following converging sequences of set  $(E_k)$  and  $(A_k)$  [3]:

$$E_0 = G, \text{ and } E_{k+1} = E_k \cup \text{Pre}(\exists, F, E_k), \quad (3)$$

$$A_0 = G, \text{ and } A_{k+1} = A_k \cup \text{Pre}(\forall, F, A_k). \quad (4)$$

These algorithms use only the basic set operations  $\cup, \cap, =$ , in addition with the “*Pre*” operation.

## 2.3 Comparison of Sequential Machines

The fundamental method for comparing the observable behaviors of  $\mathcal{M}_1 = (Y_1, I, O, \delta_1, \lambda_1, \text{Init}_1)$  and  $\mathcal{M}_2 = (Y_2, I, O, \delta_2, \lambda_2, \text{Init}_2)$  is to check that the output *ok* of the product machine  $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$  is equal to 1 for every valid state of  $\mathcal{M}$ . This machine is defined by  $\mathcal{M} = (Y, I, [\text{ok}], \delta, \lambda, \text{Init})$  where  $Y = Y_1 @ Y_2$ , (“@” is the vector concatenation),  $\delta = \delta_1 @ \delta_2$ ,  $\text{Init} = \text{Init}_1 \times \text{Init}_2$  and  $\lambda_{\text{ok}}$  depends on the comparison.

The correctness property given above holds for the machine  $\mathcal{M}$  iff the CTL state formula  $(\neg E[\text{True } U (\text{ok} = 0)])$  holds in its initial state *Init*, so the verification algorithm presented in 2.2 can be used to compare two machines. We give here a specific comparison algorithm that has been shown by experience to be much more efficient than this general algorithm. Both methods will be discussed in Part 5.

The idea used here is to compute the set *Valid* of all the valid states of the machines  $\mathcal{M}$ . This set is the limit of the converging sequence of sets  $V_k$  defined by the equations:

$$V_0 = \text{Init}, \text{ and } V_{k+1} = V_k \cup \text{Img}(\delta, V_k \times \{0, 1\}^n), \quad (5)$$

where  $\text{Img}(f, A) =_{\text{def}} \{f(a) | a \in A\}$  is the image of the set  $A$  with respect to the function  $f$ . Once *Valid* is computed, the verification comes down to testing whether

$$\text{Img}(\lambda_{\text{ok}}, \text{Valid}) = \{1\}. \quad (6)$$

This algorithm, like the one given in the previous section, uses only the basic set operations, in addition to the “*Img*” operation.

## 3. Boolean Functions and Sets

Any subset  $A$  of  $\{0, 1\}^n$  can be represented by a unique Boolean function  $\chi_A$  from  $\{0, 1\}^n$  to  $\{0, 1\}$ , defined by:  $\chi_A(a) = 1$  if and only if  $a \in A$ . The function  $\chi_A$  is called the characteristic function of  $A$ . The set operators  $(\cup, \cap, \subset, \in, \times)$  can be expressed in terms of the logical operators  $(\vee, \wedge, \Rightarrow, \neg, \Leftrightarrow)$ . For instance, the characteristic function of the set  $A \cup B$  is  $(\chi_A(y) \vee \chi_B(y))$ .

Typed decision graphs [1] are a compact canonical representation of Boolean functions. They have remarkable properties that make the symbolic manipulations on Boolean functions very efficient. Typed decision graphs, which are binary decision diagrams [4] with typed edges, are the canonical graph representation associated to Shannon's typed canonical form [1]. By associating a unique atom to each of the component of the cartesian product  $\{0, 1\}^n$ , any characteristic function can be represented by a unique typed decision graph.

The correspondence mentioned above gives the computational cost of the elementary set operations. The negation on typed decision graphs has a null cost so this is the same for the set complementation. The others Boolean operations have a complexity in  $O(|G_1| \times |G_2|)$ , where  $|G|$  is the number of vertices in the graph  $G$ , which gives the computational cost of the corresponding elementary set operations.

There is no relation between the number of elements in a set and the number of vertices in the graph of its characteristic function. However there exists some subsets of  $\{0, 1\}^n$  whose graphs have  $O(2^n / \log(n))$  vertices. Experience shows that, for most of the machines we deal with, while the sets manipulated by the verification algorithms are very large, the graphs of their characteristic functions stay small. This means that the computational cost of the basic set operations performed by the symbolic verification algorithms is low, and that the total cost of these algorithms depends on the costs of the operations “*Pre*” and “*Img*”

## 4. The “*Pre*” and “*Img*” Operations

This part explains how the operations ‘*Pre*’ and ‘*Img*’ can be easily realized when the typed decision graph of the transition relation and of the output relation of the machine can be build [5]. Then it presents the techniques we have developed to perform these operations when it is not possible to built these graphs, which happens for most complex circuits.

### 4.1 Using the transition and the output relations

The transitions relation  $\Delta$  of the machine  $\mathcal{M}$  is a subset of  $\{0, 1\}^m \times \{0, 1\}^n \times \{0, 1\}^m$ . For any states  $s$  and  $s'$  of  $\mathcal{M}$ , and for any input pattern pattern  $p$ ,  $(s, p, s')$  belongs to  $\Delta$  if and only if  $s' = \delta(s, p)$ . For any subset  $A$  and  $B$  of  $\{0, 1\}^m$ , the characteristic function of the set  $Pre(Q, A, B)$  is equal to:

$$\chi_{Pre(Q,A,B)}(s) = \chi_A(s) \wedge (Qp \exists s' \chi_B(s') \wedge \chi_\Delta(s, p, s')).$$

The output relation of the machine  $\mathcal{M}$ , noted  $\Lambda$ , is a subset of  $\{0, 1\}^m \times \{0, 1\}^k$ . For any state  $s$  of  $\mathcal{M}$ , and for any output pattern  $o$ ,  $(s, o)$  belongs to  $\Lambda$  if and only

if  $o = \lambda(s)$ . The equations 5 and 6, using the operation “*Img*”, can be written as:

$$\chi_{Img(\delta, A \times \{0,1\}^n)}(s') = \exists p \exists s \chi_A(s) \wedge \chi_\Delta(s, p, s') \quad (7)$$

$$\chi_{Img(\lambda, A)}(o) = \exists s \chi_A(s) \wedge \chi_\Lambda(s, o) \quad (8)$$

Since the formula  $(\exists x f(x))$  is equivalent to  $(f(0) \vee f(1))$ , and the formula  $(\forall x f(x))$  is equivalent to  $(f(0) \wedge f(1))$ , the graphs of the characteristic functions of  $Pre(Q, A, B)$ ,  $Img(\delta, A \times \{0,1\}^n)$  and  $Img(\lambda, A)$  can be directly computed from the graphs of  $\chi_A$ ,  $\chi_B$ ,  $\chi_\Delta$ , and  $\chi_\Lambda$  by eliminating the quantified atoms associated with  $p$ ,  $s$  and  $s'$ . This technique is very efficient [5], because it uses only the operators  $\vee$  and  $\wedge$ , which have been shown in Section 3 to have relatively low computational costs. The problem is that, for complex machines, it is not possible to build the graphs of  $\Delta$  and  $\Lambda$  [8].

## 4.2 The “Restrict” and “Expand” Operators

The equations that define the “*Pre*” operation are the following:

$$\chi_{Pre(\exists, A, B)}(s) = \chi_A(s) \wedge (\exists p \chi_B(\delta(s, p))), \text{ and} \quad (9)$$

$$\chi_{Pre(\forall, A, B)}(s) = \chi_A(s) \wedge \neg(\exists p \neg \chi_B(\delta(s, p))). \quad (10)$$

The graph of the function  $(\exists p \chi(\delta(s, p)))$ , where  $\chi$  is  $\chi_B$  or  $\neg \chi_B$ , can be computed in two steps. The first step consists in computing the function  $\chi \circ \delta$ . Then, the quantified atoms associated to the input pattern  $p$  are eliminated from the graph of this function.

It has been shown [4] that substituting some variable  $v$  in the graph  $G_1$  with the graph  $G_2$  has a computational cost in  $O(|G_1|^2 \times |G_2|)$ . In order to compute the graph  $\chi(\delta_1(s, p), \dots, \delta_m(s, p))$ , this basic substitution process must be iterated so that all variables of  $\chi$  are substituted by the graphs of the corresponding components of  $d$ . The problem is that during this composition, some intermediate graphs can be too large to be built.

Section 4.2.1 shows that it is not necessary to build the graphs of  $\chi \circ \delta$  to compute  $Pre(Q, A, B)$ . It presents the function “*Expand*” that avoids this construction. Section 4.2.2 presents the “*Restrict*” operator that further reduces the computational cost of the “*Pre*” operation by reducing the sizes of the graphs that are manipulated.

**4.2.1 The function “Expand”.** The idea that underlies the function “*Expand*” is to express the function  $\chi \circ \delta$  as a sum of  $K$  functions  $h_1, \dots, h_k$ , whose graphs have less vertices than the graph of  $\chi \circ \delta$ . Using these functions, the term  $(\exists p \chi(\delta(s, p)))$  can be rewritten into  $(\vee_j (\exists p h_j(s, p)))$ . This identity allows us to eliminate directly the quantified atoms associated to the input pattern  $p$  from the graphs of the functions  $h_1, \dots, h_k$ , and so the graph of  $\chi \circ \delta$  does not have to be built.

Each path in the graph  $G$  of the function  $\chi$  starting from the root and leading to the leaf 1 defines a cube  $c_j$  of the function  $\chi$ . This means that the function  $C_j \circ \delta$  can be taken as one function  $h_j$ . The problems are that the function  $\chi$  can have  $O(2^{|G|})$  cubes, and that even if its number of cubes is relatively small, many redundant computations will be made.

The function `Expand` performs a top-down traversal of the graph  $G$ , and stores in each of its vertices the graph of the function  $C_v \circ \delta$ , where the function  $C_v$  is the sum of all the cubes represented by the paths starting from the root of  $G$  and leading to  $v$ . Each time the top-down traversal reaches the leaf 1, the function `Expand` produces one of the function  $h_j$ . The function  $C_v$  associated to a vertex  $v$  is recursively computed using the functions  $C_w$  of the vertices  $w$  that point to  $v$ . Thanks to the sharing in the graph, partial results are factorized and redundant computations are avoided [10].

Experience shows that the graphs of the functions  $h_1, \dots, h_k$  generated by the `Expand` operation are smaller than the graph of  $\chi \circ \delta$ . The time needed to compute each of these functions directly depends on the sizes of the graphs of the functions  $\delta_j$ . The next section presents a Boolean operator that can be used to reduce the sizes of the graphs used in the term  $\chi(\delta(s, p))$ .

**4.2.2 The Operator “Restrict”.** In the equation 9 and 10, whenever  $\chi_A(s) = 0$  the characteristic function of  $Pre(Q, A, B)$  is also equal to 0. This means that, in the term  $\chi_B(\delta(s, p))$  that occurs in the equations 9 and 10, the transition function  $\delta$  can be replaced with its restriction to the domain  $A$ .

The “Restrict” operator, noted “ $\Downarrow$ ”, takes as input the typed decision graph of a boolean function  $f$  and of the characteristic function  $c$  of the set to which the function  $f$  must be restricted. The semantics of the Restrict operator [8], is given on Shannon’s canonical form in Figure 1. In this figure,  $c.root$  is the root of Shannon’s canonical form of  $c$ , and  $(c/\neg a, c/a)$  is Shannon’s expansion [1] of the function  $c$  with respect to  $a$ . The main properties of the Restrict operator [9, 10] are expressed by the following theorems.

**Theorem 1** *For any Boolean functions  $f$  and  $C \neq 0$ , if  $c(x) = 1$  then  $(f \Downarrow c)(x) = f(x)$ .*

**Theorem 2** *For any Boolean functions  $f$  and  $c \neq 0$ , Shannon’s typed canonical form of  $(f \Downarrow c)$  has at most the same number of vertices as that of  $f$ .*

Theorem 2 is not true for typed decision graphs. It can happen that the graph of  $(f \Downarrow c)$  has more vertices than the graph of  $f$ . In this case the function `restrict` returns the graph of  $f$ . Experience shows that this case occurs very rarely.

```

function restrict( $f, c$ );
  if  $c = 0$  then error;
  if  $c = 1$  then return  $f$ ;
  if  $f = 0$  or  $f = 1$  then return  $f$ ;
  let  $a = c.root$  in {
    if  $c/\neg a = 0$  then return restrict( $f/a, c/a$ );
    if  $c/a = 0$  then return restrict( $f/\neg a, c/\neg a$ );
    if  $f/\neg a = f/a$  then return restrict( $f, c/\neg a \vee c/a$ );
    return ( $\neg a \wedge \text{restrict}(f/\neg a, c/\neg a)$ )  $\vee$  ( $a \wedge \text{restrict}(f/a, c/a)$ );
  }
}

```

Figure 1. The Restrict Operators on Shannon’s canonical form.

### 4.3 Performing the “*Img*” Operation

The problem addressed here is to compute the characteristic function  $\chi$  of the image of the restrictions of a vectorial Boolean functions  $F = [f_1 \dots f_n]$  to a domain defined by its characteristic function  $\chi_A$ . A definition of  $\chi$  is [8]:

$$\chi(y_1, \dots, y_n) = (\exists x \chi_A(x) \wedge (\bigwedge_j (y_j \Leftrightarrow f_j(x)))).$$

The term  $(\bigwedge_j (y_j \Leftrightarrow f_j(x)))$  represents the transition relation of the machine, which has been shown to be in many cases too complex to be computed [8]. This section presents two algorithms that can be used to compute  $\chi$  without computing this term.

Both algorithms are bases on the “constrain” operator, noted “ $\downarrow$ ”, and work in two steps [8]. The fist step common to both algorithms consists in computing a new vectorial function  $F' = [f'_1 \dots f'_n]$  such that  $\text{Img}(F, \chi_A) = \text{Img}(F', 1)$ . The second step then consists in computing the characteristic function of  $\text{Img}(F, 1)$ , by using co-domain partitioning in the first algorithm and domain partitioning in the second algorithm.

Fig. 4.3 gives the semantics of the operator “ $\downarrow$ ” [9] on Shannon’s canonical form. The operator applies on Shannon’s canonical forms of the Boolean functions  $f$  and  $c$ , and produces Shannon’s canonical form of the function  $(f \downarrow c)$ . Its fundamental properties are expressed by the following theorem [9].

**Theorem 3** Let  $F = [f_1 \dots f_n]$  be a vector of functions, and  $c$  be a function different from 0. Let  $F \downarrow c =_{\text{def}} [(f_1 \downarrow c) \dots (f_n \downarrow c)]$ . Then  $\text{Img}(F \downarrow c, 1) = \text{Img}(F, c)$ .

**4.3.1 Co-domain Partitioning Based Algorithm.** The first recursive algorithm that computes  $\text{Img}(F, 1)$  uses the operator “ $\downarrow$ ” to partition

```

function cnst( $f, c$ );
  if  $c = 0$  then error;
  if  $c = 1$  then return  $f$ ;
  if  $f = 0$  or  $f = 1$  then return  $f$ ;
  let  $a = c.root$  in {
    if  $c/\neg a = 0$  then return cnst( $f/a, c/a$ );
    if  $c/a = 0$  then return cnst( $f/\neg a, c/\neg a$ );
    if  $f/\neg a = f/a$  then return ( $\neg a \wedge \text{cnst}(f, c/\neg a)$ )  $\vee$  ( $a \wedge \text{cnst}(f, c/a)$ );
    return ( $\neg a \wedge \text{cnst}(f/\neg a, c/\neg a)$ )  $\vee$  ( $a \wedge \text{cnst}(f/a, c/a)$ );
  }
}

```

Figure 2. The Constrain Operator on Shannon's canonical form.

the co-domain of the vectorial function  $F$ . The algorithm is a great application of the following theorem.

**Theorem 4** *Let  $F_n = [f_1 \cdots f_n]$  be a Boolean vectorial function. Then:  $\text{Img}(F_n, 1) = \text{Img}(F_{n-1} \downarrow \neg f_n, 1) \times \{0\} \cup \text{Img}(F_{n-1} \downarrow f_n, 1) \times \{1\}$ .*

The number of recursions needed to compute  $\text{Img}(F, 1)$  is bound by the number of elements of this set. Several techniques have been proposed to reduce this number of recursions. Vector partitioning [9] consists in splitting the vector  $F$  into several sub-vectors of functions which have disjoint supports of variables. In [6] it has been proposed to use a cache where partial results obtained during previous recursions are stored. However the exact matching used in [6] can be replaced by an extended matching that allows us to match any vector of  $k$  components in the cache with  $(k! \times 2^k)$  vectors, with a complexity in  $O(k \log k)$ . This extended matching test is based on the following properties.

**Theorem 5** *If  $\chi$  is the characteristic function of  $\text{Img}([f_1 \cdots f_k], A)$ , then the characteristic function of  $\text{Img}([\varepsilon_1(f_1) \cdots \varepsilon_k(f_k)], A)$ , where  $\varepsilon_j$  is the identity or the negation, is  $\chi(\varepsilon_1(y_1), \dots, \varepsilon_k(y_k))$ .*

**Theorem 6** *If  $\chi$  is the characteristic function of  $\text{Img}([f_1 \cdots f_k], A)$ , then the characteristic function of  $\text{Img}([f_{\sigma(1)} \cdots f_{\sigma(k)}], A)$ , where  $\sigma$  is a permutation of the  $k$  first integers, is  $\chi(y_{\sigma(1)}, \dots, y_{\sigma(k)})$ .*

**4.3.2 Domain Partitioning Based Algorithm.** The algorithm based on domain partitioning is a direct application of Theorem 7. The techniques described above can also be used to reduce the number of recursions needed to compute the result. This number is bounded by  $\prod_j |f_j|$ , but we think that it is directly related to  $|F|$ , where  $|F|$  is the number of vertices in the graphs that represent  $F$ . Note that this algorithm does not create any vertex. Except for characteristic functions.

$\mathcal{M}$	#reg	#in	depth	#valid	$t_{codp}$	$t_{dp}$
s838	32	35	17	17	2.4	2.5
mclc	11	11	14	35	2.3	2.6
scf	8	27	16	115	6	5.8
s298	14	3	19	218	2.9	2.8
s713	19	35	7	1544	81.7	88.2
s344	15	09	7	2625	32.7	28.6
s382	21	03	151	8865	128	79
s444	21	03	151	8865	126	75.4
cbp16	16	17	2	6.5 E4	1.2	1
cbp32	32	33	2	4.3 E9	5.7	4.5
key	56	62	2	7.2 E16	15.5	4.6
stage	64	113	2	1.8 E19	> 10000	1242
sbc	28	40	10	1.5 E5	> 10000	3530
sync	21	4	20	1469	140	154
clm1	33	13	396	3.8 E5	1227	2620
clm2	32	382	279	3.3 E6	8520	> 10000
mm10	30	13	4	1.8 E8	834	32
mm20	60	23	4	1.9 E17	> 10000	212
mm30	90	33	4	2 E26	> 10000	760

Table 1. Valid States Computation.

**Theorem 7** Let  $F = [f_1 \cdots f_n]$  be a Boolean vectorial function. Then:  
 $Img([f_1 \cdots f_n], 1) = Img([f_1 / \neg a \cdots f_n / \neg a], 1) \cup Img([f_1 / a \cdots f_n / a], 1)$ .

## 5. Experimental Results - Discussion

The algorithms are written in LISP, and the CPU times in seconds are for a BULL DPX5000 mini computer. Figure 3 gives the CPU times needed to compute the set of valid states of some digital circuits. For all circuits, #in is the number of inputs, #reg the number of state variables, depth is the number of iterations, #valid is the number of valid states ,  $t_{codp}$  and  $t_{dp}$  are the CPU times for the algorithm based on co-domain partitioning and for the algorithm based on domain partitioning respectively.

There are circuits that can be treated by only one of the two algorithms. The circuit clm2 can be treated only with co-domain partitioning: at each step during the computation, only a few states are reached. The MinMax [9] circuits mm20 and mm30 can be treated only with domain partitioning: the hit ratio in the cache is very high for this algorithm, which is not the case for the other algorithm, and the number of states that are reached at each step is very large.

The symbolic verification algorithm of temporal properties has been applied to the machines *clm1* and *sync*. The property to be proved valid on *clm1* required the computation of only one fixed point that took 38 steps and 4000s of CPU time to be obtained. The restrict operator was very useful since it reduced the graphs used to compute the term  $\chi_B(\delta(s, p))$  in such a way that during the iteration, none of them had more than 186 vertices, while some of the graphs that represent the transition function of *clm1* have more than 2500 vertices. The property to be verified on *sync* was  $Init \models AG(OK = 1)$ . The CPU time needed to make this verification was 4160 seconds and the fixed point was found in 9 steps. The restrict operator was not very useful.

Note that the property  $AG(OK = 1)$  can be verified using the algorithm presented in Section 2.3 [10], and the verification times is then the time needed to compute the valid states of *sync*, which is very much smaller. This is quite understandable since the “*Pre*” operation is intrinsically more complex than the “*Img*” operation [10].

## 6. Conclusion

In this paper we have shown that the two kinds of verification that are needed to design correct sequential circuits can be treated in a unified framework. We have presented verification algorithms that manipulate sets of inputs represented by the typed decision graphs of their characteristic functions.

Though these algorithms are very efficiently they do not allow us to deal directly with the complex circuits designed at BULL. This means that some techniques are still needed to exploit the full power of this kernel. These techniques control and data, and the decomposition of the verification task according to the circuit structure.

## Acknowledgements

The author would like to thank Gary Hachtel who saved our paper from the waste basket, and sent us benchmark circuits.

## References

- [1] J. P. Billion, “Perfect Normal Forms for Discrete Functions”, BULL Research Report N 87019, March 1987
- [2] J. P. Billion, J. C. Madre, “Original Concepts of PRIAM, an Industrial Tool for Efficient Formal Verification of Combinational Circuits”, in *The Fusion of Hardware Design and Verification*, G. J. Milne Editor, North Holland, 1988.
- [3] S. Bose, A. Fisher, “Automatic Verification of Synchronous Circuits Using Symbolic Logic Simulation and Temporal Logic”, in *Proc. of the IFIP international workshop, Applied Formal Methods for Correct VLSI Design*, leuven, November 1989.
- [4] R. E. Bryant, “Graph-based Algorithms for Boolean Functions Manipulation”, *IEEE Transactions on Computer*, Vol C35, 1986.

- [5] S. Burch, E. M. Clarke, K. L. McMillian, "Symbolic Model Checking:  $10^{20}$  states and Beyond", in *Proc. of LICS*, 1990.
- [6] H. Cho, G. Hachtel, S. W. Jeong, B. Plessier, E. Schwartz, F. Somenzi, "ATPG Aspect of FSM Verification", in *Proc. of ICCAD*, Santa-Clara, USA., June 1990
- [7] E. M. Clarke, O. Grumbreg, "Research on Automatic Verification of Finite-State Concurrent Systems", *Annual Revue Computing Science*, vol. 2, pp 269-290, 1987.
- [8] O. Coudert, C. Berthet, J. C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution", in *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989.
- [9] O. Coudert, C. Berthet, J. C. Madre, "Verification of Sequential Machines using Boolean Functional Vectors", in *Proc. of the IFIP Internatational Workshop, Applied Formal Methods for correct VLSI Design*, Leuven, November 1989.
- [10] O. Coudert, J. C. Madre, C. Berthet, "Verifying Temporal Properties of Sequential Machines without Building their State Diagrams", in *Proc. of the Workshop on Computer Aided Verification*, Rutgers, U.S.A., June 1990.

# DYNAMIC VARIABLE ORDERING FOR ORDERED BINARY DECISION DIAGRAMS

R. Rudell

*Synopsys, Inc., Mountain View CA*

## Abstract

The Ordered Binary Decision Diagram (OBDD) has proven useful in many applications as an efficient data structure for representing and manipulating Boolean functions. A serious drawback of OBDD's is the need for application-specific heuristic algorithms to order the variables before processing. Further, for many problem instances in logic synthesis, the heuristic ordering algorithms which have been proposed are insufficient to allow OBDD operations to complete within a limited amount of memory. In this paper, I propose a solution to these problems based on having the OBDD package itself determine and maintain the variable order. This is done by periodically applying a minimization algorithm to reorder the variables of the OBDD to reduce its size. A new OBDD minimization algorithm, called the sifting algorithm, is proposed and appears especially effective in reducing the size of the OBDD. Experiments with dynamic variable ordering on the problem of forming the OBDD's for the primary outputs of a combinational circuit show that many computations complete using dynamic variable ordering when the same computation fails otherwise.

## 1. Introduction

Boolean function manipulation is an important component of many logic synthesis algorithms including logic optimization and logic verification of combinational and sequential circuits. The Ordered Binary Decision Diagram (OBDD) has proven useful in these applications as an efficient data structure for the representation and manipulation of Boolean functions. However, a serious drawback of OBDD's is the need to order the variables.

When an order is found which keeps the OBDD size manageable (e.g., less than 100,000 nodes), OBDD-based techniques perform very well. However, it is usually necessary to devise heuristics to order the variables for each OBDD application. Besides the burden this places on the programmer trying to apply OBDD's in a particular setting, there is the problem that in many instances the heuristic algorithms which have been proposed are unable to find a variable order which keeps the OBDD's small. This implies that many OBDD applications give-up (*space-out*) before an operation can be completed.

Many OBDD applications in logic synthesis begin by forming the OBDD for each primary output in a combinational circuit (in terms of the primary inputs to the circuit). For many problem instances, choosing a random order for the variables leads to OBDD's which are too large. For example, it is not possible

to form the OBDD's for 23 of 35 large circuits from the IWLS'91 benchmark set when using a random variable order. Heuristic ordering algorithms, such as the *depth-first heuristic* algorithm and its variations ([10, 5, 11]), are a significant improvement over random variable ordering. However, it is still not possible to form the OBDD's for 11 of 35 large circuits from the IWLS'91 benchmark set when using this heuristic order. It has remained an open problem whether this is a limitation of the variable ordering algorithms or simply the inherent exponential worst case complexity of the OBDD representation.

This paper describes a general paradigm to improve the robustness of any OBDD package by using automatic variable re-ordering. First I review the necessary details of an OBDD package and then describe the dynamic variable ordering strategy. Two OBDD minimization algorithms are presented, including a new algorithm called the sifting algorithm. Experimental results are given for these algorithms which demonstrate the utility of dynamic variable ordering when applied to the problem of forming OBDD's for the primary outputs in a combinational logic network. The last section provides directions for future improvements of this technique.

## 2. OBDD Implementation Review

I assume that the reader is familiar with Ordered Binary Decision Diagrams as introduced by Bryant [3]. In the paper by Brace, Rudell and Bryant [2], details for an efficient implementation of a OBDD package were outlined. I review here some details necessary for the remainder of the paper.

A multi-rooted (shared) directed acyclic graph (DAG) is used to represent a set of Boolean functions. Each node in the DAG represents a Boolean function  $F$  and has an associated variable  $x_i$  and pointers to two other nodes (functions) in the DAG. The node  $F$  is written as the tuple  $(x_i, G, H)$  where  $x_i$  is called the *top variable* of the function  $F$ ,  $G$  is the positive cofactor of  $F$  with respect to  $x_i$  ( $G = F_{x_i}$ ), and  $H$  is the negative cofactor of  $F$  with respect to  $x_i$  ( $H = F_{\bar{x}_i}$ ). The node  $F$  thus represents the function  $F = x_iG + \bar{x}_iH$ .  $G$  is also known as the THEN node and  $H$  is also known as the ELSE node. The sink nodes represent the constant functions **0** and **1**.

Ordered BDD's have a total order imposed on the variables; i.e., an order is assigned to each variable, and the variables must appear in ascending order along every path in the OBDD. Because the BDD is ordered, the DAG can be levelized with all nodes with a particular top variable at a given level. Level  $i$  refers to all nodes with a top variable  $x_i$ .

A global hash table, called the *unique table*, allows a node of the DAG,  $(x_i, G, H)$ , to be found in constant time. A hash function is computed on the tuple  $(x_i, G, H)$  which provides an index into an array of bins which store the first DAG node for that hash value. All of the nodes with the same hash value

(i.e., collisions) are stored in a linked list. Each node of the DAG occupies 4 words: the variable index plus other flags, a pointer to the THEN node, a pointer to the ELSE node, and a pointer to the next node on the collision chain for the unique table.

A global cache, called the *computed table*, is used as a memory function for the recursive algorithms which operate on the DAG. This table, implemented as a hash-based cache, stores the results of recursive operations such as ITE, but overwrites an entry when a collision occurs, rather than using a link-chain to resolve collisions. Each hash-based cache entry occupies 4 words: 3 words which form the key for the operation (e.g., ITE(F,G,H) uses F, G, and H as the key) and a single word which is the operation result. A ratio of one cache entry is maintained for every four unique table entries so that the total memory usage of the package, including all overhead, is approximately 24 bytes per DAG node on a 32-bit machine.

The OBDD package uses garbage collection to recycle memory. A reference count is maintained for each node in the DAG, and a count of the number of dead (i.e., unreferenced) nodes in the DAG is maintained as nodes are created and freed. Dead nodes cannot be freed immediately because an entry from the computed table may point to the node; these references are not included in the reference count because the computed table entries are never deleted. During a recursive operation such as ITE, a *find or add* operation is performed to either find a node in the DAG or to create a new node if the given node does not exist. If a new node is created causing the unique table to become too full, then either a garbage collection is performed if there are enough dead nodes in the DAG to make it worthwhile, or the unique table array is increased in size by a factor of two.

The key drawback to this package, which this paper addresses, is that the ordering of the variables is specified in advance by the user. No assistance is offered to help order the variables, and the order cannot be subsequently changed.

### 3. Dynamic Variable Ordering

In this paper, I propose a general paradigm for maintaining variable orders in an OBDD. The idea is to have the OBDD package determine and maintain the variable order of the OBDD. This variable order is changed automatically by the OBDD package, transparently to the user, as operations are performed. Because the variable order within the OBDD is no longer static, this technique is referred to as **dynamic variable ordering**.

Dynamic variable ordering differs from the typical use of OBDD's where the variables are ordered once when the OBDD is created and the order is maintained throughout all subsequent processing. It also differs slightly from other proposed variable ordering schemes in that the re-ordering is not performed at the

explicit request of the user. Instead, the package determines appropriate points at which to stop processing, choose a new order, and then resume processing.

When using dynamic variable ordering, a total order is defined for all variables before and after each package operation; however, the order is periodically adjusted by the OBDD package, as a consequence of an operation, to find a better order. Logically, the variable order changes in-between package operations. Thus, we maintain all advantages provided by the ordered BDD data structure, such as canonicity and efficient recursive algorithms.

A well-designed interface to an OBDD package hides all details of the OBDD data structure. The programmer simply creates the variables and then uses package operations such as AND, OR, and NOT to form new functions. This allows dynamic variable ordering to be applied transparently to the user of the OBDD package.

There are two goals we hope to achieve with dynamic variable ordering. The first goal is to allow OBDD operation sequences which fail when using a fixed heuristic variable order to succeed when a new order is chosen mid-stream. The second goal is to reduce the need for the heuristic ordering algorithms – i.e., problems which complete with a heuristic variable order should also complete when starting from a random variable order. If we can deliver on these goals, the advantage of dynamic variable ordering to the user of the OBDD package is clear.

One implementation of dynamic variable ordering is as follows. At each garbage collection within the OBDD package, which is triggered based on the growth of the number of nodes in the DAG, a variable-reordering algorithm is applied to the OBDD to reduce the OBDD size. Any variable ordering algorithm can be applied at this step, but the algorithms which are used must be efficient because they will be applied repeatedly as OBDD processing proceeds. Of course, the algorithms must also be effective in finding a better variable order for the OBDD.

Dynamic variable ordering motivates the exploration of algorithms for OBDD minimization; i.e., reducing the size of all functions simultaneously represented by a multi-rooted OBDD by changing the variable order. Two algorithms for OBDD minimization are considered in the next section.

## 4. Variable Reordering Algorithms

Many people, including Brace [1], Fujita et al. [6], and Ishiura et al. [7] have made the observation that swapping the order of two adjacent variables in an OBDD affects only the DAG nodes at the two levels; all other nodes remain unchanged. In this section, I describe how to implement this operation so that its complexity is proportional to the number of nodes at the particular level of the DAG and independent of the size of the entire DAG. This efficient adjacent

variable swap forms the core for many OBDD minimization algorithms. I then describe the *window permutation algorithm* and the *sifting algorithm* for minimizing the size of the OBDD.

## 4.1 Efficient Variable Swap

There are two problems with making the variable swap of  $x_i$  and  $x_{i+1}$  have local complexity. The first is that we need to find all nodes at level  $i$  without walking the entire DAG starting from the roots. The second is that each node in the DAG must represent the same function before and after the variable swap to avoid patching any references to that node.

A memory-efficient scheme to find all nodes at level  $i$  replaces the single unique table with an array of hash tables, one per level of the DAG. The variable index is used to locate the hash table which stores all nodes for that level of the DAG. The hash table has an array of bins which store the first node for each hash value for this level. Hence, all nodes at a given level can be visited by walking the collision chain which starts at each hash table array position.

A node  $F$  at level  $i$  can be pointed to by other nodes above it in the DAG, and by functions which have already been returned to the user. To reduce memory, back-pointers are not maintained. Hence, there is no way to reach all references to node  $F$  without walking the entire DAG. Therefore, to perform a local variable swap, it is necessary to maintain an identical logical function at each node. This is done by overwriting the node representing  $F$  with the new node which results from the modification to the variable order. This is done as follows.

Let  $F = (x_i, F_1, F_0)$  be a node at level  $i$ . Let  $F_{11}$  be the cofactor of  $F_1$  with respect to  $x_{i+1}$ . Computing this cofactor is trivial: the result is either the THEN node pointed to by  $F_1$  (if  $x_{i+1}$  is the top variable of  $F_1$ ) or  $F_1$  (otherwise). Similarly, let  $F_{10}$  be the negative cofactor of  $F_1$ , and let  $F_{01}, F_{00}$  be the two cofactors of  $F_0$ . Node  $F$  is overwritten with the tuple  $(x_{i+1}, (x_i, F_{11}, F_{01}), (x_i, F_{10}, F_{00}))$ . Expansion of this formula shows that it preserves the function of node  $F$  and inspection ensures that the new variable order (i.e.,  $x_{i+1}$  is above  $x_i$ ) is established for all paths through  $F$ .

The new nodes required at level  $i$  (i.e.,  $(x_i, F_{11}, F_{01})$  and  $(x_i, F_{10}, F_{00})$ ) may be degenerate nodes (e.g., in the case that  $F_{11} = F_{01}$ ), or may already exist in the DAG as required to implement other functions. When  $F$  is re-expressed as a result of the variable swap, the DAG's rooted at  $F_1$  and  $F_0$  can be freed. Note, however, that the nodes  $F_{00}, F_{01}, F_{10}, F_{11}$  all have references after the variable swap, so that only the root nodes  $F_1$  and  $F_0$  can be freed as a result of the swap. To be specific, node  $F_1$  can be freed if the only reference to  $F_1$  previously came from node  $F$ .

We can make use of this observation to perform incremental garbage collection during the variable swap. Before OBDD minimization is applied, a garbage

collection is performed and the computed table is cleared. Thereafter, nodes at level  $i + 1$  can be deleted incrementally if they have no other reference beside the reference from level  $i$ .

Attributed edges have been proposed by, among others, Karplus [8], Madre and Billon [9], and Minato et al. [11]. The edges in the OBDD are tagged to indicate a modification of the referenced function. This reduces the size of the DAG by allowing a single node to represent several different functions. The most popular and useful attribute is the *negate-output edge*, although other attributes, such as *negate-input edge* [11] and *negate-else edge* [4] have been proposed. The inclusion of attributed edges is usually transparent to the algorithms which operate on the OBDD. In particular, the complexity of an adjacent variable swap is unaffected by the inclusion of negate-output edges. However, inclusion of either negate-input or negate-else edges appears to destroy the local complexity of a variable swap by requiring pointers to a modified node to be changed. (More details can be found in [12].) One impact of dynamic variable ordering, then, is that these last two edge attributes cannot be used.

## 4.2 Window Permutation Algorithm

Fujita et al. [6] and Ishiura et al. [7] presented similar heuristic algorithms for minimizing the size of a OBDD using adjacent variable exchange. I refer to this algorithm as the *window permutation algorithm*..

The window permutation algorithm proceeds by choosing a level  $i$  in the DAG and exhaustively searching all  $k!$  permutations of the  $k$  adjacent variables starting at level  $i$ . This is done using  $k! - 1$  pairwise exchanges followed by up to  $k(k - 1)/2$  pairwise exchanges to restore the best permutation seen. This is then repeated starting from each level until no improvement in the DAG size is seen. Figure 1 shows the variable permutations which are explored when applying a window of size  $k = 3$  starting at variable  $x_2$ . Six permutations are explored with 5 adjacent variable swaps, and then 3 additional variable swaps (worst case) are used to restore the best permutation.

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	initial
$x_1, x_3, x_2, x_4, x_5, x_6, x_7$	swap ( $x_2, x_3$ )
$x_1, x_3, x_4, x_2, x_5, x_6, x_7$	swap ( $x_2, x_4$ )
$x_1, x_4, x_3, x_2, x_5, x_6, x_7$	swap ( $x_3, x_4$ )
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap ( $x_3, x_2$ )
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap ( $x_4, x_2$ )
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap ( $x_4, x_3$ )
$x_1, x_3, x_2, x_4, x_5, x_6, x_7$	swap ( $x_2, x_3$ )
$x_1, x_3, x_4, x_2, x_5, x_6, x_7$	swap ( $x_2, x_4$ )

Figure 1. Window Permutation Example.

Marking can be used to record when a variable exchange at a given level may be profitable. A level is marked after the permutation at the level is known optimal. This mark is reset when a new permutation is determined for any of the preceding  $k - 1$  levels; when all levels in the DAG are marked, the window permutation algorithm cannot further improve the DAG size).

Because the swap of two adjacent variables is efficient, the window permutation algorithm remains practical for values of  $k$  as large as 4 or 5. However, results presented in Section 5.2 and Section 5.3 indicate that the window permutation algorithm is limited in its ability to find good variable orders.

### 4.3 Sifting Algorithm

I propose a new OBDD minimization algorithm in this paper which I call the *sifting algorithm*. This algorithm is based on finding the optimum position for a variable, assuming all other variables remain fixed. If there are  $n$  variables in the DAG (excluding the constant level which is always at the bottom), then there are  $n$  potential positions for a variable, including its current position. Among these  $n$  positions, the subgoal employed by the sifting algorithm is to find the spot which minimizes the size of the DAG.

Ideally, we could find the best position for a variable assuming all other variables remain fixed with a low-complexity analysis of the OBDD. However, this does not appear possible. Therefore, the optimum position for a variable is determined by brute-force enumeration as follows. The variable is exchanged with its successor variable until the variable becomes the next to last variable in the DAG; i.e., the variable is sifted down to the bottom of the DAG. Then the variable is exchanged with its predecessor variable until the variable becomes the top variable in the DAG; i.e., the variable is sifted up to the top of the DAG. The best DAG size seen during this search is remembered and the position of the variable is restored by moving the variable from the top position down to its optimum position. Figure 2 shows the variable permutations which are explored when applying the sifting algorithm to variable  $x_4$ . The 7 positions for variable  $x_4$  are explored using 9 adjacent swaps, and the optimum position is restored with an additional 6 swaps (worst-case).

The sifting algorithm proceeds as follows. The variables are sorted into decreasing size based on the number of nodes at each level of the DAG. Then each variable is moved to its locally optimum position assuming that all other variables remain fixed. Each variable is moved only once in this process, although the algorithm could be iterated to convergence.

The sift algorithm has the advantage that a variable can move a long distance in the ordering. Note that the DAG-size can increase significantly after the first few variable swaps, and then eventually reduce below the starting point. This allows a type of up-hill move to be taken – the acceptance of the entire sequence

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	initial
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	swap ( $x_4, x_5$ )
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	swap ( $x_4, x_6$ )
$x_1, x_2, x_3, x_5, x_6, x_7, x_4$	swap ( $x_4, x_7$ )
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	swap ( $x_7, x_4$ )
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	swap ( $x_6, x_4$ )
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap ( $x_5, x_4$ )
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap ( $x_3, x_4$ )
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap ( $x_2, x_4$ )
$x_4, x_1, x_2, x_3, x_5, x_6, x_7$	swap ( $x_1, x_4$ )
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap ( $x_4, x_1$ )
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap ( $x_4, x_2$ )
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap ( $x_4, x_3$ )
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	swap ( $x_4, x_5$ )
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	swap ( $x_4, x_6$ )
$x_1, x_2, x_3, x_5, x_6, x_7, x_4$	swap ( $x_4, x_7$ )

Figure 2. Sifting Algorithm Example.

of pairwise swaps is based on the best position seen regardless of any increase in the intermediate DAG size. A limitation of the window permutation algorithm appears to be that several moves can be required to move a variable a long distance and these moves can be blocked by an intermediate up-hill move.

The sift algorithm requires  $O(n^2)$  swaps of adjacent levels in the DAG, and each of these variable swaps has complexity proportional to the width of the DAG. To control the worst-case complexity, the search in a particular direction is terminated if the DAG size grows to twice its original size.

## 5. Experimental Results

The IWLS'91 benchmark set includes a directory of 76 combinational circuits (`cmlexamples`) and 40 sequential circuits (`smlexamples`). This includes the ISCAS'85 and ISCAS'89 benchmarks. Because most of these circuits are trivially small, I focus here on the 35 largest multiple-level examples. All DAG-sizes are given in thousands of nodes, and the run-times are measured on a Sun Microsystems SparcStation-10 Model 41.

Due to space limitations, only summary results are presented in this paper. Complete tables of results appear in [12].

### 5.1 Random Orders vs. Heuristic Orders

The first experiment was to form the OBDD's for all primary outputs. The same variable order was used for all of the primary outputs. The variable order was first determined with a single random trial, and then using the depth-first heuristic ordering algorithm.

When the maximum DAG size was set to 100,000 nodes (2.4 mB memory), it was not possible to form the OBDD's for 23 of the 35 circuits when using a random variable order, and it was not possible to form the OBDD's for 11 of the 35 circuits when using the depth-first heuristic ordering algorithm. The 11 circuits which failed when using a depth-first ordering were *c2670*, *c3540*, *c6288*, *c7552*, *i10*, *mm9a*, *mm9b*, *mm30a*, *s9234.1*, *s15850.1*, and *s38417*.

When the maximum DAG size was increased to 1,000,000 nodes (24 mB memory), the random order failed for 13 of the 35 circuits and the heuristic order failed for 7 of the 35 circuits. The 4 additional circuits which completed when given more memory were *c3540*, *i10*, *s9234.1*, and *s15850.1*.

## 5.2 OBDD Minimization Comparison

The next experiment compares the window permutation algorithm against the sift algorithm for OBDD minimization. The OBDD's for 24 of the 35 examples can be formed using the heuristic ordering algorithm and a 100,000 node limit. These 24 circuits were minimized after the OBDD's had been formed with the window permutation algorithm for  $k = 2, 3, 4, 5$  and the sifting algorithm. The relative DAG-size and CPU ratios for each minimization algorithm is given in Figure 3.

Algorithm	size	cpu
No minimization	1.00	1.00
Window, $k=2$	0.81	1.19
Window, $k=3$	0.72	1.49
Window, $k=4$	0.70	2.83
Window, $k=5$	0.67	9.19
Sift	0.55	3.84

Figure 3. Minimization Comparison.

The sift algorithm results in OBDD's which are 45% <sup>1</sup> smaller than the heuristic order, while the window permutation algorithm with  $k = 4$  produces OBDD's which are only 30% smaller. The sift algorithm produces OBDD's which are 20% smaller than the window permutation algorithm ( $k = 4$ ) at the cost of an additional 40% in run-time.

## 5.3 Dynamic Variable Ordering

The next experiment compares the window permutation algorithm and the sifting algorithm in the context of dynamic variable ordering. For this experiment, I focused on the 11 examples which cannot complete with the heuristic order. Dynamic variable ordering was performed by applying the corresponding OBDD minimization algorithms at each garbage collection, and the measurement criteria was to see if the examples could complete. The results are given in Fig-

ure 4 where FAIL refers to the number of primary outputs which failed to have their OBDD formed.

	No Dynamic Ordering		Window Algorithm k=4		Sift Algorithm	
	Example	SIZE	FAIL	SIZE	FAIL	SIZE
C2670	>100	2	>100	2	6.6	0
C3540	>100	2	>100	7	27.2	0
C6288	>100	22	>100	22	>100	21
C7552	>100	4	>100	4	8.2	0
i10	>100	7	>100	6	41.2	0
mm9a	>100	7	4.4	0	2.0	0
mm9b	>100	7	5.2	0	2.5	0
mm30a	>100	60	>100	60	17.6	0
s9234.1	>100	7	>100	6	4.5	0
s15850.1	>100	8	54.0	0	17.5	0
s38417	>100	532	>100	414	>100	203

Figure 4. Results for 11 Hard Examples.

Dynamic variable ordering using the window permutation algorithm ( $k=4$ ) was able to complete 3 of the 11 failed examples (*mm9a*, *mm9b*, and *s15850.1*). Dynamic variable ordering using the sifting algorithm was able to complete 9 of the 11 failed examples. Only *c6288* and *s38417* still fail with the 100,000 node limit. Even for these two failed examples, fewer outputs failed when the sifting algorithm was used.

## 5.4 Performance Impact

The last experiment compares the run-time for the 35 largest circuits without dynamic variable ordering, with dynamic variable starting from the heuristic variable order, and with dynamic variable ordering starting from a randomly generated variable order. The results are summarized in Figure 5. The sift algorithm was used when performing dynamic variable ordering.

Algorithm	size	cpu
No minimization	1.00	1.00
DVO/Heuristic	0.56	6.80
DVO/Random	0.58	11.80

Figure 5. Performance Results.

For the 24 examples which complete both with and without dynamic variable ordering, the run-time was increased an average of 6.8 when using dynamic variable ordering starting from the heuristic order. The average OBDD size for

these 24 examples was reduced by a factor of 1.8 (45%) when dynamic variable ordering was used.

Figure 5 also compares dynamic variable ordering starting from the heuristic order (DVO/heuristic) to dynamic variable ordering starting from a randomly generated variable order (DVO/random). The sifting algorithm was still able to complete for all but 3 examples (*c6288, mm30a, and s38417*). Interestingly, the DAG sizes starting from the random order were only slightly larger than the DAG sizes when starting from the heuristic order while the run-time increased by almost a factor of 2.

## 5.5 Summary

The sift algorithm is superior to the window permutation algorithm, both as a static OBDD minimization algorithm and in the application of dynamic variable ordering. The sift algorithm consistently produces smaller OBDD's than the window permutation algorithm, although it has run-times which are longer.

The application of the sift algorithm in conjunction with dynamic variable ordering allows 9 of the 11 combinational circuits which could not complete using a static variable order to complete.

When a random variable order was used as the starting point rather than the heuristic variable order, OBDD processing was still able to complete for 32 of the 35 largest examples, including 8 of the 11 difficult examples. While some improvement still exists when starting from the heuristic order, for most examples a random order does not affect the ability of the OBDD processing to complete.

These results indicate that the goals of completing more computations and reducing the dependence on the need for heuristic ordering algorithms have been achieved for this application.

## 6. Conclusions

This paper proposes a modification to an OBDD package whereby the OBDD package owns and maintains the order of the variables. At each garbage collection, an algorithm is applied on the OBDD to reorder the variables so as to reduce the number of nodes in the OBDD. Two OBDD minimization algorithms were tried: the window permutation algorithm and the sifting algorithm. Little benefit was seen from the application of the window permutation algorithm for  $k = 4$ . Dynamic variable ordering using the sifting algorithm was able to complete several OBDD operations which were not able to complete without dynamic variable ordering, and the resulting OBDD tended to be significantly smaller. In almost all cases, dynamic variable ordering was able to complete the OBDD operation sequences even when starting from a random variable order rather than a heuristically determined variable order. The drawback of dynamic variable ordering is that the runtime for the OBDD operations increases significantly.

## 7. Future Directions

One direction for this work is to investigate dynamic variable ordering when applied to other OBDD problems. For example, the fixed-point algorithm found in sequential verification has the problem that determining a good variable order *a priori* for both the transition relation OBDD and the state space OBDD is difficult. It would be interesting to see if dynamic variable ordering could improve the efficiency and application of these algorithms.

The utility of dynamically determining the variable order for the OBDD has been demonstrated; however, the run-time impact is very large. One idea to improve the sifting algorithm would be to devise a more efficient algorithm to determine the optimum position for a variable (assuming all other variables remain fixed). Another idea would be to explore exact bounding techniques that determine when a search in a particular direction can be terminated. Also, exploring completely different algorithms for OBDD minimization is a possibility.

Finally, it would be interesting to determine bounds on the growth of the OBDD as a single variable is moved  $\pm k$  positions.

## 8. Acknowledgements

Discussions with Karl Brace in 1989 led to the observation that only local operations were needed to exchange two variables in the OBDD and that all nodes at a given level of the OBDD could be reached efficiently at no cost. I would like to thank Jerry Burch and David Long for several discussions on the manual techniques they use to find good variable orders. Their suggestions motivated the sifting algorithm.

## Notes

1. Averages for the benchmark set are computed as the arithmetic mean of the ratio computed for each example.

## References

- [1] K. Brace. Personal Communication, June 1989.
- [2] K. Brace, R. Bryant, and R. Rudell. *Efficient Implementation of a BDD Package*. In *Proceedings 27th Design Automation Conference*, June 1990.
- [3] R. E. Bryant. *Graph-based Algorithms for Boolean Function Manipulation*. *IEEE Trans. Comp.*, C-35(8):677–691, August 1986.
- [4] J. Burch. Personal Communication, June 1992.
- [5] M. Fujita, H. Fujisawa, and N. Kawato. *Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams*. In *Proceedings International Conference on Computer-Aided Design*, pages 2–5, November 1988.

- [6] M. Fujita, Y. Matsunaga, and T. Kakuda. *On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis*. In *Proceedings European Design Automation Conference*, pages 50–54, March 1991.
- [7] N. Ishiura, H. Sawada, and S. Yajima. *Minimization of Binary Decision Diagrams Based on Exchanges of Variables*. In *Proceedings International Conference on Computer-Aided Design*, pages 472–475, November 1991.
- [8] K. Karplus. *Representing Boolean Functions with If-Then-Else DAGs*. Computer Engineering UCSC-CRL-88-28, UC Santa Cruz, December 1988.
- [9] J.-C. Madre and J.-P. Billon. *Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behavior*. In *Proceedings 25th Design Automation Conference*, pages 205–210, June 1988.
- [10] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. *Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment*. In *Proceedings International Conference on Computer-Aided Design*, pages 6–9, November 1988.
- [11] S. Minato, N. Ishiura, and S. Yajima. *Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation*. In *Proceedings 27th Design Automation Conference*, June 1990.
- [12] R. Rudell. *Dynamic Variable Ordering for Ordered Binary Decision Diagrams*. Technical Report, Synopsys, Inc., 700 E. Middlefield Road, Mountain View, CA 94043, January 1993.

# VERIFICATION OF LARGE SYNTHEZIZED DESIGNS

Daniel Brand

*IBM Research Division*

*Thomas J. Watson Research Center*

*Yorktown Heights, New York, U.S.A*

## Abstract

The problem of checking equality of Boolean functions can be solved successfully using existing techniques for only a limited range of examples. We extend the range by using a test generator and the divide and conquer paradigm.

## 1. Introduction

Since the synthesis process, whether automatic or manual, cannot be guaranteed to be error-free, there is a need for checking its correctness. That is, we have to prove the equality  $F = G$ , where  $F$  is the input into synthesis and  $G$  is the result of synthesis. The traditional approach to this problem is to build a canonical BDD representation [6] for each function and then simply compare the BDDs for identity. Many variations and improvements have been made [2, 7, 10, 14] so as to extend the range of applicability. The applicability of BDDs is limited because their memory requirements may grow exponentially with the size of the function. Therefore “running out of memory” is the usual failure mode of BDDs. This is more serious than the failure mode of “running out of time” because it is much easier to allocate more time to a problem than more memory. This has been recognized by [9] where correctness can be guaranteed with a reliability increasing with increasing CPU time.

One method of Boolean reasoning that does not explode in terms of memory is a test generator. The most straightforward approach towards proving  $F = G$  using a test generator would form the exclusive OR,  $F \oplus G$  (Figure 1), and then ask whether the output of the XOR gate is testable. If it is testable for stuck at 0 then the resulting pattern constitutes a counter-example to the assertion of functional equivalence. If it is not testable for stuck at 0 then the two functions are identical. If it is not testable for stuck at 1 then the two functions are complements of each other.

Throughout the paper we will use the term “miter” to refer to the configuration of Figure 1. In general a miter can appear in the middle of larger logic, where it is defined to consist of a two-input XOR gate, plus the symmetric set difference between the transitive fanins of the two inputs into the XOR gate. In

other words, a miter starts at an XOR gate and extends towards primary inputs till nets shared by both cones of logic.

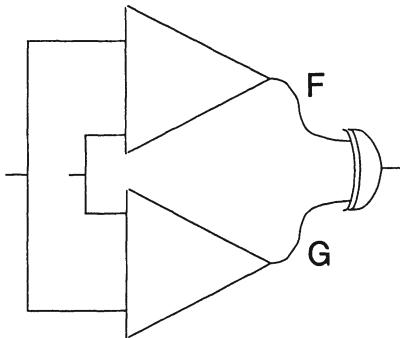


Figure 1. A miter.

The above straightforward approach is not likely to be successful because test generators tend to have a lot of difficulty with miters. While there are test generation strategies specifically designed for miters [8], they are bound to fail if the miter is big enough. The main observation of this paper is that the difficulty for a test generator is dependent mainly on the size of the miter, rather than on the size of the surrounding logic. The key to our approach is to keep applying a test generator to configurations where the size of the miter is small and independent of the size of the functions subject to verification.

One way of making the verification problem easier is to do it in stages [5]. Suppose that  $f$  is a sub-function of  $F(f)$  and  $g$  is a sub-function of  $G(g)$ . We would like to first prove  $f = g$  and then use it to simplify the proof of  $F(f) = G(g)$ . To do that we must first solve two problems.

1. It may be difficult to find such sub-functions  $f$  and  $g$  because synthesis frequently does not preserve the functionality of internal nodes. If  $f$  was optimized subject to the don't cares of the whole  $F$  then there may be no  $g = f$ .
2. Suppose that we do find  $f = g$ ; how do we take advantage of it? We cannot simply replace  $f$  and  $g$  with a new variable  $y$ . Even if  $f = g$  and  $F(f) = G(g)$  it is possible that  $F(y) \neq G(y)$  because  $F$  might have been optimized subject to the don't cares of  $f$ . This problem is referred to as "false negative", that is,  $f = g$  and  $F(y) \neq G(y)$  need not necessarily imply that  $F(f) \neq G(g)$ .

Our solution to the first problem is not to ask whether  $f = g$ , i.e., "is  $f \oplus g$  testable for stuck at 0?". Instead we insert an XOR gate between  $f$  and all its immediate fanouts and make  $g$  the other input of the XOR gate (see Figure 2). Then we check whether the output of the XOR gate is testable (in the context of the surrounding logic of  $F$ ). Suppose that it is not testable for stuck at 0. That means that for every input pattern either  $f = g$ , or  $f \neq g$ , but the difference

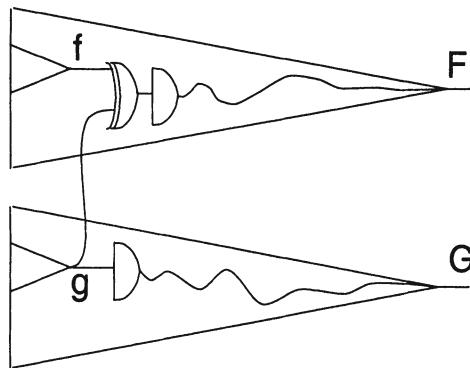


Figure 2. Can  $g$  replace  $f$  inside  $F$ ?

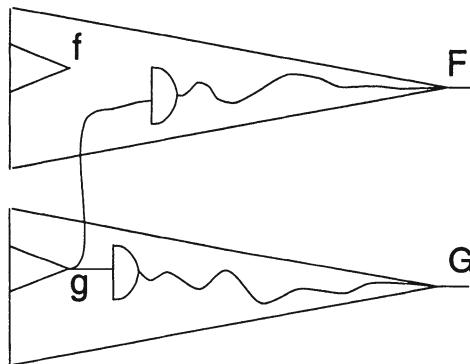


Figure 3.  $g$  replaces  $f$ .

cannot be propagated to primary outputs. In other words,  $g$  can replace  $f$  inside  $F$  without changing the functionality of  $F$ . Similarly, if the XOR gate is not testable for stuck at 1 then  $\bar{g}$  can replace  $f$ .

Our solution to the second problem is to actually perform the replacement. However, it is important that the logic of  $g$  not be copied inside  $F$ , but rather  $F$  and  $G$  must share the same copy of  $g$  (see Figure 3). Proving  $F(g) = G(g)$  is easier than proving  $F(f) = G(g)$  because the miter  $F(g) \oplus G(g)$  is smaller than the miter  $F(f) \oplus G(g)$ .

The whole process proceeds from inputs of  $F$  to its outputs. At each step we have a sub-function  $f$  and try to find a sub-function  $g$  of  $G$  that could replace  $f$ . If we find a suitable  $g$  we make the replacement, while retaining the functionality of  $F$ . As we proceed,  $F$  keeps resembling  $G$  more and more and the process

stops at the outputs of  $F$ , which should be replaceable by the outputs of  $G$ . If an output of  $G$  cannot replace the corresponding output of  $F$  then the test generator gives an input pattern constituting a counter-example.

## 2. Algorithm

As described in the introduction, the approach is based on the following

**Lemma 1** *Let  $x$  be a vector of variables and  $y$  be a single variable. Consider arbitrary functions  $f(x), g(x), F(x, y)$ . (Since all three functions depend on  $x$  we will drop  $x$  from our notation; for example, by  $F(f)$  we will mean  $F(x, f(x))$ .) The following two identities hold*

$$F(f) = F(g) \iff F(f \oplus g) = F(0) \quad (1)$$

$$F(f) = F(\bar{g}) \iff F(f \oplus g) = F(1) \quad (2)$$

Proof:

We will be done if we show that (1) and (2) are true for any input pattern  $x$ . For a given input  $x$  there are four possible combinations of values for  $f$  and  $g$ . For illustration we will show the case  $f = 0$  and  $g = 1$ ; all the other cases are similar. Substituting  $f = 0, g = 1$

in (1) we get  $F(0) = F(1) \iff F(1) = F(0)$

in (2) we get  $F(0) = F(0) \iff F(1) = F(1)$

Q.E.D.

The right hand side of (1) says that the output of the XOR is not testable for stuck at 0 and the left hand side says that in this case we may replace  $f$  by  $g$ . Similarly (2) says that if the XOR is not testable for stuck at 1 then we can replace  $f$  by  $\bar{g}$ .

We now give the algorithm for proving  $F = G$ , where both functions are given by multi-output combinational networks. The algorithm is followed by a detailed explanation.

0. for each net calculate its primary inputs
1. form a list  $fs$  of nets in  $F$  in topological order
2. for each  $f$  in  $fs$
3. form a list  $gs$  of some nets in  $G$
4. for each  $g$  in  $gs$
5. if  $F(f \oplus g) = F(0)$  then replace  $f$  by  $g$
6. if  $F(f \oplus g) = F(1)$  then replace  $f$  by  $\bar{g}$

For each of the above statements we will give an explanation as well as its computational complexity. Let the number of connections in both  $F$  and  $G$  be  $n$ . We will assume that the number of nets is also of the same order as  $n$ .

*Statement 0:* calculates information used for a heuristic selection of candidates  $g$  in statement 3. It collects information whose size can be  $O(n^2)$  in the worst case. Therefore its time and space complexity is  $O(n^2)$ .

*Statement 1:* The only nets required in  $fs$  are primary outputs; all other nets are placed there only for speed. The more internal nets are included in  $fs$ , the more frequently we need to invoke the test generator, but the easier will be the test generator's work because it will be working on smaller miters.

Our heuristic is to have the nets in  $fs$  separated by approximately two stages of logic. In our experiments the number of stages of separation did not significantly affect the performance of the algorithm, provided for any miter that is too large we consider smaller miters in its place. The topological ordering is done in linear time.

*Statement 2:* There can be at most  $n$  of the nets  $f$ .

*Statement 3:* In contrast to  $fs$  the choice of  $gs$  has a tremendous impact on performance. If the list  $gs$  were too long then we would call the test generator too many times even for nets  $f$  that cannot be replaced by any  $g$ . On the other hand, if  $gs$  were too short then we might fail to find a  $g$  to replace a particular  $f$ , which would make the test generator's job harder later on. Our heuristic forms a list  $gs$  giving preference to nets  $g$  with a name related to the name of  $f$ , followed by nets that have the same simulation result as  $f$  on a small number of random patterns, followed by all remaining nets.

The list  $gs$  is then pruned in order to minimize the number of calls to the test generator:

- We require that each  $g$  depend on no more primary inputs than  $f$  does.
- We use approximate fault simulation to discard candidates  $g$  which are not likely to replace  $f$  (similarly as in [1]).

Then the list  $gs$  is truncated after  $k$  elements; in our experiments we used  $k = 20$ .

Statement 3 has a worst case complexity  $O(n)$  because we may be forced to consider all  $n$  nets  $g$ . Since this statement is executed  $n$  times it contributes  $O(n^2)$  to the overall algorithm.

*Statement 5 and 6:* We call the test generator [11, 12] to determine whether  $g$  can replace  $f$ . If so, we make the replacement and terminate the loop of  $gs$ . This statement contributes only  $O(n)$  complexity to the algorithm because we allow the test generator to run only a fixed amount of time independent of the size of the network. For each  $f$  the test generator is called at most  $k$  times.

Thus the overall worst case complexity is  $O(n^2)$ , but we cannot guarantee to verify every design. In order to obtain such a guarantee we would have to allow an unbounded amount of test generation time, in which case the worst case complexity would depend exponentially on the differences between the two designs.

The fact that in our experiments we did not experience an exponential blowup suggests that in synthesized logic it is normally possible to keep the miter size constant and small. This does not imply optimization based on local transformations only; the test generator does take into account global information far away from the miter under consideration.

### 3. Experimental results

*Table 1.* Verification of ISCAS benchmarks.

CIRCUIT	CONNECTS		CPU TIME [sec]	RELATIVE TIME
	BEFORE	AFTER		
C17	31	19	1	20
C432	712	346	4	4
C499	1105	539	38	24
C880	1125	609	5	3
C1355	1937	903	9	3
C1908	2244	647	22	9
C2670	2783	1211	58	15
C3540	3666	1652	39	8
C5315	5346	2880	29	4
C6288	9120	4690	193	13
C7522	8107	3288	136	12
S27	26	25	1	12
S208	187	115	1	3
S298	267	175	1	2
S344	304	211	1	2
S349	308	211	1	2
S382	336	217	1	2
S386	367	193	1	2
S400	350	214	1	2
S420	373	231	5	7
S444	382	217	2	3
S510	456	409	3	3
S526	1475	309	2	2
S526N	475	288	2	2
S641	617	310	3	3
S713	668	309	3	3
S820	799	485	5	4
S832	811	485	11	8
S838	739	419	47	35
S1196	1055	849	8	3
S1238	1087	853	10	4
S1238	1087	853	10	4
S1423	1260	815	8	3
S1488	1420	947	11	5
S1494	1426	939	10	4
S5378	4475	2096	30	4
S9234	8240	2903	77	6
S15850	14343	5616	192	8
S35932	30352	17954	519	8
S38417	33798	15187	614	9
S38584	34498	19669	736	9

Table 1 shows results on the ISCAS benchmarks [3, 4]. Each benchmark was synthesized using the standard script of BooleDozer [13], which includes data flow optimization, redundancy removal, factoring, technology mapping and others. We did not run any timing optimization because it is very dependent on timing assertions and we do not anticipate that would cause problems to our algorithm. On the other hand, to the standard script we added a transformation that collapses as large pieces of logic as possible into a two-level representation, minimizes and factors. We added this transformation because it destroys the original structure of the design and therefore makes it difficult to find an internal net  $g$  in the implementation that can replace a given net  $f$  in the specification.

For each design we give the number of connections of the two designs being compared – before and after synthesis. Then we give the CPU time in seconds on IBM RS/6000 model 550. In the last column we give a ratio between the verification time and the time to read in the two designs, because this is a measure independent of the hardware and other system variables that are not pertinent to our algorithm. (A linear algorithm would have a constant relative time.)

## 4. Conclusions

We have shown how a test generator can be used for verification, and demonstrated its effectiveness by verifying all the ISCAS benchmarks, which to our knowledge has not been accomplished by any other approach yet.

The algorithm proved successful because there are apparently many nets in a specification for which there exists a corresponding net in the implementation. This appears true even after extensive global optimizations. It is not very surprising, given the experience of transduction [15], which relies on related phenomenon. Namely, it is very common that in a logic network there are nets computing related functions possibly for unrelated reasons.

It is possible, however, that our algorithm may fail on some designs in the future. Even then it would not be necessary for a designer to partition his design (which is a common industry practice) because our algorithm is not as sensitive to design size as it is to the differences between the two designs being compared. Therefore any partitioning, if necessary, should be in the synthesis process. That is, we can write several checkpoint files during synthesis and do verification between successive pairs of checkpoint files.

## Acknowledgments

I am grateful to Sandip Kundu and Prakash Narain for being able to use their test generator. Andreas Kuehlmann, Vijay Iyengar, Leon Stok and Louise Treillyan read the manuscript and provided valuable suggestions.

## References

- [1] Abramovici, M., Menon, P. R., and Miller, D. J. (1984). Critical path tracing: An alternative to fault simulation. *IEEE Design and Test*, 1:93–93.
- [2] Ashar, P., Ghosh, A., Devadas, S., and Newton, A. (1991). Combinational and sequential logic verification using general binary decision diagrams. In *Proceedings of the International Workshop on Logic Synthesis*.
- [3] Berglez, F., Pownall, P., and Humm, R. (1985). Accelerated ATPG and fault grading via testability analysis. In *International Symposium on Circuits and Systems*, pages 695–698. IEEE.
- [4] Berglez, F., Bruan, D., and Kozminski, K. (1989). Combinational profiles of sequential benchmark circuits. In *International Symposium on Circuits and Systems*, pages 1929–1934. IEEE.
- [5] Berman, C. and Trevillyan, H. (1989). Functional comparison of logic designs for VLSI chips. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 456–459, Santa Clara, CA. IEEE.
- [6] Bryant, R. E. (1986). Graph based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691.
- [7] Fujita, M., Fujisawa, H., and Kawato, N. (1988). Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 2–5. IEEE.
- [8] Goel, P. (1981). An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, 30.
- [9] Jain, R., Bittner, J., Fussell, D., and Abraham, J. (1991). Probabilistic design verification. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 468–471. IEEE.
- [10] Jeong, S.-W., Plessier, B., Hachtel, G., and Somezi, F. (1991). Extended BDD's: Trading off canonicity for structure in verification algorithms. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 464–467, Santa Clara, CA. IEEE.
- [11] Kunda, R. P., Narain, P., Abraham, J. A., and Rathi, B. D. (1990). Speed up of test generation using high-level primitives. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 594–599, Orlando. ACM/IEEE.
- [12] Kundu, S., Huisman, L. H., Nair, I., Iyengar, V. S., and Reddy, L. N. (1992). A small test generator for large designs. In *Proceedings of the International Test Conference*, pages 30–40. IEEE.
- [13] Kung, D., Damiano, R., Nix, T., and Geiger, D. (1992). BDDMAP: A technology mapper based on a new covering algorithm. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*. ACM/IEEE.
- [14] Malik, S., Wang, A., Brayton, R. K., and Sangiovanni-Vincentelli, A. (1988). Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 6–9. IEEE.
- [15] Muroga, S., Kambayashi, Y., Lai, H. C., and Culliney, J. N. (1989). The transduction method – design of logic networks based on permissible functions. *IEEE Transactions on Computers*, 38(10):1404–1424.

# GRASP—A NEW SEARCH ALGORITHM FOR SATISFIABILITY

João P. Marques Silva

*Cadence European Laboratories,  
IST/INESC-ID,  
1000 Lisboa, Portugal*

Karem A. Sakallah

*Department of EECS,  
University of Michigan,  
Ann Arbor, Michigan 48109-2122*

## Abstract

This paper introduces GRASP (Generic seaRch Algorithm for the Satisfiability Problem), an integrated algorithmic framework for SAT that unifies several previously proposed search-pruning techniques and facilitates identification of additional ones. GRASP is premised on the inevitability of conflicts during search and its most distinguishing feature is the augmentation of basic backtracking search with a powerful conflict analysis procedure. Analyzing conflicts to determine their causes enables GRASP to backtrack non-chronologically to earlier levels in the search tree, potentially pruning large portions of the search space. In addition, by “recording” the causes of conflicts, GRASP can recognize and preempt the occurrence of similar conflicts later on in the search. Finally, straightforward bookkeeping of the causality chains leading up to conflicts allows GRASP to identify assignments that are necessary for a solution to be found. Experimental results obtained from a large number of benchmarks, including many from the field of test pattern generation, indicate that application of the proposed conflict analysis techniques to SAT algorithms can be extremely effective for a large number of representative classes of SAT instances.

## 1. Introduction

The Boolean satisfiability problem (SAT) appears in many contexts in the field of computer-aided design of integrated circuits including automatic test pattern generation (ATPG), timing analysis, delay fault testing, and logic verification, to name just a few. Though well-researched and widely investigated, it remains the focus of continuing interest because efficient techniques for its solution can have great impact. SAT belongs to the class of NP-complete problems whose algorithmic solutions are currently believed to have exponential worst case complexity [6]. Over the years, many algorithmic solutions have been proposed for SAT, the most well known being the different variations of the Davis-Putnam procedure [3]. The best known version of this procedure is based on a backtracking search algorithm that, at each node in the search tree, elects an assignment and prunes subsequent search by iteratively applying the *unit clause*

and the *pure literal* rules [18]. Iterated application of the unit clause rule is commonly referred to as *Boolean Constraint Propagation* (BCP) or as *derivation of implications* in the electronic CAD literature [1].

Most of the recently proposed improvements to the basic Davis-Putnam procedure [5, 10, 17, 18] can be distinguished based on their decision making heuristics or their use of preprocessing or relaxation techniques. Common to all these approaches, however, is the chronological nature of backtracking. Nevertheless, non-chronological backtracking techniques have been extensively studied and applied to different areas of Artificial Intelligence, particularly Truth Maintenance Systems (TMS), Constraint Satisfaction Problems (CSP) and Automated Deduction, in some cases with very promising experimental results. (Bibliographic references to the work in these areas can be found in [15].)

Interest in the direct application of SAT algorithms to electronic design automation (EDA) problems has been on the rise recently [2, 10, 17]. In addition, improvements to the traditional structural (path sensitization) algorithms for some EDA problems, such as ATPG, include search-pruning techniques that are also applicable to SAT algorithms in general [8, 9, 13]. The main purpose of this paper is to introduce a procedure for the analysis of conflicts in search algorithms for SAT. Even though the conflict analysis procedure is described in the context of SAT, it can be naturally extended to EDA-specific algorithms, thus complementing other well-known search-pruning techniques [2, 9].

The proposed conflict analysis procedure has been incorporated in GRASP (*Generic seaRch Algorithm for the Satisfiability Problem*), an integrated algorithmic framework for SAT. Several features distinguish the conflict analysis procedure in GRASP from others used in TMSs and CSPs. First, conflict analysis in GRASP is tightly coupled with BCP and the causes of conflicts need not necessarily correspond to decision assignments. Second, clauses can be added to the original set of clauses, and the number and size of added clauses is user-controlled. This is in explicit contrast with nogood recording techniques developed for TMSs and CSPs. Third, GRASP employs techniques to prune the search by analyzing the implication *structure* generated by BCP. Exploiting the “anatomy” of conflicts in this manner has no equivalent in other areas.

Some of the proposed techniques have also been applied in several structural ATPG algorithms [8, 16], among others. The GRASP framework, however, permits a unified representation of all known search-pruning methods and potentiates the identification of additional ones. The basic SAT algorithm in GRASP is also customizable to take advantage of application-specific characteristics to achieve additional efficiencies [13]. Finally, the framework is organized to allow easy adaptation of other algorithmic techniques, such as those in [2, 9], whose operation is orthogonal to those described here.

The remainder of this paper is organized in four sections. In Section 2, we introduce the basics of backtracking search, particularly our implementation of

BCP, and describe the overall architecture of GRASP. This is followed, in Section 3, by a detailed discussion of the procedures for conflict analysis and how they are implemented. Extensive experimental results on a wide range of benchmarks, including many from the field of ATPG, are presented and analyzed in Section 4. In particular, GRASP is shown to outperform two recent state-of-the-art SAT algorithms [5, 17] on most, but not all, benchmarks. The paper concludes in Section 5 with some suggestions for further research.

## 2. Definitions

### 2.1 Basic Definitions and Notation

A conjunctive normal form (CNF) formula  $\varphi$  on  $n$  binary variables  $x_1, \dots, x_n$  is the conjunction (AND) of  $m$  clauses  $\omega_1, \dots, \omega_m$  each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. A formula  $\varphi$  denotes a unique  $n$ -variable Boolean function  $f(x_1, \dots, x_n)$  and each of its clauses corresponds to an implicate of  $f$ . Clearly, a function  $f$  can be represented by many equivalent CNF formulas. A formula is complete if it consists of the entire set of prime implicants for the corresponding function. In general, a complete formula will have an exponential number of clauses. We will refer to a CNF formula as a *clause database* and use “formula,” “CNF formula,” and “clause database” interchangeably. The satisfiability problem (SAT) is concerned with finding an assignment to the arguments of  $f(x_1, \dots, x_n)$  that makes the function equal to 1 or proving that the function is equal to the constant 0.

A backtracking search algorithm for SAT is implemented by a *search process* that implicitly traverses the space of  $2^n$  possible binary assignments to the problem variables. During the search, a variable whose binary value has already been determined is considered to be *assigned*; otherwise it is *unassigned* with an implicit value of  $X \equiv \{0, 1\}$ . A *truth assignment* for a formula  $\varphi$  is a set of assigned variables and their corresponding binary values. It will be convenient to represent such assignments as sets of variable/value pairs; for example  $A = \{(x_1, 0), (x_7, 1), (x_{13}, 0)\}$ . Alternatively, assignments can be denoted as  $A = \{(x_1 = 0), (x_7 = 1), (x_{13} = 0)\}$ . Sometimes it is convenient to indicate that a variable  $x$  is assigned without specifying its actual value. In such cases, we will use the notation  $v(x)$  to denote the binary value assigned to  $x$ . An assignment  $A$  is complete if  $|A| = n$ ; otherwise it is partial. Evaluating a formula  $\varphi$  for a given truth assignment  $A$  yields three possible outcomes:  $\varphi|_A = 1$  and we say that  $\varphi$  is satisfied and refer to  $A$  as a *satisfying assignment*;  $\varphi|_A = 0$  in which case  $\varphi$  is unsatisfied and  $A$  is referred to as an *unsatisfying assignment*; and  $\varphi|_A = X$  indicating that the value of  $\varphi$  cannot be resolved by the assignment. This last case can only happen when  $A$  is a partial assignment. An assignment partitions the clauses of  $\varphi$  into three sets: satisfied clauses (evaluating to 1); unsatisfied

clauses (evaluating to 0); and unresolved clauses (evaluating to  $X$ ). The unassigned literals of a clause are referred to as its *free literals*. A clause is said to be *unit* if the number of its free literals is one.

## 2.2 Formula Satisfiability

Formula satisfiability is concerned with determining if a given formula  $\varphi$  is satisfiable and with identifying a satisfying assignment for it. Starting from an empty truth assignment, a backtrack search algorithm traverses the space of truth assignments implicitly and organizes the search for a satisfying assignment by maintaining a *decision tree*. Each node in the decision tree specifies an elective assignment to an unassigned variable; such assignments are referred to as *decision assignments*. A *decision level* is associated with each decision assignment to denote its depth in the decision tree; the first decision assignment at the root of the tree is at decision level 1. The search process iterates through the steps of:

- 1 Extending the current assignment by making a decision assignment to an unassigned variable. This *decision process* is the basic mechanism for exploring new regions of the search space. The search terminates successfully if all clauses become satisfied; it terminates unsuccessfully if some clauses remain unsatisfied and all possible assignments have been exhausted.
- 2 Extending the current assignment by following the logical consequences of the assignments made thus far. The additional assignments derived by this *deduction process* are referred to as *implication assignments* or, more simply, *implications*. The deduction process may also lead to the identification of one or more unsatisfied clauses implying that the current assignment is not a satisfying assignment. Such an occurrence is referred to as a *conflict* and the associated unsatisfying assignments, called *conflicting assignments*.
- 3 Undoing the current assignment, if it is conflicting, so that another assignment can be tried. This *backtracking process* is the basic mechanism for retreating from regions of the search space that do not correspond to satisfying assignments.

The decision level at which a given variable  $x$  is either electively assigned or forcibly implied will be denoted by  $\delta(x)$ . When relevant to the context, the assignment notation introduced earlier may be extended to indicate the decision level at which the assignment occurred. Thus,  $x = v@d$  would be read as “ $x$  becomes equal to  $v$  at decision level  $d$ .”

The average complexity of the above search process depends on how decisions, deductions, and backtracking are made. It also depends on the formula

itself. The implications that can be derived from a given partial assignment depend on the set of available clauses. In general, a formula consisting of more clauses will enable more implications to be derived and will reduce the number of backtracks due to conflicts. The limiting case is the complete formula that contains all prime implicants. For such a formula no conflicts can arise since all logical implications for a partial assignment can be derived. This, however, may not lead to shorter execution times since the size of such a formula may be exponential.

## 2.3 Function Satisfiability

Given an initial formula  $\varphi$  many search systems attempt to augment it with additional implicants to increase the deductive power during the search process. This is usually referred to as “learning” [12] and can be performed either as a preprocessing step (static learning) or during the search (dynamic learning). Even though learning as defined in [10, 12] only yields implicants of size 2 (i.e. non-local implications), the concept can be readily extended to implicants of arbitrary size.

Our approach can be classified as a dynamic learning search mechanism based on diagnosing the causes of conflicts. It considers the occurrence of a conflict, which is unavoidable for an unsatisfiable instance unless the formula is complete, as an opportunity to “learn from the mistake that led to the conflict” and introduces additional implicants to the clause database only when it stumbles. ***Conflict diagnosis*** produces three distinct pieces of information that can help speed up the search:

- 1 New implicants that did not exist in the clause database and that can be identified with the occurrence of the conflict. These clauses may be added to the clause database to avert future occurrence of the same conflict and represent a form of ***conflict-based equivalence*** (CBE).
- 2 An indication of whether the conflict was ultimately due to the most recent decision assignment or to an earlier decision assignment.
  - If that assignment was the most recent (i.e. at the current decision level), the opposite assignment (if it has not been tried) is immediately implied as a necessary consequence of the conflict; we refer to this as a ***failure-driven assertion*** (FDA).
  - If the conflict resulted from an earlier decision assignment (at a lower decision level), the search can backtrack to the corresponding level in the decision tree since the subtree rooted at that level corresponds to assignments that will yield the same conflict. The ability to identify a backtracking level that is much earlier than the

Current Truth Assignment:  $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \dots\}$   
 Current Decision Assignment:  $\{x_1 = 1@6\}$

$$\begin{aligned}\omega_1 &= (\neg x_1 + x_2) \\ \omega_2 &= (\neg x_1 + x_3 + x_9) \\ \omega_3 &= (\neg x_2 + \neg x_3 + x_4) \\ \omega_4 &= (\neg x_4 + x_5 + x_{10}) \\ \omega_5 &= (\neg x_4 + x_6 + x_{11}) \\ \omega_6 &= (\neg x_5 + \neg x_6) \\ \omega_7 &= (x_1 + x_7 + \neg x_{12}) \\ \omega_8 &= (x_1 + x_8) \\ \omega_9 &= (\neg x_7 + \neg x_8 + \neg x_{13}) \\ &\dots\end{aligned}$$

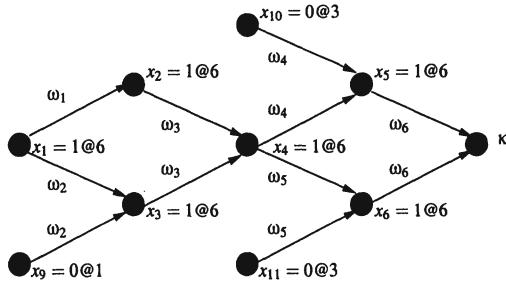


Figure 1. Clause database and partial implication graph.

current decision level is a form of non-chronological backtracking that we refer to as *conflict-directed backtracking* (CDB), and has the potential of significantly reducing the amount of search.

These conflict diagnosis techniques are discussed further in Section 3.

## 2.4 Structure of the Search Process

The basic mechanism for deriving implications from a given clause database is Boolean constraint propagation (BCP) [5, 18]. Consider a formula  $\varphi$  containing the clause  $\omega = (x + \neg y)$  and assume  $y = 1$ . For any satisfying assignment to  $\varphi$ ,  $\omega$  requires that  $x$  be equal to 1, and we say that  $y = 1$  implies  $x = 1$  due to  $\omega$ . In general, given a unit clause  $(l_1 + \dots + l_k)$  of  $\varphi$  with free literal  $l_j$ , consistency requires  $l_j = 1$  since this represents the only possibility for the clause to be satisfied. If  $l_j = x$ , then the assignment  $x = 1$  is required; if  $l_j = \neg x$  then  $x = 0$  is required. Such assignments are referred to as *logical implications* (implications, for short) and correspond to the application of the unit clause rule proposed by M. Davis and H. Putnam [3]. BCP refers to the iterated application of this rule to a clause database until the set of unit clauses becomes empty or one or more clauses become unsatisfied.

Let the assignment of a variable  $x$  be implied due to a clause  $\omega = (l_1 + \dots + l_k)$ . The *antecedent assignment* of  $x$ , denoted as  $A(x)$ , is defined as the set of assignments to variables other than  $x$  with literals in  $\omega$ . Intuitively,  $A(x)$  designates those variable assignments that are directly responsible for implying the assignment of  $x$  due to  $\omega$ . For example, the antecedent assignments of  $x$ ,  $y$  and  $z$  due to the clause  $\omega = (x + y + \neg z)$  are, respectively,  $A(x) = \{y = 0, z = 1\}$ ,  $A(y) = \{x = 0, z = 1\}$ , and  $A(z) = \{x = 0, y = 0\}$ . Note that the antecedent assignment of a decision variable is empty.

The sequence of implications generated by BCP is captured by a directed *implication graph*  $I$  defined as follows (see Figure 1):

- 1 Each vertex in  $I$  corresponds to a variable assignment  $x = v(x)$ .
- 2 The predecessors of vertex  $x = v(x)$  in  $I$  are the antecedent assignments  $A(x)$  corresponding to the unit clause  $\omega$  that led to the implication of  $x$ . The directed edges from the vertices in  $A(x)$  to vertex  $x = v(x)$  are all labeled with  $\omega$ . Vertices that have no predecessors correspond to decision assignments.
- 3 Special conflict vertices are added to  $I$  to indicate the occurrence of conflicts. The predecessors of a conflict vertex  $\kappa$  correspond to variable assignments that force a clause  $\omega$  to become unsatisfied and are viewed as the antecedent assignment  $A(\kappa)$ . The directed edges from the vertices in  $A(\kappa)$  to  $\kappa$  are all labeled with  $\omega$ .

The decision level of an implied variable  $x$  is related to those of its antecedent variables according to:

$$\delta(x) = \max \{ \delta(y) | (y, v(y)) \in A(x) \} \quad (1)$$

## 2.5 Search Algorithm Template

The general structure of the GRASP search algorithm is shown in Figure 2. We assume that an initial clause database  $\varphi$  and an initial assignment  $A$ , at decision level 0, are given. This initial assignment, which may be empty, may be viewed as an additional problem constraint and causes the search to be restricted to a subcube of the  $n$ -dimensional Boolean space. As the search proceeds, both  $\varphi$  and  $A$  are modified. The recursive search procedure consists of four major operations:

- 1 **Decide()**, which chooses a decision assignment at each stage of the search process. Decision procedures are commonly based on heuristic knowledge. For the results given in Section 4, the following greedy heuristic is used:

*At each node in the decision tree evaluate the number of clauses directly satisfied by each assignment to each variable. Choose the variable and the assignment that directly satisfies the largest number of clauses.*

Other decision making procedures have been incorporated in GRASP, as described in [15].

- 2 **Deduce()**, which implements BCP and (implicitly) maintains the resulting implication graph. (See [15] for the details of **Deduce()**.)
- 3 **Diagnose()**, which identifies the causes of conflicts and can augment the clause database with additional implicants. Realization of different conflict diagnosis procedures is the subject of Section 3.

```

// Global variables: Clause database  $\varphi$ 
// Variable assignment  $A$ 
// Return value: FAILURE or SUCCESS
// Auxiliary variables: Backtracking level  $\beta$ 
//
GRASP()
{
    return (Search (0,  $\beta$ ) != SUCCESS) ? FAILURE : SUCCESS;
}
//
// Input argument: Current decision level  $d$ 
// Output argument: Backtracking level  $\beta$ 
// Return value: CONFLICT or SUCCESS
//
Search( $d$ , & $\beta$ )
{
    if (Decide( $d$ ) == SUCCESS)
        return SUCCESS;
    while (TRUE) {
        if (Deduce( $d$ ) != CONFLICT) {
            if (Search( $d$  + 1,  $\beta$ ) == SUCCESS) { return SUCCESS; }
            else if ( $\beta \neq d$ ) { Erase(); return CONFLICT; }
        }
        if (Diagnose( $d$ ,  $\beta$ ) == CONFLICT) { Erase(); return CONFLICT; }
        Erase();
    }
}
//
Diagnose( $d$ , & $\beta$ )
{
     $\omega_C(\kappa)$  = Conflict_Induced_Clause(); // From (4)
    Update_Clause_Database( $\omega_C(\kappa)$ );
     $\beta$  = Compute_Max_Level(); // From (7)
    if ( $\beta \neq d$ ) {
        add new conflict vertex  $\kappa$  to  $I$ ;
        record  $A(\kappa)$ ;
        return CONFLICT;
    }
    return SUCCESS;
}

```

Figure 2. Description of GRASP.

4 Erase(), which deletes the assignments at the current decision level.

We refer to Decide(), Deduce() and Diagnose() as the *Decision*, *Deduction* and *Diagnosis engines*, respectively. Different realizations of these engines lead to different SAT algorithms. For example, the Davis-Putnam procedure can be

emulated with the above algorithm by defining a decision engine, requiring the deduction engine to implement BCP and the pure literal rule, and organizing the diagnosis engine to implement chronological backtracking.

### 3. Conflict Analysis Procedures

When a conflict arises during BCP, the *structure* of the implication sequence converging on a conflict vertex  $\kappa$  is analyzed to determine those (unsatisfying) variable assignments that are directly responsible for the conflict. The conjunction of these conflicting assignments is an implicant that represents a sufficient condition for the conflict to arise. Negation of this implicant, therefore, yields an implicate of the Boolean function  $f$  (whose satisfiability we seek) that does not exist in the clause database  $\varphi$ . This new implicate, referred to as a *conflict-induced clause*, provides the primary mechanism for implementing failure-driven assertions, non-chronological conflict-directed backtracking, and conflict-based equivalence (see Section 2.3). In TMS [16] and in some algorithms for CSP [11], “nogoods” provide conditions similar to conflict-induced clauses. Nevertheless, the basic mechanism for creating conflict-induced clauses differs.

We denote the conflicting assignment associated with a conflict vertex  $\kappa$  by  $A_C(\kappa)$  and the associated conflict-induced clause by  $\omega_C(\kappa)$ . The conflicting assignment is determined by a backward traversal of the implication graph starting at  $\kappa$ . Besides the decision assignment at the current decision level, only those assignments that occurred at previous decision levels are included in  $A_C(\kappa)$ . This is justified by the fact that the decision assignment at the current decision level is directly responsible for all implied assignments at that level. Thus, along with assignments from previous levels, the decision assignment at the current decision level is a sufficient condition for the conflict. To facilitate the computation of  $A_C(\kappa)$  we partition the antecedent assignments of  $\kappa$  as well as those for variables assigned at the current decision level into two sets. Let  $x$  denote either  $\kappa$  or a variable that is assigned at the current decision level. The partition of  $A(x)$  is then given by:

$$\begin{aligned}\Lambda(x) &= \{(y, v(y)) \in A(x) \mid \delta(y) < \delta(x)\} \\ \Sigma(x) &= \{(y, v(y)) \in A(x) \mid \delta(y) = \delta(x)\}\end{aligned}\tag{2}$$

For example, referring to the implication graph of Figure 1,  $\Lambda(x_6) = \{x_{11} = 0 @ 3\}$  and  $\Sigma(x_6) = \{x_4 = 1 @ 6\}$ . Determination of the conflicting assignment  $A_C(\kappa)$  can now be computed using the following recursive definition:

$$A_C(x) = \begin{cases} (x, v(x)) & \text{if } A(x) = \emptyset \\ \Lambda(x) \cup \left[ \bigcup_{(y, v(y)) \in \Sigma(x)} A_C(y) \right] & \text{otherwise} \end{cases}\tag{3}$$

and starting with  $x = \kappa$ . The conflict-induced clause corresponding to  $A_C(\kappa)$  is now determined according to:

$$\omega_C(\kappa) = \sum_{(x, v(x)) \in A_C(\kappa)} x^{v(x)} \quad (4)$$

where, for a binary variable  $x$ ,  $x^0 \equiv x$  and  $x^1 \equiv \neg x$ . Application of (2)-(4) to the conflict depicted in Figure 1 yields the following conflicting assignment and conflict-induced clause at decision level 6:

$$\begin{aligned} A_C(\kappa) &= \{x_1 = 1, x_9 = 0, x_{10} = 0, x_{11} = 0\} \\ \omega_C(\kappa) &= (\neg x_1 + x_9 + x_{10} + x_{11}) \end{aligned} \quad (5)$$

### 3.1 Standard Conflict Analysis Engine

The identification of a conflict-induced clause  $\omega_C(\kappa)$  enables the derivation of further implications that help prune the search. Immediate implications of  $\omega_C(\kappa)$  include asserting the current decision variable to its opposite value and determining a backtracking level for the search process. Such immediate implications do not require that  $\omega_C(\kappa)$  be added to the clause database. Augmenting the clause database with  $\omega_C(\kappa)$ , however, has the potential of identifying future implications that are not derivable without  $\omega_C(\kappa)$ . In particular, adding  $\omega_C(\kappa)$  to the clause database ensures that the search engine will not regenerate the conflicting assignment that led to the current conflict.

**3.1.1 Failure-Driven Assertions.** If  $\omega_C(\kappa)$  involves the current decision variable, erasing the implication sequence at the current decision level makes  $\omega_C(\kappa)$  a unit clause and causes the immediate implication of the decision variable to its opposite value. We refer to such assignments as failure-driven assertions (FDAs) to emphasize that they are implications of conflicts and not decision assignments. We note further that their derivation is automatically handled by our BCP-based deduction engine and does not require special processing. This is in contrast with most search-based SAT algorithms that treat a second branch at the current decision level as another decision assignment. Using our running example (see Figure 1) as an illustration, we note that after erasing the conflicting implication sequence at level 6, the conflict-induced clause  $\omega_C(\kappa)$  in (5) becomes a unit clause with  $\neg x_1$  as its free literal. This immediately implies the assignment  $x_1 = 0$  and  $x_1$  is said to be asserted.

**3.1.2 Conflict-Directed Backtracking.** If all the literals in  $\omega_C(\kappa)$  correspond to variables that were assigned at decision levels that are *lower* than the current decision level, we can immediately conclude that the search process needs to backtrack. This situation can only take place when the conflict in question is produced as a direct consequence of diagnosing a previous conflict and

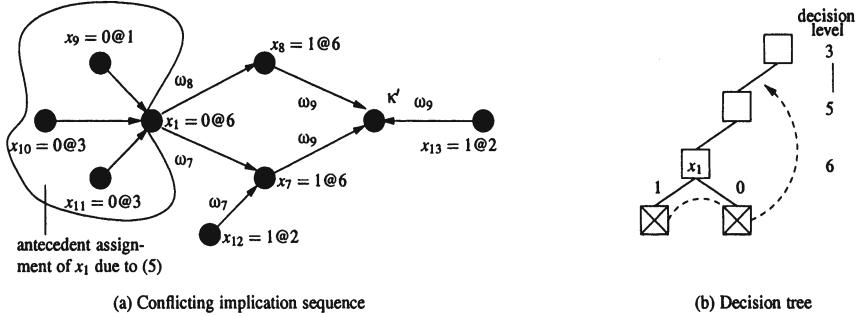


Figure 3. Non-chronological backtracking.

is illustrated in Figure 3 (a) for our working example. The implication sequence generated after asserting  $x_1 = 0$  due to conflict  $\kappa$  leads to another conflict  $\kappa'$ . The conflicting assignment and conflict-induced clause associated with this new conflict are easily determined to be

$$\begin{aligned} A_C(\kappa') &= \{x_9 = 0, x_{10} = 0, x_{11} = 0, x_{12} = 1, x_{13} = 1\} \\ \omega_C(\kappa') &= (x_9 + x_{10} + x_{11} + \neg x_{12} + \neg x_{13}) \end{aligned} \quad (6)$$

and clearly show that the assignments that led to this second conflict were all made prior to the current decision level.

In such cases, it is easy to show that no satisfying assignments can be found until the search process backtracks to the highest decision level at which assignments in  $A_C(\kappa')$  were made. Denoting this **backtrack level** by  $\beta$ , it is simply calculated according to:

$$\beta = \max \{\delta(x) | (x, v(x)) \in A_C(\kappa')\} \quad (7)$$

When  $\beta = d - 1$ , where  $d$  is the current decision level, the search process backtracks **chronologically** to the immediately preceding decision level. When  $\beta < d - 1$ , however, the search process may backtrack **non-chronologically** by jumping back over several levels in the decision tree. It is worth noting that all truth assignments that are made after decision level  $\beta$  will force the just-identified conflict-induced clause  $\omega_C(\kappa')$  to be unsatisfied. A search engine that backtracks chronologically may, thus, waste a significant amount of time exploring a useless region of the search space only to discover after much effort that the region does not contain any satisfying assignments. In contrast, the GRASP search engine jumps *directly* from the current decision level back to decision level  $\beta$ . At that point,  $\omega_C(\kappa')$  is used to either derive a FDA at decision level  $\beta$  or to calculate a new backtracking decision level.

For our example, after occurrence of the second conflict the backtrack decision level is calculated, from (7), to be 3. Backtracking to decision level 3, the deduction engine creates a conflict vertex corresponding to  $\omega_C(\kappa')$ . Diagnosis

of this conflict leads to a FDA of the decision variable at level 3 (see Figure 3 (b)).

The pseudo-code illustrating the main features of the diagnosis engine in GRASP is shown in Figure 2. General proofs of the soundness and completeness of GRASP can be found in [7, 14].

## 3.2 Variations on the Standard Diagnosis Engine

The standard conflict diagnosis, described in the previous section, suffers from two drawbacks. First, conflict analysis introduces significant overhead which, for some instances of SAT, can lead to large run times. Second, the size of the clause database grows with the number of backtracks; in the worst case such growth can be exponential in the number of variables.

The first drawback is inherent to the algorithmic framework we propose. Fortunately, the experimental results presented in Section 4 clearly suggest that, for specific instances of SAT, the performance gains far outweigh the procedure's additional overhead.

One solution to the second drawback is a simple modification to the conflict diagnosis engine that guarantees the worst case growth of the clause database to be polynomial in the number of variables. The main idea is to be selective in the choice of clauses to add to the clause database. Assume that we are given an integer parameter  $k$ . Conflict-induced clauses whose size (number of literals) is no greater than  $k$  are marked *green* and handled as described earlier by the standard diagnosis engine. Conflict-induced clauses of size greater than  $k$  are marked *red* and kept around only while they are unit clauses. Implementation of this scheme requires a simple modification to procedure `Erase()`, which must now delete red clauses with more than one free literal, and to the diagnosis engine, which must attach a color tag to each conflict-induced clause. With this modification the worst case growth becomes polynomial in the number of variables as a function of the fixed integer  $k$ .

Further enhancements to the conflict diagnosis engine involve generating stronger implicants (containing fewer literals) by more careful analysis of the structure of the implication graph. Such implicants are associated with the dominators [15] of the conflict vertex  $\kappa$ . These dominators, referred to as *unique implication points* (UIPs), can be identified in linear time with a single traversal of the implication graph. Additional details of the above improvements to the standard diagnosis engine can be found in [15].

## 4. Experimental Results

In this section we present an experimental comparison of GRASP with two state-of-the-art and publicly available SAT programs, TEGUS [17] and POSIT [5]. TEGUS was adapted to read CNF formulas and augmented to con-

tinue searching when all its default options were exhausted in order to abort fewer faults. No changes were made to POSIT.

GRASP and POSIT have been implemented in C++, whereas TEGUS has been implemented in C. The programs were compiled with GCC 2.7.2 and run on a SUN SPARC 5/85 machine with 64 MByte of RAM. The experimental evaluation of the three programs is based on two different sets of benchmarks:

- The UCSC benchmarks [4], developed at the University of California, Santa Cruz, that include instances of SAT commonly encountered in test pattern generation of combinational circuits for bridging and stuck-at faults.
- The DIMACS challenge benchmarks [4], that include instances of SAT from several authors and from different application areas.

For the experimental results given below, GRASP was configured to use the decision engine described in Section 2.5, to allow the generation of clauses based on UIPs, and to limit the size of clauses added to the clause database to 20 or fewer literals. All SAT programs were run with a CPU time limit of 10,000 seconds (about three hours).

For the tables of results the following definitions apply. A benchmark suite is partitioned into classes of related benchmarks. In each class, #M denotes the total number of class members; #S denotes the number of class members for which the program terminated in less than the allowed 10,000 CPU seconds; and Time denotes the total CPU time, in seconds, taken to process all members of the class.

The results obtained for the UCSC benchmarks are shown in Table 1. The BF and SSA benchmark classes denote, respectively, CNF formulas for bridging and stuck-at faults. For these benchmarks GRASP performs significantly better than the other programs. Both POSIT and TEGUS abort a large number of problem instances and require much larger CPU times. These benchmarks are characterized by extremely sparse CNF formulas for which BCP-based conflict analysis works particularly well. The performance difference between GRASP and TEGUS, a very efficient ATPG tool, clearly illustrates the power of the search-pruning techniques included in GRASP.

An experimental study of the effect of the growth of the clause database on the amount of search and the CPU time can be found in [15]. In general, adding larger clauses helps reducing the number of backtracks and the CPU time. This holds true until the overhead introduced by the additional clauses offsets the gains of reducing the amount of search.

GRASP was also compared with the other algorithms on the DIMACS benchmarks [4]], and the results are included in Table 1. We can conclude that for classes of benchmarks where GRASP performs better the other programs either take a very long time to find a solution or are unable to find a solution in less than 10,000 seconds. We can also observe that benchmarks on which POSIT

Benchmark Class	#M	GRASP		TEGUS		POSIT	
		#S	Time	#S	Time	#S	Time
BF-0432	21	21	47.6	19	53,852	21	55.8
BF-1355	149	149	125.7	53	993,915	64	946,127
BF-2670	53	53	68.3	25	295,410	53	2,971
SSA-0432	7	7	1.1	7	1,593	7	0.2
SSA-2670	12	12	51.5	0	120,000	12	2,826
SSA-6288	3	3	0.2	3	17.5	3	0.0
SSA-7552	80	80	19.8	80	3,406	80	60.0
AIM-100	24	24	1.8	24	107.9	24	1,290
AIM-200	24	24	10.8	23	14,059	13	117,991
BF	4	4	7.2	2	26,654	2	20,037
DUBOIS	13	13	34.4	5	90,333	7	77,189
II-32	17	17	7.0	17	1,231	17	650.1
PRET	8	8	18.2	4	42,579	4	40,691
SSA	8	8	6.5	6	20,230	8	85.3
AIM-50	24	24	0.4	24	2.2	24	0.4
II-8	14	14	23.4	14	11.8	14	2.3
JNH	50	50	21.3	50	6,055	50	0.8
PAR-8	10	10	0.4	10	1.5	10	0.1
PAR-16	10	10	9,844	10	9,983	10	72.1
II-16	10	9	10,311	10	269.6	9	10,120
H	7	5	27,184	4	32,942	6	11,540
F	3	0	30,000	0	30,000	0	30,000
G	4	0	40,000	0	40,000	0	40,000
PAR-32	10	0	100,000	0	100,000	0	100,000

Table 1. Results on the UCSC and DIMACS benchmarks.

performs better than GRASP can also be handled by GRASP; only the overhead inherent to GRASP becomes apparent.

Another useful experiment is to measure how well conflict analysis works in practice. For this purpose statistics regarding some DIMACS benchmarks are shown in Table 2, where #B denotes the number of backtracks, #NCB denotes the number of non-chronological backtracks, #LJ is the size of the largest non-chronological backtrack, #UIP indicates the number of unique implication points found, %G denotes the variation in size of the clause database, and Time is the CPU time in seconds. From these examples several conclusions can be drawn. First, the number of non-chronological backtracks can be a significant percentage of the total number of backtracks. Second, the jumps in the decision tree can save a large amount of search work. As can be observed, in some cases the jumps taken potentially save searching millions of nodes in the decision tree. Third, the growth of the clause database is not necessarily large. Fourth, UIPs do occur in practice and for some benchmarks a reasonable number is found given

Benchmark	#B	#NCB	#LJ	#UIP	%G	GRASP Time	TEGUS Time	POSIT Time
aim.200.2.y2	109	50	13	25	153	0.38	2.80	7,991
aim.200.2.y3	74	35	16	15	100	0.31	0.64	>10,000
aim.200.2.n1	29	20	12	5	23	0.13	69.93	>10,000
aim.200.2.n2	39	20	37	4	44	0.19	87.53	>10,000
bf0432-007	335	124	17	32	48	5.18	6,649	11.79
bf1355-075	40	20	24	2	7	1.25	4.83	>10,000
bf1355-638	11	7	8	4	1	0.32	>10,000	>10,000
bf2670-001	16	8	22	2	3	0.40	>10,000	25.64
dubois30	233	72	16	21	466	0.68	>10,000	>10,000
dubois50	485	175	26	51	632	2.80	>10,000	>10,000
dubois100	1438	639	67	150	1034	26.22	>10,000	>10,000
pret60_40	147	98	17	8	407	0.41	652.30	175.49
pret60_60	131	83	16	10	354	0.35	639.27	173.12
pret150_25	428	313	38	35	588	4.84	>10,000	>10,000
pret150_75	388	257	49	20	447	3.85	>10,000	>10,000
ssa0432-003	37	6	5	1	31	0.15	221.71	0.01
ssa2670-130	130	45	34	10	17	2.07	>10,000	14.23
ssa2670-141	377	97	16	28	66	3.42	>10,000	70.82
ii16a1	110	19	13	0	0	13.61	5.99	>10,000
ii16b2	2664	120	9	39	64	175.85	6.94	16.38
ii16b1	88325	2588	41	624	132	>10,000	21.65	16.73

Table 2. Statistics of running GRASP on selected benchmarks.

the number of backtracks. Finally, for most of these examples conflict analysis causes GRASP to be much more efficient than POSIT and TEGUS. Nevertheless, either POSIT or TEGUS can be more efficient in specific benchmarks, as the examples of the last three rows of Table 2 indicate. TEGUS performs particularly well on these instances because they are satisfiable and because TEGUS iterates several decision making procedures.

## 5. Conclusions and Research Directions

This paper introduces a procedure for conflict analysis in satisfiability algorithms and describes a configurable algorithmic framework for solving SAT. Experimental results indicate that conflict analysis and its by-products, non-chronological backtracking and identification of equivalent conflicting conditions, can contribute decisively for efficiently solving a large number of classes of instances of SAT. For this purpose, the proposed SAT algorithm is compared with other state-of-the-art algorithms.

The natural evolution of this research work is to apply GRASP to different EDA applications, in particular test pattern generation, timing analysis, delay

fault testing and equivalence checking, among others. Despite being a fast SAT algorithm, GRASP introduces noticeable overhead that can become a liability for some of these applications. Consequently, besides the algorithmic organization of GRASP, special attention must be paid to the implementation details. One envisioned compromise is to use GRASP as the second choice SAT algorithm for the hard instances of SAT whenever other simpler, but with less overhead, algorithms fail to find a solution in a small amount of CPU time.

Future research work will emphasize heuristic control of the rate of growth of the clause database. Another area for improving GRASP is related with the deduction engine. Improvements to the BCP-based deduction engine are described in [14] and consist of different forms of probing the CNF formula for creating new clauses. This approach naturally adapts and extends other deduction procedures, e.g. recursive learning [9] and transitive closure [2].

## Acknowledgments

This work was supported in part by NSF under grant MIP-9404632.

## References

- [1] Abramovici, M., Breuer, M. A., and Friedman, A. D. (1990). *Digital Systems Testing and Testable Design*. Computer Science Press.
- [2] Chakradhar, S. T., Agrawal, V. D., and Rothweiler, S. G. (1993). A Transitive Closure Algorithm for Test Generation. *IEEE Transactions on Computer-Aided Design*, 12(7):1015–1028.
- [3] Davis, M. and Putnam, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery*, 7:201–215.
- [4] DIMACS Challenge (1993). Dimacs benchmarks in <ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf>. UCSC benchmarks in <ftp://Dimacs.Rutgers.EDU/pub/challenge/sat/benchmarks/cnf>.
- [5] Freeman, J. W. (1995). *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, Philadelphia, PA.
- [6] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.
- [7] Ginsberg, M. (1993). Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46.
- [8] Giraldi, J. and Bushnell, M. L. (1991). Search State Equivalence for Redundancy Identification and Test Generation. In *Proceedings of the International Test Conference*, pages 184–193.
- [9] Kunz, W. and Pradhan, D. (1992). Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits. In *Proceedings of the International Test Conference*, pages 816–825.
- [10] Larrabee, T. (1990). *Efficient Generation of Test Patterns Using Boolean Satisfiability*. PhD thesis, Department of Computer Science, Stanford University. Technical Report STAN-CS-90-1302.

- [11] Schiex, T. and Verfaillie, G. (1993). Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 48–55.
- [12] Schulz, M. H. and Auth, E. (1989). Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification. *IEEE Transactions on Computer-Aided Design*, 8(7):811–816.
- [13] Silva, J. P. M. and Sakallah, K. A. (1994). Dynamic Search-Space Pruning Techniques in Path Sensitization. In *Proceedings of the Design Automation Conference*, pages 705–711.
- [14] Silva, J. P. M. (1995). *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits*. PhD thesis, University of Michigan.
- [15] Silva, J. P. M. and Sakallah, K. A. (1996). GRASP-A New Search Algorithm for Satisfiability. Technical Report CSE-TR-292-96, University of Michigan.
- [16] Stallman, R. M. and Sussman, G. J. (1977). Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9:135–196.
- [17] Stephan, P. R., Brayton, R. K., and Sangiovanni-Vincen-telli, A. L. (1992). Combinational Test Generation Using Satisfiability. Technical Report UCB/ERL M92/112, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley.
- [18] Zabih, R. and McAllester, D. A. (1988). A Rearrangement Search Strategy for Determining Propositional Satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–160.

## Part II

# SYSTEM DESIGN AND ANALYSIS

System Design and Analysis Overview <i>Hugo De Man and Jan Rabaey</i>	93
An Efficient Microcode-Compiler for Custom DSP-Processors ( <i>ICCAD 1987</i> ) <i>Gert Goossens, Jan Rabaey, Joos Vandewalle and Hugo De Man</i>	107
HYPER-LP: A System for Power Minimization Using Architectural Transformations ( <i>ICCAD 1992</i> ) <i>Anantha P. Chandrakasan, Miodrag Potkonjak, Jan Rabaey and Robert W. Brodersen</i>	117
Power Analysis of Embedded Software: First Step Towards Software Power Minimization ( <i>ICCAD 1994</i> ) <i>Vivek Tiwari, Sharad Malik and Andrew Wolfe</i>	129
A Methodology for Correct-by-Construction Latency Insensitive Design ( <i>ICCAD 1999</i> ) <i>Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha and Alberto L. Sangiovanni-Vincentelli</i>	143
Exploring Performance Tradeoffs for Clustered VLIW ASIPs ( <i>ICCAD 2000</i> ) <i>Margarida F. Jacome, Gustavo de Veciana and Viktor Lapinskii</i>	159

# SYSTEM DESIGN AND ANALYSIS OVERVIEW

Hugo De Man

*Katholieke Universiteit Leuven, Belgium*

*Interuniversity Microelectronics Center, Leuven, Belgium*

Jan Rabaey

*University of California at Berkeley*

*Berkeley, CA, USA*

## Abstract

At the inception of ICCAD in 1983, system-level design was only a small fish in the EDA pond. In the earlier conferences, only one or at most 2 sessions were dedicated to the topic. This has changed dramatically over the years, and today system design is one of the pillars of the conference. In this paper, we describe the major trends in the field as can be traced from the papers published in the conference as well as other seminal publications. While doing so, we put the papers selected for this volume in the context of the ongoing trends at the time of publication, and their impact on the field. In addition, we provide some background that may help to determine why some proposed approaches did or did not succeed in the long term. We conclude the paper with some reflections on the past and the future.

## 1. Some Reflections

When browsing through the papers, one cannot escape the realization that the concept of “system” has evolved substantially over the twenty year history of the conference. One can conclude upfront that there are perhaps as many definitions of an electronic system as there are engineers designing them. A famous (or infamous) quote states that the best definition of a system is “the design abstraction just above the level a designer is currently working on.” Although ICCAD does not stand for CAD for IC’s, it is clear that its concept of what a system is coincides roughly with what can be integrated on a single digital chip over time. Moore’s law has allowed us to integrate ever more parts of complete systems on a single piece of silicon. Hence many people today use the term “System-On-a-Chip”, although in reality no chip contains a complete system in itself. We must keep in mind that complete systems such as telecom, automotive, broadcasting services have a much broader scope of which the digital SoC is only a subsystem embedded in transducer networks, complex analog and RF subsystems with which it interacts very closely. Last but not least, a complete system view refers to the interaction of this heterogeneous electronic system with the non-electronic environment.

Hence it is true that SoC design is rapidly evolving from simple component design to design of complete complex subsystems on a chip and that the traditional chip-architects must become more and more familiar with the challenging application domains of the future. How ICCAD will cope with that in the future is an interesting question but a first step in that direction can be found in the embedded tutorials on application domains such as mobile and broadband communication, multimedia, microsystems and the like ([1-5]).

With this constraint in mind one has to recognize that “system design” in the ICCAD sense so far has been restricted to the digital domain and mainly refers to design technology above the logic and RT-levels, whereby the goal is to map or refine a behavioural description directly into micro-architectures that, in turn, can be transformed into the layout or memory content of digital processors on a chip.

In this paper, we sketch the evolution of system-level design domain as observed from twenty years of ICCAD. We conclude the paper with some reflections.

## 2. The Evolution of System-Level Design in 20 Years of ICCAD

### 2.1 The Pre-ICCAD Period

Mapping behaviour into silicon is a two-step process. On the one hand, it requires a behavioural or high-level synthesis step translating a specification into a micro-architecture. In a second step, the micro-architectural structure must be mapped into a layout. A specification consists of a behavioural description in a high-level language together with a set of area, time and power constraints. Basically this requires three ingredients: a language, a target architectural style and a method to synthesize layout from the structural description of the architecture. In all three areas some work took place before the first ICCAD in 1983. Work on micro-architectural synthesis of dedicated data-paths executing instructions generated from a finite state machine (later called FSMD) started in the late seventies. Pioneering work was done by G. Zimmerman at Kiel University [6] resulting in the MIMOLA system and by D. Thomas' group in CMU's leading to the System Architect Workbench [7]. These efforts, whose goal was to generate processor architectures from the instruction set, laid the foundations for high-level synthesis. The internal representation by a control- and data-flowgraph (CDFG), source level optimization, operation scheduling, resource allocation, operation-to-operator assignment, and controller synthesis were clearly identified and the first tools were demonstrated.

Perhaps one of the biggest bottlenecks for the success of high-level synthesis then and still today was the lack of a proper description language. The first attempts to create behavioural languages allowing for bit-true data-types date

back to the pioneering work of Piloty on CONLAN [8] and Barbacci on ISPS [9]. Up to 1987 there was no standardised HDL language like VHDL or VERILOG available and, as a result, high level synthesis tools and systems have been developed around many different custom languages that allowed easy compiling into the internal Control-Data Flow Graph (CDFG) representation but were foreign to the designer community.

As far as implementation into silicon is concerned, it is interesting to see that high-level mapping into register transfer architectures historically preceded the explosion of logic synthesis that took place from the late seventies till the mid-eighties. In the late seventies excellent heuristic algorithms for two-level logic optimisation were developed that could be mapped rather straightforwardly into PLA layout structures for the control part of the architectures [12]. Synthesis of much more efficient multi-level logic and its automated implementation in standard cell layout became practical only after the invention of algebraic methods for logic synthesis and their implementation in a comprehensive CAD system such as MIS, presented in 1986 by Brayton et al [10, 11]. It took even longer before logic synthesis was capable of handling the regular bit-vector structure of data-path operators.

This explains the early success of the so-called procedural layout generators or module generators that mapped the logic repetitive structure of data-paths into an abutment of the data-path leaf cells. This technique dates back to the landmark paper in 1979 by Johansen of Caltech who launched the term Silicon Compilation [13].

## 2.2 The Silicon Compilation Era

Walking through the early ICCAD proceedings, one can see that papers in the category “Systems” in the 1983-1986 era were predominantly addressing Silicon Compilers in the Johansen sense. Procedural module compilers became popular for the layout generation of data-paths in microprocessors and DSP processors. For example, in one of the first papers presented at ICCAD, Buric et al. [14] describe how to impose timing and electrical constraints in data path compilers. It took until 1987 before this approach became substituted by logic synthesis tools and technology mapping into standard cells. Today, module generation remains only in vogue for highly-regular parameterised structures such as embedded memory. This might change in the future when optical proximity effects in lithography, process variations and signal-integrity considerations, may ask again for high degree of regularity, even for the implementation of logic functions.

## 2.3 Architectural or Behavioural Synthesis

Although activities in high-level synthesis (HLS) were reported extensively at the Design Automation Conference from 1979 on, it would take until 1985 for the first session on high-level synthesis to appear at ICCAD. From then on, the synthesis of hardware architectures remained a topic of interest for many years, with many excellent contributions on the classical aspects of scheduling, allocation and assignment, the three basic steps of all high-level synthesis systems. In retrospect, it is important to see that, over time, many researchers came to the conclusion that, although the basic steps are always there, their sequence and nature depends very much on the so-called target architecture to be synthesised, and that in turn this target architecture is a strong function of the application domain. We believe that this is one of the basic reasons (besides the specification language) that high-level synthesis, in spite of its great potential, has not yet reached (or will maybe never reach) the commercial success of RTL based register-transfer level synthesis, now in general use in the industry.

The original target domains for HLS were processor data-paths (synthesized from an instruction set), and real-time digital signal processors. A pioneering example of the former application-type, which is control dominated, was the Yorktown Heights Silicon Compiler of IBM [15]. Yet, it was in the digital signal processing (DSP) arena that HLS made its first real inroads and industrial impact. DSP is data-dominated and transformational in nature whereby repetitive processing on streams of multi-dimensional arrays is dominant. A hallmark example of a synthesis environment in this class was the CATHEDRAL-II compiler from IMEC [16].

Notice that both compilers need very different scheduling techniques. Camposano introduced path-based scheduling in [17]. Care is taken that an optimum schedule length is obtained for every possible path in the CDFG, which is predominantly a CFG. A considerable improvement on this technique was later introduced in [18]. In the area of data-centric scheduling, a pioneering approach was introduced by Goossens et al. (included in this volume at page 107) [19]. Here the main attention is the scheduling of repetitive data-flow specifications containing nested loops and multi-dimensional signal arrays. This list based scheduling for DSP and its later adaptations are successfully implemented in commercial high-level synthesis systems. The architectural style for the first HLS systems was basically a multi-operator datapath with local register files and a rich instruction set. Register files are interconnected by a custom-designed network, and the instructions are generated by a (primitive) PLA based fixed instruction sequencer. Today we would call this a VLIW architecture with a fixed microcoded controller. These architectures could successfully handle DSP algorithms in the audio domain, given the CMOS technology available at that time, but were not powerful enough to meet the speed requirements of video,

graphics and/or radar applications. Highly pipelined and specialised data-paths with specialized instruction sets were needed for those tasks. It is remarkable that suddenly in 1989 a number of pioneering contributions in this domain were made. Two schools of thought emerged: Park and Kurdahi [20], and Hwang et al. [21] proposed to perform pipelined scheduling first, followed by allocation. Note et al [22] on the other hand, use the structure of nested-loop code to assemble single-cycle programmable data-path operators by clustering similarly structured CDFG sub-graphs into dedicated data path structures, and use the pipelined scheduling techniques afterwards. The latter method formed the basis for several high throughput compilers for video and high throughput systems such as the PHIDEO compiler from Philips Semiconductor [24]. Most of these compilers made use of the innovative force-directed scheduling technique proposed by Paulin in 1987 [23], which proved to be very well suited for time-critical pipelined scheduling. A considerable extension to this end (and its application in PHIDEO) was proposed in 1992 by Verhaegh et al. [24]. Most scheduling problems in system level design are subject to strong external timing and synchronisation constraints. This lead after 1994 to increased attention to a new approach using automata theory and symbolic techniques based on ROB-DDs. Pioneering work in this area was reported in [62] and [63]. On the other hand, it became apparent however that in multi-media architectures, the memory structures used to store the multidimensional signals often tend to dominate area, timing and power dissipation. This was first observed in two landmark papers at ICCAD-91 ([25,26]), and has led to a waterfall of contributions in the subsequent years (for instance, [27,28]). The 1994 contribution of Kolson et al. [29] is one of the first papers to recognize the importance of code transformations to reduce memory size (area) and data-transfers (power). Since then numerous contributions have been made that lead to dramatic performance improvement and power reduction in memory architectures. An in-depth overview of this impact was presented in a tutorial paper in 2000 [30]. While optimization of the memory itself is important, one should not ignore the address generation process. Research in the area of the automatic synthesis of address generators was triggered by a seminal paper by Grant and Denyer in 1988 [31] and was followed by others in subsequent years.

Similar to developments in optimizing compilers, researchers in the HLS world quickly realized that optimizing transformations are an essential component of any synthesis environment. In effect, the potential impact of transformations is even greater here, because of the broader design space and the wider variety of cost functions. Initial transformation and design space exploration systems (such as HYPER [32]) focused primarily on the minimization of traditional cost functions such as speed and area in data-path-like architectures. As mentioned in the preceding paragraph, this was later followed by transformational tools targeting the memory architecture (e.g. [30]). In the early 90s,

power minimization became a topic of the foremost importance. Pioneering research in low-power design techniques was performed at UC Berkeley in the infopad project. One of the major findings was that the biggest gains in power efficiency are obtained by appropriate transformations at algorithmic level combined with optimal supply voltage selection. This knowledge was implemented in the HYPER synthesis tool by Chandrakasan et al [33]. The paper, included in this volume at page 117, draws the attention to the fact that, for the first time, the supply voltage becomes an additional parameter dimension in the design exploration space. This has had great impact on the daily low power design practice. This pioneering paper inspired a chain of refinements (such as [34] and [35], just to mention a few).

In summary, it is fair to state that ICCAD contributed considerably to the basic design knowledge, methodology, and tools for the synthesis of fixed hardware co-processor architectures.

## 2.4 The Advent of Hardware-Software Co-Design

In the beginning of the nineties it became clear that the use of programmable cores, combined with hardware coprocessors communicating over customized bus networks, became the preferred target architecture for embedded applications rather than the hardwired special-purpose processor offered by HLS. In spite of an area, power and performance overhead, programmable components allow for a high degree of reuse, offer a higher system design productivity, allow for late-binding of function to component reducing the design risk, and enable upgrading in the field. Hence a new dimension in the design space was added: software embedded in a hardware architecture, quickly dubbed as hardware-software codesign (HSC).

This trend starts in 1992 with a tutorial and 3 papers in session 10. In the beginning most work concentrates on single processor-single accelerator co-processor architectures. After 1999, due to further integration capability, the attention shifts to multi-processor systems and performance estimation. This leads to a waterfall of seminal papers such as [55-57]. Starting in 1995 also the topic of dynamic power management of embedded systems pops up and in the years 1998-2002 dynamic voltage and frequency scaling become the most popular topics to control power dissipation in systems dominated dynamic software tasks. A good overview of work on dynamic power management can be found in [58, 59] by Benini and De Micheli while [60] by Shina and Chandrakasan is representative for the state of the art in dynamic voltage scaling.

Work in the multiprocessor area then leads to the concept of platform architectures, discussed in depth in a tutorial at ICCAD 2001. Since software is very energy consuming there is an increasing attention to characterizing and optimizing software power dissipation, a topic totally neglected by the traditional

computer science discipline. The term Hardware dependent Software (HdS) is launched. The main (initial) issues of research the HSC space are:

- Co-simulation of hardware and software
- Optimal hardware-software partitioning
- Interface synthesis
- Power and performance estimation and optimization of embedded software implementations

At ICCAD little attention was paid to the first topic and, although some papers at the conference still address the partitioning issue, today's HSC practice indicates that such partitioning is either clear from an application's standpoint or can best be found using an interactive system that allows the user to explore the design space either using high-level estimations [61] or by repeated synthesis as an estimation tool. In contrast, pioneering and influential work was presented at the conference in the latter two areas.

When putting multiple components of a heterogeneous nature together on a die, ensuring correct communication between them is of primary importance. Rather than relying on predefined interfaces, automatic synthesis of the interface is a far more effective approach. Groundbreaking work in this area was performed by Borriello and his co-workers. Their three papers at ICCAD on this topic truly form the foundation of all the efforts in this space ([36-38]). A wide range of papers have been published in subsequent years (for example, [39],[40]), some of which were the basis of EDA start-up's such as Coware. When put in the context of platform-based design, we see that interface synthesis evolves into the concept of communication-based design, which is perhaps one of the key disciplines to improve design productivity at the "system" level.

An important component of design-space exploration in the HSC space is the capability to quickly predict the performance and power dissipation of the software components. Creating easy-to-produce and reliable models is one of the main challenges here. One of the landmark papers in this area, included in this volume at page 129, is the 1994 contribution of Tiwari, Malik and Wolfe [41]. It addresses the power modelling of software and is based on a set of carefully chosen power consumption measurements of the individual instructions of a core processor. The method also includes the interaction between different instructions and estimates power losses due to cache misses. While pragmatic, the approach represents great progress with respect to a complete simulation of the processor at the gate and register level, and provides a very effective methodology for power modelling. It has since has been used substantially in academia and industry alike.

In 1995 the same research group proposes an ILP method, implemented in the CINDERELLA tool, to find a tight bound on the worst-case execution time of embedded software, including a direct-mapped instruction cache [42]. This work was extended upon by Ernst et al. in 1997 [43]. Using symbolic simulation and formal techniques, this paper further tightens the timing bounds. The resulting precision enables a substantial reduction in performance overhead, while leading to provably correct system and/or interface timing in embedded systems. Along the same lines, Mooney and DeMicheli [44] present synthesis techniques that map multi-task graph representations on a HW/SW architecture whereby an RTOS scheduler is synthesized such that hard real-time constraints are predictably met. In 1998, Li and Wolf [45] introduce the first hardware/software co-synthesis approach to optimize the memory hierarchy along with the rest of the architecture of a real-time distributed system. As caches and memories typically represent a significant fraction of the cost, size, weight, and power consumption of an embedded system, this approach can substantially reduce the system cost. Authors of the same group later extend the synthesis technique to include bus arbitration in single-bus multiprocessor embedded systems [46]. In the section on platform-based design, we will see that the current trend is to move away from single bus systems to networked multiprocessor architectures.

But first let us focus on another trend that is clearly visible at the level of the programmable processors themselves.

## 2.5 Synthesising and Programming Application-Specific Instruction Processors (ASIPs)

Throughout most of the System-on-a-Chip (SoC) era, designers have chosen to integrate IP (intellectual property) versions of traditional discrete processors into their systems. In recent years, it became clear that it is possible to obtain better performance for lower power dissipation by embracing domain-specific processors with greatly enhanced parallelism and specialised instruction sets. These processors are usually of the VLIW type. The crucial challenge then becomes to create a design methodology and tool set that allows for the description or synthesis of an Instruction Set Architectures (ISA) and the automatic generation of a retargetable compiler, which produces code of a quality and size rivalling a good assembly programmer. Pioneering work in this area goes back to the MIMOLA design system originally developed by Zimmerman and Marwedel [6], the FLEXWARE system of Paulin and Liem, and the CHESS system of Goossens et al. An excellent overview of this early work can be found in [47]. Several major contributions in this area were presented at ICCAD over the last decade.

In 1994, Huang and Despain [48] presented a method for combining instruction set design, microarchitecture design, and instruction-set mapping within a single formulation: a simultaneous scheduling/allocation problem with an integrated instruction formation process. The formulation takes as inputs the application, architecture template, objective function and design constraints, and generates as outputs the instruction set, resource allocation (which instantiates the architecture template) and assembly code for the application. Choi et al [49] introduce a new approach to generate application-specific instructions for DSP applications in 1998. The proposed approach supports for the first time multi-cycle complex instructions combined with single cycle instructions. A complete design system for ASIPs based on the LISA language, developed for the description of ISAs, was introduced in 2001 [50].

As stated earlier, VLIW architectures are popular for ASIPs as they allow a high degree of instruction-level parallelism, reflecting the concurrency that is typically present in the applications at hand. In particular, one paper presented at ICCAD on this topic deserves special attention. In a paper selected for this volume (page 159), Jacome et al. [51] present a design technique that creates an optimal VLIW architecture consisting of clusters of customized functional units with local register files communicating over an interconnect network. This technique allows an exploration of the design space where latency is traded off against power and clock speed. Given a kernel, the proposed algorithm explores the space of feasible clustered data paths and returns: a data-path configuration; a binding and scheduling for the operations; and a corresponding estimate for the best achievable latency over the specified family. It is demonstrated that clustering functional units around a register file, hence reducing the interconnect requirements, can have a striking effect on power and performance of the processor. It is interesting to observe how this paper extends concepts that were popular in the HLS for hardwired data-paths (e.g. [22]) to programmable VLIW processors.

## 2.6 The Advent of Platforms and Communication-Based Design

In the mid-nineties, we entered the age of deep-submicron and chips could now easily house a number of programmable processors connected to one or more buses, coprocessors, hardware accelerators (IP), and reconfigurable fabrics. While this creates enormous opportunities from an application and product perspective, it also poses formidable challenges. The design complexity of these components is quite daunting and leads to huge NRE costs. Add to that the huge mask costs associated with fabrication in 150 nm and below technology nodes. All these considerations advocate multiple reuse of the same chip or chip architecture for a given application domain.

The solution for these challenges is a design methodology called platform-based design that facilitates design reuse by abstracting hardware to a higher level (platforms) that is visible to the application software. The hardware platform should comprise a family of flexible (parameterizable) architectures that adequately support the functions in the application space. Also, a software platform is needed to abstract the hardware platform into a programmer's model to allow effective mapping. This union of hardware and software platforms is now called an architectural platform. Once a system platform has been identified for the application space and the architecture space, the final chip design involves design exploration within the architecture platform to determine the best mapping of application to architecture. While this approach is steadily emerging as a dominant trend in system-level design, it has received minor attention at ICCAD so far (besides a tutorial in 2001 [52]). The most prominent presence at the conference was in the area of communication-based synthesis. Interconnecting a large number of complex components on a single die, such that functionality, performance and power constraints are met, is one of the main challenges of System-on-a-Chip design. More and more, SOCs resemble networked multi-processor architectures. This so-called Network-on-a-Chip approach is just in its infancy, but has resulted in complete sessions at recent DAC and ICCAD conferences. The paper presented by Carloni et al. in 1999 at the conference [53], and included in this volume, puts an interesting perspective on how to address the on-chip global interconnect. It proposes a synthesis methodology for synchronous systems that makes the design functionally insensitive to the latency of long wires. Given a synchronous specification of a design, a functionally equivalent synchronous implementation that can tolerate arbitrary communication latency between designed registers is generated. Using automatically inserted registers, a long wire is broken into short segments which can be traversed in a single clock cycle. This results in a design that is robust with respect to delays of long wires. The method is demonstrated using an out-of-order microprocessor with speculative-execution. The important message of the paper is that the interconnect problem is most adequately addressed in a top-down fashion, interpreting the functional constraints and translating them into a network architecture that easily meets the circuit-level timing requirements. While currently in the early stages of research, we expect platform-based design to dominate the system-level design sessions at the conference in the years to come.

### 3. Quo Vadis (or Where are We Heading)?

Compared to most of the other topical areas in design automation, system-level design is a relative newcomer, having only a minor presence in the earlier ICCAD conferences. In addition, the commercial success of most of the early efforts in this space has been rather limited (to phrase it nicely). Earlier in

the paper, we identified the three necessary components for a successful design methodology in the system design space: a specification language or environment that covers the application space of interest, an effective target architecture, and the means to translate that architecture into a physical artifact. It is our conjecture that these components were not simultaneously present, except in some narrow application spaces where it is hard to recuperate the cost of the tool development. Looking at the present and the future, it seems that the time for effective and widespread system-level design methodology has finally come. The recent levels of understanding of the models-of-computation [54-56] enable the capture of system-level specifications in a general sense (getting away from the stovepipe systems of the past), programmable platforms present a clear and well-defined architectural target, and the route from architecture to silicon is well-understood (although under some challenging pressures at present).

What the future will hold is hard to surmise. However, looking down the semiconductor roadmap we can identify some important trends. Flexibility and reuse (often called programmability) will become ever more important, systems-on-chip (or systems-in-a package) will become ever more heterogeneous combining mixed signal and hybrid technologies. Power and energy constraints will limit the level of integration that can be reached as well as force us to look at innovative systemarchitectures, and the reliability and robustness of the integrated systems will be severely challenged. Expect these topics to be present in a big way in the ICCADs of the future.

## References

- [1] G. Goossens, I. Bolsens, B. Lin, and F. Catthoor, "Design of Heterogeneous ICs for Mobile and Personal Communication Systems," Embedded Tutorial ICCAD, Proc. ICCAD, pp.524-531, Nov. 1994.
- [2] P. Lippens, V. Nagasamy, and W. Wolf, "CAD Challenges in Multimedia Computing," Embedded Tutorial ICCAD, Proc. ICCAD, pp. 502-508, Nov. 1995.
- [3] K. Vissers, P. van der Wolf, and G. van Rootselaar, "System Level Design and Debug of High-Performance Embedded Media Systems," Embedded Tutorial ICCAD, pp. 461-466, Nov. 1999.
- [4] J. Rabaey, M. Potkonjak, F. Koushanfar, S. Li, and T. Tuan, "Challenges and Opportunities in Broadband and Wireless Communication Designs," Embedded Tutorial ICCD, pp. 76-81, Nov. 2000.
- [5] ICCAD 2001 Embedded Tutorial (M. Kamon, Moderator), "VLSI MicroSystems: The Power of Many", Embedded Tutorial ICCAD, Nov 2001.
- [6] G. Zimmermann, "The MIMOLA Design System: A Computer Aided Digital Processor Design Method", Proc. 16th Design Automation Conference, pp. 53-58, 1979.
- [7] A. C. Parker, D.E. Thomas, D.P. Siewiorek, M. Barbacci, L. Hafer, G. Leive and J. Kim, "The CMU Design Automation System: An Example of Automated Data Path Design," Proc. 16th Design Automation Conference, pp. 73-80, 1979.

- [8] R. Piloyt et al., "The CONLAN Project: Status and Future Plans," Proc. 19th Design Automation Conference, pp. 202-207, 1982.
- [9] M. Barbacci, "The Symbolic Manipulation of Computer Descriptions: An Introduction to ISPS," Tech. Rep. Dept. of Computer Science, Carnegie-Mellon University, March 1980.
- [10] R. Brayton and C. McMullen, "The Decomposition and Factorization of Boolean Expressions," Proc. ISCAS Conference, pp. 49-54, May 1982.
- [11] R. Brayton et al. "Multiple-Level Logic Optimization System," Proc. ICCAD, pp. 356-359, Nov. 1986.
- [12] R. Brayton, G. Hachtel, L. Hemachandra, A. R. Newton, A.R., A. Sangiovanni-Vincentelli, "A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization", Proc. ISCAS Conf., Rome, pp.42-48, May 1982.
- [13] D. Johansen, "Bristle Blocks: a Silicon Compiler," Proc. 16th Design Automation Conference, pp. 310-313, 1979.
- [14] M. Buric, T. Matheson, and O. Christensen, "Embedding Electrical and Timing Constraints in Hierarchical Circuit Layout Generators," Proc. ICCAD, pp. 3-5, Nov. 1983.
- [15] R. K. Brayton, R. Camposano, G. D. Micheli, R. Otten, and J. van Eijndhoven, "The Yorktown Silicon Compiler System," in Silicon Compilation (D. Gajski, ed.), pp. 204-310, Addison Wesley, 1988.
- [16] H. De Man, J. Rabaey, P. Six, L. Claesen, "CATHEDRAL-II : A Silicon compiler for Digital Signal Processing", IEEE Design and Test, pp. 13-25, Dec. 1986.
- [17] R. Camposano: "Path-Based Scheduling for Synthesis", IEEE Trans. Computer-Aided Design, Vol. 10, pp. 85-93, Jan. 1991.
- [18] G. Lakshminarayana, G. Khouri, N.K. Jha :"Wavesched: A Novel Scheduling Technique for Control-Flow Intensive Behavioral Descriptions", Proc. ICCAD, pp.245-251, Nov. 1997.
- [19] G.Goossens, J. Rabaey, J. Vandewalle and H. De Man "An Efficient Microcode-Compiler for Custom DSP-Processors" Proc. ICCAD, pp. 24-27, Nov. 1987.
- [20] N. Park, F. J. Kurdahi "Module Assignment and Interconnect Sharing in Register-Transfer Synthesis of Pipelined Data Paths" Proc. ICCAD Conf., pp. 16-19, Nov. 1989.
- [21] K.S. Hwang, A.E. Casavant, C. Chang, M.A. d'Abreu, "Scheduling and Hardware Sharing in Pipelined Data-paths", Proc. ICCAD, pp. 24-27, Nov. 1989.
- [22] S.Note, F.Catthoor, J.Van Meerbergen, H.De Man, "Definition and Assignment of Complex Data-Paths suited for High Throughput Applications", Proc. ICCAD, pp., 6-9 Nov. 1989.
- [23] P. Paulin and J. Knight, "Force-Directed Scheduling in Automatic Data Path Synthesis," Proc. 24th Design Automation Conference, pp.195-202, June 1987.
- [24] W. Verhaegh, P. Lippens, E. Aarts, J. Korst, A. van der Werf, and J. van Meerbergen, "Efficiency Improvements for Force-Directed Scheduling," Proc. ICCAD, pp. 286-289, Nov. 1992.
- [25] J. Vanhoof, I. Bolsens, and H. De Man, "Compiling Multi-Dimensional Data Streams into Distributed DSP ASIC Memory," Proc. ICCAD, pp. 272-275, Nov. 1991.
- [26] I. Ahmad and C. Chen, "Post-Processor for Data Path Synthesis Using Multiport Memories," Proc. ICCAD, pp. 276-279, Nov. 1991.
- [27] P. Lippens, J. van Meerbergen, W. Verhaegh, and A. van der Werf, "Allocation of Multiport Memories for Hierarchical Data Streams," Proc. ICCAD, pp. 728-731, Nov. 1993.

- [28] F. Balasa, F. Catthoor, and H. De Man, "Exact Evaluation of Memory Size for Multi-Dimensional Signal Processing Systems," Proc. ICCAD, pp. 669-672, Nov. 1993.
- [29] D. Kolson, A. Nicolau, and N. Dutt, "Integrating Program Transformations in the Memory-Based Synthesis of Image and Video Algorithms," pp. 27-30, Nov. 1994.
- [30] L. Nachtergael, V. Tiwari, and N. Dutt, "System and Architecture-Level Power Reduction for Microprocessor-Based Communication and Multimedia Applications," Proc. ICCAD, pp. 569-572, Nov. 2000.
- [31] D. Grant, P. Denyer, and I. Finlay, "Synthesis of Address Generators," Proc. ICCAD, pp. 108-111, Nov. 1988.
- [32] M. Potkonjak and J. Rabaey, "Optimizing Resource Utilization Using Transformations," Proc. ICCAD, pp. 88-91, Nov. 1991.
- [33] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen, "HYPER-LP: A System for Power Minimization Using Architectural Transformations," Proc. ICCAD, pp. 300-303, Nov. 1992.
- [34] A. Raghunathan and N.K. Jha, "An Iterative Improvement Algorithm for Low Power Data Path Synthesis", Proc. ICCAD, pp. 597-602, Nov. 1995
- [35] R. Mehra and J. Rabaey, "Exploiting Regularity for Low-Power Design," Proc. ICCAD, pp. 166-169, Nov. 1996.
- [36] P. Chou, R. Ortega, and G. Borriello, "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems," Proc. ICCAD, pp. 488-491, Nov. 1992.
- [37] P. Chou, R. Ortega, and G. Borriello, "Interface Co-Synthesis Techniques for Embedded Systems," Proc. ICCAD, pp. 280-287, Nov. 1995.
- [38] R. Ortega and G. Borriello, "Communication Synthesis for Distributed Embedded Systems," Proc. ICCAD, pp. 434-437, Nov. 1998.
- [39] J. Sun and R. Brodersen, "Design of System Interface Modules," Proc. ICCAD, pp. 478-481, Nov. 1992.
- [40] B. Lin and S. Vercauteren, "Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation," Proc. ICCAD, pp. 101-108, Nov. 1994.
- [41] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," Proc. ICCAD, pp. 437-445, Nov. 1994.
- [42] Y. Li, S. Malik, and A. Wolfe, "Performance Estimation of Embedded Software with Instruction Cache Modeling,", pp. 380-387, Nov. 1995.
- [43] R. Ernst and W. Ye, "Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification," Proc. ICCAD, pp. 598-601, Nov. 1997.
- [44] V. Mooney and G. De Micheli , "Real Time Analysis and Priority Scheduler Generation for Hardware-Software Systems with a Synthesized Run-Time System," Proc. ICCAD, pp. 605-608, Nov. 1997.
- [45] Y. Li and W. Wolf , "Hardware/Software Co-Synthesis with Memory Hierarchies," Proc. ICCAD, pp. 430-433, Nov. 1998.
- [46] D. Rhodes and W. Wolf, "Co-Synthesis of Heterogeneous Multiprocessor Systems Using Arbitrated Communication," Proc. ICCAD, pp. 339-342, Nov. 1999.
- [47] P. Marwedel And G. Goossens (ed.), "Code Generation for Embedded Processors," Kluwer Academic Publishers, 1995.
- [48] I. Huang and A. Despain, "Generating Instruction Sets and Microarchitectures from Applications," Proc. ICCAD, pp. 391-396, Nov. 1994.

- [49] Hoon Choi, In-Cheol Park, Seung Ho Hwang, Chong-Min Kyung, "Synthesis of Application Specific Instructions for Embedded DSP Software," Proc. ICCAD, pp. 665-668, Nov. 1998.
- [50] Andreas Hoffmann, Oliver Schliebusch, Achim Nohl, Gunner Braun, Oliver Wahlen, Heinrich Meyr, "Methodology for the Design of Application Specific Instruction Set Processors (ASIP) using the Machine Description Language LISA," Proc. ICCAD, pp. 625-630, Nov. 2001.
- [51] M. Jacome, G. De Veciana and V. Lapinskii, "Exploring Performance Tradeoffs for Clustered VLIW ASIPs," Proc. ICCAD, pp. 504-510, Nov. 2000.
- [52] ICCAD 2001 Embedded Tutorial (E. Sentovic, Moderator), "Platform-Based Design," Nov. 2001.
- [53] Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha, Alberto L. Sangiovanni-Vincentelli, "A Methodology for Correct-by-Construction Latency Insensitive Design," Proc. ICCAD, pp. 309-314, Nov. 1999.
- [54] E.Lee, A. Sangiovanni-Vincentelli, "Comparing Models of Computation", Proc. ICCAD, pp. 234-243, Nov. 1996
- [55] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele, "Representation of process mode correlation for scheduling". Proc. ICCAD pp.54-61, Nov., 1998.
- [56] L. Thiele, K. Strehl, Dirk Ziegenbein, Rolf Ernst, J. Teich, "FunState - An Internal Design Representation for Codesign" Proc. ICCAD, pp. 558-565, Nov. 1999.
- [57] P.Lieverse, T. Stefanov, P. van der Wolf, E. Deprettere, "System Level Design with Spade: an M-JPEG Case Study ", Proc. ICCAD, pp. 31-38, Nov. 2001
- [58] E-Y Chung, L. Benini, G. De Micheli, "Dynamic Power Management Using Adaptive Learning Tree", Proc. ICCAD, pp. 274-279, Nov. 1999
- [59] L. Benini and G. De Micheli, "Dynamic Power Management of Circuits and Systems: Design Techniques and CADTools", Kluwer, 1997.
- [60] A. Sinha, A. Chandrakasan, "Energy Efficient Real-Time Scheduling", Proc. ICCAD, pp.458-463 , Nov. 2001
- [61] G.F. Marchioro, J.M. Daveau, A.A.Jerraya, "Transformational Partitioning for Co-Design of Multiprocessor Systems", Proc. ICCAD. pp , Nov. 1997
- [62] C.N.Coehlo, G. De Micheli, "Dynamic Scheduling and Synchronization Synthesis of Concurrent Digital Systems under System-Level Constraints", Proc. ICCAD, pp. 175-181, Nov. 1994.
- [63] S. Haynal, F. Brewer, "Efficient Encoding for Exact Symbolic Automata-Based Scheduling", Proc. ICCAD, pp. 477-481, Nov. 1998.

# AN EFFICIENT MICROCODE-COMPILER FOR CUSTOM DSP-PROCESSORS \*

Gert Goossens<sup>1</sup>, Jan Rabaey<sup>2</sup>, Joos Vandewalle<sup>3</sup> and Hugo De Man  
*IMEC Laboratory*  
*B-3030 Leuven, Belgium*

## Abstract

In this paper, a microcode compiler for custom DSP-processors is presented. This tool is part of the CATHEDRAL II silicon compiler. Two optimization problems in the microcode compilation process are highlighted : microprogram scheduling and memory allocation. Algorithms to solve them, partly based on heuristics, are presented. Our compiler successfully handles repetitive programs, and is able to decide on hardware binding. In most practical examples, optimal solutions are found. Whenever possible, indications of the complexity are given.

## 1. Introduction

The *CATHEDRAL II* silicon compiler aims at the automatic synthesis of synchronous multi-processor ASICs for DSP-applications [3] [4]. It proposes a full separation between *architecture synthesis* and *layout generation (meet-in-the-middle philosophy)*. The architecture synthesis tools transform an applicative behavioural representation of a DSP-algorithm, into a structural description of multi-processor data paths [3], and a procedural microcoded controller definition. In a first step, a customized data path, composed of well-characterized parametrizable modules, is generated by the *rule-based synthesis* program *JACK-THE-MAPPER* [9]. At the same time, the behavioural representation is translated into an applicative register-transfer (RT) description. In a second step, the *microcode compiler ATOMICS*<sup>1</sup> transforms the applicative RT-program into a procedural finite-state machine description. The output of this tool is linked with a PLA-based controller compiler, performing state optimization and layout generation, and with an RT-level simulator. In this paper we will describe the microcode compiler *ATOMICS*, with emphasis on efficient optimization algorithms for *microprogram scheduling* (section 2) and *memory allocation* (section 3).

The applicative RT-input language of *ATOMICS* is characterized by a highly architecture-independent format, and powerful control constructs, e.g. *FOR*-loops and conditional statements. In many compilers, loop constructs are not

---

\*Research supported by the ESPRIT 97 program of the EC.

<sup>1,2,3</sup>Authors are currently with <sup>1</sup>Target Compiler Technologies, <sup>2</sup>University of California at Berkeley and

<sup>3</sup>Katholieke Universiteit Leuven.

supported since they cause certain optimization problems to become hard. In the sequel, RT-programs (not) containing loops will be referred to as (*non-*) *repetitive programs*. *ATOMICS* is based on a parametrizable multiple-branch controller model with horizontal microcode, allowing for simultaneous data path actions and jump address computation [3]. The current version of *ATOMICS* does not perform any transformations that create, delete or alter RTs from the input description.

## 2. Microprogram scheduling

In order to capture the *time*-concept in a repetitive RT-program, the *potential* of an RT is defined as the RT's machine-cycle number in an imaginary implementation, in which the body of every *FOR*-loop is executed once. The goal of *microprogram scheduling* then is to map the individual RTs to potentials, thereby exploiting the inherent parallelism in the DSP-algorithm in order to *minimize the global machine-cycle count*. In order to reduce the complexity of the scheduling problem for repetitive RT-programs, loops are scheduled hierarchically, starting at the deepest level of nesting.

### 2.1 A constrained optimization problem

The search space of the optimization is bounded by three categories of constraints :

- 1 *Forward data-precedences*, expressing the need for a minimal (integer) delay  $\delta$  between any pair of RTs  $A$  and  $B$ , respectively *writing* and *reading* a variable to/from a storage element. This can be denoted :

$$p_B(j) \geq p_A(j) + \delta(j) \quad (1)$$

where  $p_A(j)$  and  $p_B(j)$  represent the potentials of  $A$  and  $B$  respectively, in constraint  $j$ . Usually,  $\delta$  equals 1 cycle. Dependencies between conditional RTs and RTs producing the according condition variables are also modelled in this way. In this case  $\delta$  is usually larger than 1, due to the internal controller pipelining.

- 2 *Resource-allocation constraints* (or *conflicts*), expressing the requirement that certain RTs sharing hardware resources (e.g. data path operators, memory, busses) cannot be scheduled at the same machine cycle. If  $C$  and  $D$  represent the conflicting RTs in constraint  $i$ , the latter can be denoted :

$$p_C(i) \neq p_D(i) \quad (2)$$

- 3 *Looping data-precedences*, expressing a precedence relation in a *FOR*-loop, between a *writing* RT in the current loop iteration and a *reading* RT

in the next iteration. Looping precedence  $j'$  between RTs A and B can be denoted :

$$p_B(j') \geq p_A(j') + \delta(j') - (\Delta p + 1) \quad (3)$$

where  $\Delta p + 1$  is the total number of potentials spanned by the loop. (3) is only relevant when  $\delta(j') > 1$ .

No constraints occur between conditional RTs with disjoint condition fields. Two graph representations will be used (e.g. Fig. 1(a)). In both cases, RTs are modelled as vertices. Forward data-precedences are modelled as arcs in a directed weighted graph, from the writing to the reading RT's vertex, with a weight equal to the delay (*data-precedence graph*). Resource-allocation constraints are represented as arcs in an undirected graph (*resource-allocation graph*).

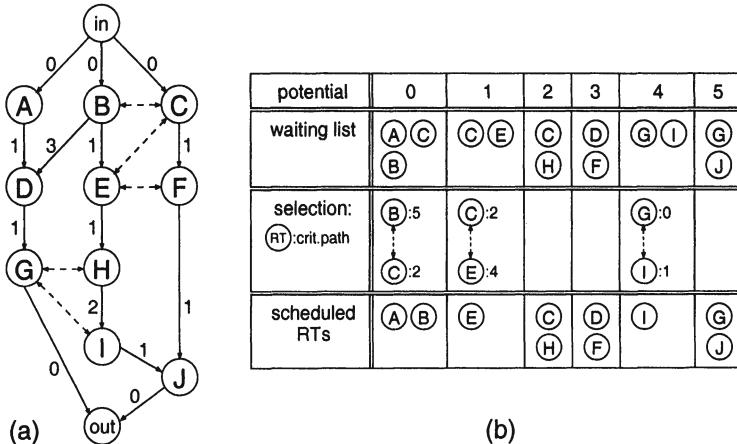


Figure 1. Scheduling of non-repetitive RT-program : (a) data-precedence graph (solid) and resource-allocation graph (dotted). Numbers indicate weights of data-precedences; (b) scheduling based on critical path criterion for conflict resolution.

Scheduling problems are known to be NP-complete. (1), (2) and (3) can e.g. be reformulated as an *integer linear program* (ILP) [3]. Efficient *heuristic scheduling algorithms* from the theory of *project management* [2] have been applied to the DSP-domain by Zeman [10], for the subclass of *non-repetitive RT-programs*. We will present an iterative scheduling algorithm, based on [10], which allows to schedule *repetitive programs* in an automatic and efficient way. In this paper, we assume that storage elements in the data path may contain multiple storage-fields (e.g. RAM, ROM, local register-files). Although not described, our iterative scheduling technique allows to model other memory elements too, such as pipeline latches.

## 2.2 Scheduling non-repetitive programs

We will review Zeman's scheduling technique, for non-repetitive programs, as an extension of classical levelling of the data-precedence graph. The levels of the vertices, correspond to the potentials assigned to the RTs. A global search method (levelling), used to take into account the forward data-precedences, is combined with local heuristic searches to take into account the NP-hard resource-allocation constraints. All potentials are treated consecutively, starting from 0. Inspection of the data-precedence graph reveals which RTs are ready to be assigned the current potential. These RTs are placed in a waiting list. If resource-allocation conflicts occur, a heuristic selection criterion is used to derive a conflict-free subset from the waiting list. The non-selected RTs are shifted to the waiting list of the next (higher) potential. In *ATOMICS*, a *critical-path criterion* is used : the scheduling priority of an RT involved in a conflict at the current potential, is measured by the length of the critical path in the data-precedence graph, emerging from the RT. Modifications of the criterion are discussed in [6]. The algorithm is illustrated in Fig. 1(b). Compared to a levelling algorithm, only minor extra CPU-time is spent : the complexity of a critical path analysis is linear in the number of vertices in the subgraph under consideration.

## 2.3 Scheduling repetitive programs

Below, we will extend Zeman's scheduling algorithm, in order to schedule *repetitive programs*. An *iterative procedure* allows to take into account the additional *looping precedences*. The convergence is guaranteed, and acceptable bounds on the number of iterations can be derived.

In the algorithm for *non-repetitive* RT-programs, the *forward precedences* (1) can be interpreted as a specification of a *lower bound on the potential*  $p_B$  of an RT  $B$  which is ready to be scheduled. This bound is a function of the potentials  $p_A(j)$  of all precedent RTs, i.e. all RTs pointing to  $B$  via a forward precedence, and of the associated delays. By definition,  $B$  is ready to be scheduled when all of its precedents  $A$  have been scheduled; hence at this point the bound can be computed explicitly, and equals the potential of the waiting list in which  $B$  will occur for the first time. In a *repetitive* program, an additional bound on  $p_B$  is provided by the *looping precedences* (3). In this expression,  $\Delta p$  can be estimated (see below); the potentials  $p_A(j')$  are however unknown during the treatment of the theoretically optimal potential of  $B$ . Therefore, the bound cannot be computed explicitly. In *ATOMICS*, this problem is solved by calling Zeman's algorithm repeatedly in an iteration process : in order to compute the lower bounds (1) and (3) on  $p_B$  during iteration  $k$ , the potentials  $p_A(j')$  are taken from the previous iteration  $k - 1$ , whereas the potentials  $p_A(j)$  are taken from the current iteration, since they are known in time.

According to experiments, the convergence of the iteration process is critically dependent on the estimate of  $\Delta p$  [6] : a too small choice of  $\Delta p$  results in an empty solution space for the optimization, which is manifested as divergence of the iteration. Divergence is *observable* from certain vertices in the graph [6]. A too large choice of  $\Delta p$  results in convergence to a sub-optimal solution. A reliable strategy to determine the optimal  $\Delta p$  is to call the previous algorithm repeatedly in a *second iteration loop* over  $\Delta p$ . The optimal  $\Delta p$  is determined in a limited number of steps, via a *binary* or *incremental* search between a reliable lower and upper bound on  $\Delta p$ . An upper bound on  $\Delta p$  is e.g. the sum of all delays in the precedence graph; this is then also a (pessimistic) bound on the number of outer-loop iterations.

An example is given in Fig. 2. With our algorithm, the optimal schedule has been found in numerous designs. Applications include a 4-processor pitch-extractor for speech, an adaptive interpolator for error correction in Compact Disc, and an echo canceller for digital telephony. Some results are listed in Table 1.

RT-system	pitch-extractor		
	pitch computation processor		recognition processor
# RTs	60		66
# forward data-precedences	188		162
# looping data-precedences	51		169
# resource-alloc. conflicts	46		98
# potentials	15		16
cycle count	137		3523
FOR-loop indices	time	i	j
# inner iterations	1	2	1
CPU-time (VAX8600/VMS)	0.21 s		1.27 s

Table 1. Design figures for ATOMICS scheduler.

## 2.4 Automated hardware-assignment during scheduling

A designer often might wish to use multiple instances of a certain hardware operator in the data path (e.g. ALUs, multipliers,...). The decision on the number of instances of each operator will be termed the *hardware allocation*. When multiple instances of an operator have been allocated, the next step in the design is to bind each RT to one or more specific instances. This problem will be termed the *hardware assignment*. Below, an optimized hardware-assignment technique will be introduced, which is based on the scheduling algorithm. Full details can be found in [7]. For a given allocation, ATOMICS will try to find an assignment which minimizes the machine-cycle count. For this purpose, an

(a)  $\Delta p^{(0)} = 5$  (initial estimate based on non-repetitive schedule)  $p_{\odot}^{(0,0)} = 0$

potential $p^{(0,1)}$	0	1	2	3	4	5	6
waiting list	(A) (B) (C)	(C) (E)	(C) (H)	(D) (F)	(G) (I)	(I)	(J)
selection: sel.idx.	(RT) 0.	(B) $\rightarrow$ (C) 1.	(C) $\rightarrow$ (E) 0.667			(G) $\rightarrow$ (I) -2.	0.
scheduled RTs	(A) (B)	(E)	(C) (H)	(D) (F)	(G)	(I)	(J)

$\Delta p^{(0)}$  exceeded (divergence) 

(b)  $\Delta p^{(1)} = 6$  (incremental search)  $p_{\odot}^{(1,0)} = 0$

potential $p^{(1,1)}$	0	1	2	3	4	5	6
waiting list	(A) (B) (C)	(C) (E)	(C) (H)	(D) (F)	(G) (I)	(I)	(J)
selection: sel.idx.	(RT) 0.167	(B) $\rightarrow$ (C) 1.333	(C) $\rightarrow$ (E) 1.	(0.2)		(G) $\rightarrow$ (I) -1.	0.5
scheduled RTs	(A) (B)	(E)	(C) (H)	(D) (F)	(G)	(I)	(J)

 looping precedence (G) (A) not satisfied 

$p_{\odot}^{(1,1)} = 0 \quad p_{\odot}^{(1,1)} = 1$

potential $p^{(1,2)}$	0	1	2	3	4	5	6
waiting list	(B) (C)	(A) (C) (E)	(C) (H)	(D) (F)	(G) (I)	(I)	(J)
selection: sel.idx.	(RT) 0.167	(B) $\rightarrow$ (C) 1.333	(C) $\rightarrow$ (E) 1.	(0.2)		(G) $\rightarrow$ (I) 0.	0.5
scheduled RTs	(B)	(A) (E)	(C) (H)	(D) (F)	(G)	(I)	(J)

 convergence reached 

Figure 2. Scheduling of repetitive program, obtained from Fig. 1, by adding looping precedences  $G - A$  and  $out - in$ , with delays 4 and 1 respectively : (a) 1st outer iteration step, consisting of 1 inner iteration step ; (b) 2nd outer iteration step, consisting of 2 inner iteration steps. (The selection index is a modified critical-path measure.)

RT input-description has to be provided in which all RTs are formally using *differently named operator-instances*, called *formal instances*. *ATOMICS* will then *merge* these formal instances into the actual number of instances, specified in the *allocation*. Automated hardware-assignment is non-trivial with architectures in which single RTs may cover multiple operators. E.g. in *CATHEDRAL II* data paths, register-files are placed locally at the inputs of operators [3]; therefore the assignment of each RT requires choosing both a *source* operator (for fetching and modifying source data) and a *destination* operator (for storing the result). In general, the assignment of RTs involved in a data-precedence, cannot be performed independently of each other. This fact encumbers the locality of the search process.

In order to combine hardware assignment with scheduling, two *types of resource-allocation conflicts* have to be distinguished : *soft* conflicts, occurring between RTs covering different formal instances of an operator; and (classical) *hard* conflicts, occurring between RTs sharing the same actual instance of an operator. RTs which are involved in a soft conflict, may be scheduled on the same potential, as long as sufficient actual operator-instances are available to execute them in parallel. In this case, the final assignment should obey the condition that the formal instances which caused the soft conflict, may not be merged into one actual instance. Such condition is termed a *merging constraint*.

As in section 2.2, *ATOMICS* will process consecutive potentials, starting from 0. At every potential, with the help of the critical path criterion, a maximal subset of RTs is derived from the waiting list, which contains a) no hard conflicts and b) only soft conflicts that don't introduce an *inconsistent* (unsolvable) set of *merging constraints* for the given allocation. A set of merging constraints can be represented as arcs in an undirected graph, termed *merging graph*, with formal operator-instances as vertices. The problem of checking the consistency of a set of merging constraints at every potential [7] is equivalent to finding an acceptable *vertex colouring* [5] of the merging graph, in which the number of colours corresponds to the the number of allocated operators. Although vertex colouring is NP-complete, the limited size of the merging graph usually allows to apply a simple *branch and bound* algorithm [7]. Every acceptable vertex colouring, of the merging graph obtained after completion of the scheduling operation, provides a valid hardware-assignment. The cheapest colouring in terms of interconnection cost can then be chosen.

### 3. Memory allocation

When the RT-program has been scheduled, an allocation of variables to storage fields can be carried out, aiming at a minimal number of fields. In *ATOMICS*, this allocation is used to dimension register-files in the data path. The set of

potentials during which a variable is to be stored is called the variable's *lifetime* [1]. The lifetime of a variable in a non-repetitive program can be modelled as an *integer interval* (Fig. 3). In the repetitive case however, lifetimes may have to be defined as the *union of non-overlapping intervals* [6] (Fig. 4). One storage field may be allocated to different variables if they have disjoint lifetimes.

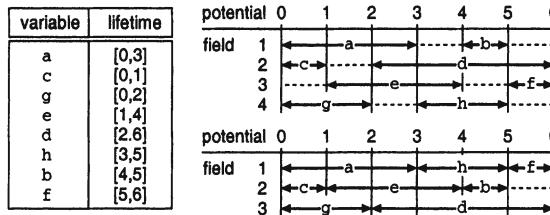


Figure 3. Lifetime intervals of variables in a register-file, an arbitrary (top) and an optimal register-allocation (bottom).

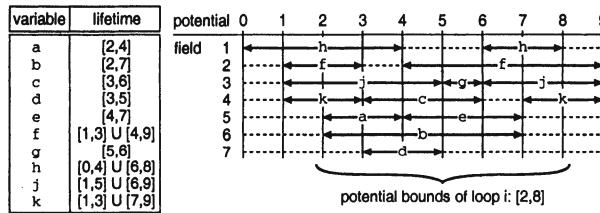


Figure 4. Register-allocation for repetitive program.

The general memory allocation problem is equivalent to *clique partitioning*, which is NP-complete [8]. However, for the class of non-repetitive programs, a *globally optimal* allocation can be found in *linear* time, with the algorithm presented in Table 2. For an optimality proof, we refer to [6]. The analysis is demonstrated in Fig. 3 (bottom).

This algorithm requires that lifetime intervals be continuous. For the class of repetitive programs, ATOMICS uses an efficient heuristic procedure, incorporating the previously described algorithm. This procedure consists of two steps :

- Consider the variables with a lifetime composed of several non-overlapping intervals. A heuristic graph-based algorithm [6] allows to “fill the gaps” in these lifetimes, with smaller continuous lifetime intervals of other variables. Both the “filling” and the “filled” variables are assigned to the same storage-field. In this way, discontinuous lifetimes are formally replaced by single continuous intervals.

---

$p :=$  minimal potential;  
 create new storage-field  $f$ ;  
 while there exist unassigned variables do  
     if there exist unassigned variables  $v_i$ , coming alive at  $p$  then  
         select arbitrary  $v_i$ ;  
         assign  $v_i$  to  $f$ ;  
          $p :=$  endvalue of  $v_i$ 's lifetime  
     else  
          $p := p + 1$   
     end if;  
     if  $p >=$  maximal potential then  
          $p :=$  minimal potential;  
         create new storage-field  $f$   
     end if  
 end do.

---

Table 2. Optimal register-allocation for non-repetitive program.

- Next, the optimal allocation algorithm for non-repetitive programs is applied.

With this procedure we succeeded in finding the optimum in almost any practical design, in very low CPU-time. An example is given in Fig. 4. As has been observed in [8], direct application of general heuristics for clique partitioning often produces suboptimal solutions to the memory allocation problem. In Fig. 5, a special configuration is shown, for which the results of the basic clique partitioning algorithm presented in [8] are compared with our technique.

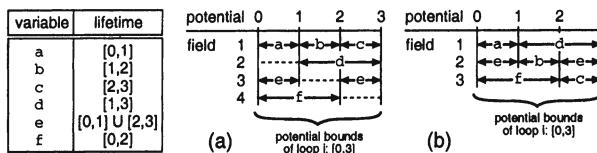


Figure 5. Example of register-allocation, comparing our technique (b) with the approach described in [8] (a).

## 4. Conclusions

The microcode compiler *ATOMICS* for custom DSP-systems has been presented. The problems of scheduling (with an extension towards automated hardware binding) and memory allocation, have been emphasized. Efficient algorithms have been presented, which are partly based on heuristics, to compute quasi-optimal solutions in very low CPU-time. Loop-constructs in the applicative input description are supported. Current research is aimed at further

optimization of repetitive programs, e.g. by allowing overlaps in time between successive loop-iterations. Furthermore, it is investigated how memory optimization can be taken into account during scheduling.

## Notes

1. "A Tool for Optimized Micro-Instruction Compilation and Scheduling".

## References

- [1] A.V. Aho, J.D. Ullman, "*Principles of compiler design*", Addison-Wesley, Reading, 1977.
- [2] E.G. Coffman Jr, "*Computer and job scheduling theory*", J. Wiley and Sons, New York, 1976.
- [3] H. De Man et al., "CATHEDRAL II : a synthesis and module generation system for multiprocessor systems on a chip", *Proc. NATO-ASI on log. synth. and silicon compilation for VLSI-design*, L'Aquila, July 1986.
- [4] H. De Man et al., "CATHEDRAL II : a silicon compiler for digital signal processing", *IEEE Design and Test*, pp. 13-25, Dec. 1986.
- [5] A. Gibbons, "*Algorithmic graph theory*", pp. 198-201, Cambridge Univ. Press, Cambridge, 1985.
- [6] G. Goossens et al., "An efficient microcode-compiler for custom multiprocessor DSP-systems", *Technical report ESPRIT97/IMEC/3.87/D/d(4)/1*, available from IMEC vzw, Leuven, March 1987.
- [7] G. Goossens, "Techniques for automated hardware assignment", *Technical report ESPRIT97/IMEC/9.87/D/d(4)/1*, available from IMEC vzw, Leuven, Sept. 1987.
- [8] C.J. Tseng, D.P. Siewiorek, "Automated synthesis of data paths in digital systems", *IEEE Trans. CAD*, pp. 379-395, July 1986.
- [9] J. Vanhoof et al., "A knowledge-based CAD system for synthesis of multi-processor digital signal processing chips", *Proc. VLSI'87*, Vancouver, Aug. 1987.
- [10] J. Zeman, G.S. Moschytz, "Systematic design and programming of signal processors, using project management techniques", *IEEE Trans. ASSP*, pp. 1536-1549, Dec. 1983.

# **HYPER-LP: A SYSTEM FOR POWER MINIMIZATION USING ARCHITECTURAL TRANSFORMATIONS**

Anantha P. Chandrakasan<sup>1</sup>

*EECS Department,*

*University of California at Berkeley*

Miodrag Potkonjak<sup>2</sup>

*C & C Research Laboratories,*

*NEC USA, Princeton*

Jan Rabaey and Robert W. Brodersen

*EECS Department,*

*University of California at Berkeley*

## **Abstract**

The increasing demand for “portable” computing and communication, has elevated power consumption to be the most critical design parameter. An automated high-level synthesis system, HYPER-LP, is presented for minimizing power consumption in application specific datapath intensive CMOS circuits using a variety of architectural and computational transformations. The sources of power consumption are reviewed and the effects of architectural transformations on the various power components are presented. The synthesis environment consists of high-level estimation of power consumption, a library of transformation primitives (local and global), and heuristic/probabilistic optimization search mechanisms for fast and efficient scanning of the design space. Examples with varying degree of computational complexity and structures are optimized and synthesized using the HYPER-LP system. The results indicate that an order of magnitude reduction in power can be achieved over current-day design methodologies while maintaining the system throughput; in some cases this can be accomplished while preserving or reducing the implementation area.

## **1. Introduction**

The major VLSI design and research efforts until now have been focused on optimizing speed to realize computationally intensive real-time tasks such as video compression and speech recognition. As a result, many systems have successfully integrated various complex signal processing modules meeting users computation and entertainment demands. While these solutions have provided

---

Authors are currently with <sup>1</sup>Massachusetts Institute of Technology and <sup>2</sup>Univ. of California at Los Angeles.

answers to the real-time problem, they have not addressed the rapidly increasing demand for portable operation. This strict limitation on power dissipation which portability imposes, must be met by the designer while still meeting ever higher computational requirements.

An example of a system requiring portability, moving beyond today's portable computers, is a future personal communications terminal described in [1] that will support speech communication and recognition, data transfer, computing services, and high-quality, full-motion video. The intense computational nature of the terminal functions coupled with the requirement of portability will place severe constraints on the total power being consumed. For example, a basic terminal with speech recognition and video decompression units implemented using current day technology will require about 20lbs of battery for 10hrs of operation [1]. Clearly, more power efficient means for implementing these functions need to be developed. One major degree of freedom available in optimizing design for such applications is that once real-time operation is achieved, there is *no* advantage in making the computation any faster. The goal then becomes one of reducing the power consumption while maintaining the system throughput.

Fortunately, the increasing density of VLSI systems, due to sub-micron feature size scaling and high-density packaging such as multichip modules, has enabled the development of an architectural strategy which can be used to trade-off area and power for a fixed throughput [2].

In this work, we attack the problem of automatically finding computational structures that results in the lowest power consumption for a specified throughput given a high-level algorithmic specification. The basic approach is to scan the design space by utilizing various flowgraph transformations, high-level power estimation, and efficient heuristic/probabilistic search mechanisms. While transformations have been successfully applied recently in high-level synthesis with the goal of optimizing speed and/or area, the problem of power optimization has not been addressed. It will be shown that optimizing for power using transformations requires a different strategy than those used for speed or area optimization.

## 2. Sources of Power Consumption

In CMOS technology, there are three sources of power dissipation arising from: switching (dynamic) currents, short-circuit currents, and leakage currents. The switching component, however, is the only one which cannot be made negligible if proper design techniques are followed. The power consumption due to the switching of a CMOS gate with a load capacitor,  $C_L$ , is given by the following formula [3]:

$$P_{switching} = P_t(C_L * V_{dd}^2 * f) \quad (1)$$

where  $f$  is the clock frequency,  $V_{dd}$  is the supply voltage, and  $p_t$ , is the probability of a power consuming transition (or the activity factor). Probabilistic

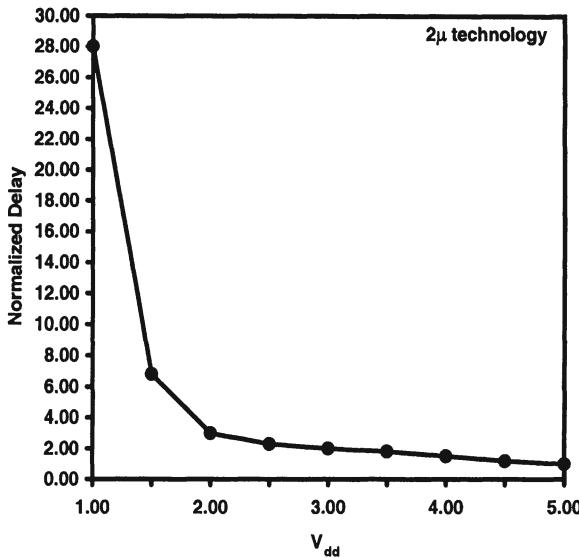


Figure 1. Plot of normalized delay vs.  $V_{dd}$ .

approaches have been proposed to estimate the internal node activities of a network given the distribution of the input signals (i.e. the probability the value is a “1” or “0”) [4, 5].

In our analysis and optimizations, we will refer to the energy per computation of a gate or module (e.g. an adder), which is given by:

$$\text{Energy} = P_{total}/f_{clk} = C_{avg}V_{dd}^2 \quad (2)$$

where  $C_{avg}$  is the average capacitance being switched per clock cycle (i.e  $C_{avg} = p_t * C_{Ltotal}$ ).

The energy consumed by a logic block per computation is therefore a quadratic function of the operating voltage, as verified experimentally for a number of logic functions and logic styles in [2]. It is clear that operating at the lowest possible voltage is most desirable, however, this comes at the cost of increased delays and thus reduced throughput. This is seen from Figure 1 which shows an experimentally derived plot of normalized delay vs.  $V_{dd}$  for a typical CMOS gate. Once again, the delay dependence on supply voltage was verified to be relatively independent of various logic functions and logic styles [2].

By modifying the architecture through a variety of transformations, however, the throughput can be regained, and thus a power savings can be accomplished while retaining the required functionality. It is also possible to reduce the power by choosing an architecture that minimizes the effective capacitance; through reductions in the number of operations, the average transition activity, the interconnect capacitance, and internal bit widths and using operations that require

less energy per computation. It is these two strategies which will be pursued to minimize the power dissipation. There is, however, a strong interaction between optimizing capacitance and voltage for a fixed throughput. A lot of transformations will have conflicting effects on these parameters making the optimization a non-trivial task.

### 3. Transformations for Optimizing Power

Transformations are changes to the computational structure in a manner that the input/output behavior is preserved. The use of transformations makes it possible to explore a number of alternative architectures and to choose those which result in the lowest power. A brief summary of transformations for optimizing power is presented in this section. A detailed discussion of the effects of transformations on power is presented in [6].

#### 3.1 Critical Path Reduction

This is probably the single most important type of transformations for power reduction. It is not only the most common type of transformation, but also often has the strongest impact on power. The basic idea is to reduce the critical path, so that supply voltage can be lowered while keeping the throughput fixed. The reduction of critical path is most often possible due to the exploitation of concurrency. Many transformations profoundly affect the amount of concurrency in the computation including pipelining and loop unrolling.

To illustrate the reduction of power using speedup techniques, consider a module with capacitance  $C$  running at a maximum frequency of  $f @ 5V$  ( $\rightarrow P_{ref} = C(5)^2 f$ ). By transforming this structure to a parallel architecture with two identical units (unrolling), the clock frequency can be dropped to half the original rate while maintaining the original throughput. Since the modules have twice the available time as the original case, the voltage can be dropped to 2.9V (where the delays increase by a factor of 2, Figure 1). The power of the transformed solution is  $P_{par} = 2C(2.9)^2 f/2$  and a factor of  $(5/2.9)^2$  reduction in power is achieved without sacrificing performance. The above analysis assumed that there is no overhead for parallelizing. In reality, the overhead due to routing and control must be taken into account. Even so, the quadratic dependence of voltage on power usually more than compensates for the increase in capacitance resulting in an overall reduction of power. However at very low voltages ( $< 1.5V$ ), the delays (and hence the overhead circuitry) increase very rapidly, causing the power to increase with further reduction of the supply voltage [2].

### 3.2 Reducing the Number of Operations

The most obvious approach to capacitance reduction, is to reduce the number of operations (and hence the number of switching events) in the data control flow graph. While this almost always has the effect of reducing the effective capacitance, the effect on critical path is case dependent. Transformations which directly reduce the number of operations in a data control flow graph include: common subexpression elimination, manifest expression calculation, loop merging, and distributivity.

### 3.3 Reducing the Transition Activity

Designs using static CMOS logic can exhibit spurious transitions due to finite propagation delays from one logic block to the next (called critical races or dynamic hazards). i.e. a node can have multiple transitions in a single clock cycle before settling to the correct logic value. The amount of extra transitions is a complex function of logic depth, input patterns, and skew. To minimize the “extra” transitions and power in a design, it is important to balance all signal paths and reduce the logic depth. For example, consider the two implementations for adding four numbers shown in Figure 2 (assuming a non-pipelined implementation). Assume that all primary inputs arrive at the same time. Since there is a finite propagation delay through the first adder for the chained case, the second adder is computing with the new C input and the previous output of  $A + B$ . When the correct value of  $A + B$  finally propagates, the second adder recomputes the sum. Similarly, the third adder computes three times per cycle. In the tree implementation, however, the signal paths are more balanced and the amount of extra transitions is reduced. The capacitance switched for a chained implementation is a factor of 1.5 larger than the tree implementation for a four input addition and 2.5 larger for an eight input addition. The above simulations were done on layouts generated by the LagerIV silicon compiler [7] using the IRSIM [8] switch-level simulator over 1000 random input patterns.

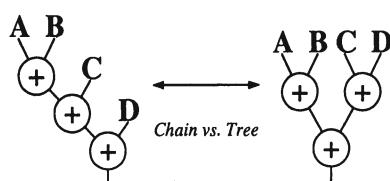


Figure 2. Reducing the glitching activity.

### 3.4 Reducing the Interconnect Capacitance

It is often possible to reduce the required amount of hardware, while preserving the critical path (or number of control steps) [9]. This is possible because after certain transformations, operations are more uniformly distributed over the available time, and thus a denser scheduling (effective utilization of the hardware) can be achieved. These transformations include retiming for resource utilization, associativity, distributivity and commutativity. Smaller capacitance is achieved because there are fewer interconnects and/or fewer functional elements and registers, which are obstacles during floor-planning and routing, which indirectly influence interconnect area and capacitance.

### 3.5 Operation Substitution

Certain operations inherently require less energy per computation than other operations. A prime example of this is strength reduction, often used in software compilers, in which multiplications are substituted for additions.

Another powerful transformation in this category is converting multiplications with constants into shift-add operations. Since multiplications with fixed coefficients are quite common in many signal processing applications (DCT, FFT, various types of filters, etc.), this transformation can prove to be beneficial.

### 3.6 Bit-width Optimization

The number of bits used can strongly affect all the key parameters of a design, including speed, area and power. A smaller bit-width typically results in fewer switching events (and hence lower capacitance), faster circuits (and hence lower supply voltage), and smaller area (and hence lower average interconnect length). Certain transformations (e.g. associativity and distributivity) can have a profound impact on the bit-width.

## 4. Power Estimation

The goal is to develop an objective function that is highly correlated to the final (and unknown) power dissipation of the circuit. The objective function should be very easy to compute since it has to be evaluated many times during the optimization process. Elaborate power estimation, while being much more accurate, will require the hardware mapping and compilation steps to convert a flowgraph to layout, making it impractical during the optimization process. Hence a model correlated to the power must be developed strictly from the flowgraph level.

The goal of power optimization in this work is to keep the throughput constant by allowing the supply voltage to vary. Given that the sample period is fixed, power optimization is equivalent to minimizing the total energy switched,

$C_{total}V^2$ , where  $V$  is appropriate voltage required to meet the initial throughput rate.

## 4.1 Capacitance Estimate

Estimating the total capacitance being switched involves considering four components:

$$C_{total} = C_{exu} + C_{registers} + C_{control} + C_{interconnect} \quad (3)$$

The capacitance estimation is built on top of an existing estimation routine in HYPER that determines the bounds and activity of various execution, register and interconnect components as well as the active implementations area [10]. A brief description of the estimation routines is presented below.

**4.1.1 Execution Units and Registers.** The capacitance contributed due to the execution units is determined by multiplying (over all types of units utilized) the number of times the operation was performed per sample period with the average capacitance of the unit type. The total capacitance is hence given by:

$$C_{exu} = \sum_{i=1}^{numtypes} N_i \cdot C_i \quad (4)$$

where  $numtypes$  is the total number of operation types,  $N_i$  the number the times the operation of type  $i$  is performed per sample period, and  $C_i$  is the average capacitance being switched per operation of type  $i$ . The average capacitance for the various modules has been characterized as a function of bit-width (through SPICE and IRSIM simulations) for a uniformly distributed set of inputs. In general the probabilities are not uniform, however, this assumption is made to simplify the cost evaluation. The effect of inter-module capacitance (between modules inside a datapath) is taken into account by incorporating an average loading capacitance during the characterization of the leaf-cells. It is important to note that the contribution due to the execution units is relatively independent of resource utilization (or the degree of time-sharing) since the required number of operations must be performed within the sample period. However, the amount of parallelism will affect the interconnect capacitance.

Registers are treated the same way as the execution units. The number of register accesses per sample period (read/write) is multiplied with the capacitance per register access to yield a register contribution given by  $N_{registers} \cdot C_{register}$ . Once again, while the total number of registers is not important in calculating the register switching capacitance, it will affect floorplanning and therefore the interconnect capacitance.

**4.1.2 Interconnect.** A relatively accurate model for interconnect capacitance is important when performing power trade-offs since often the interconnects starts to dominate over the logic capacitance and restricts improvement in power that can be achieved. Determining the interconnect capacitance is a difficult task since we have to in essence emulate the partitioning, place, and route.

The goal of interconnect estimation is to estimate the inter-block (between macro blocks, such as between datapaths) routing capacitance. A statistical model based approach is used to predict the inter-block capacitance from high level parameters such as the number of global interconnects, active area and bit-width. This model for estimating the average interconnect capacitance switched requires the number of interconnects,  $N$ , the average activity,  $\alpha$ , and the average interconnect length,  $L$ . The average interconnect length is obtained from statistical estimates of the final routed chip area and typical interconnect length distributions as a function of area [11]. The interconnect capacitance is then estimated as:

$$C_{\text{interconnect}} = \alpha * N * L * B * C_L \quad (5)$$

where  $C_L$  is the capacitance per unit length and  $B$  is the bit-width. A more extensive model for the interconnect is currently being developed.

**4.1.3 Control Logic.** From several circuits, it was observed that there is a strong correlation between the control capacitance switched and the total relevant capacitance (muxes, tri-states, registers, and any other module that requires control signals). Based on this information, a simple model is used to predict the total control capacitance as a function of high-level parameters. Notice that control contribution will be a function of the architecture style used.

## 4.2 Supply Voltage Estimation

We are interested in computing the power supply voltage at which the transformed flowgraph will meet the timing constraints. The initial flowgraph which meets the timing constraints is typically assumed to be operating at a supply voltage of 5V with a critical path of  $T_{\text{initial}}$  (the initial voltage will be lower if  $T_{\text{initial}} < T_{\text{sampling}}$ ). After each move, the critical path is re-estimated, and the new supply voltage at which the transformed flowgraph still meets the time constraint,  $T_{\text{sampling}}$ , is determined. For example, if the initial solution requires 10 control steps running at a supply voltage of 5V, then a transformed solution that requires only 5 control steps can run at a supply voltage of 2.9V while meeting the throughput constraint. This relationship (of delay- $V_{dd}$ ) was modelled using Neville's algorithm for rational function interpolation and extrapolation [12].

## 5. Optimization Algorithm

The transformation mechanism is based on two types of moves, global and local. While global moves optimize the whole DCFG simultaneously, local moves involve applying a transformation only on one or very few nodes in the DCFG. The most important advantage of global moves is, of course, a higher optimization effect; the advantages of local moves is their simplicity and small computational cost. We used the following global transformations (i) retiming and pipelining for critical path reduction (ii) associativity (iii) constant elimination and (iv) loop unrolling. In the library of local moves we have implemented three algebraic transformations: associativity, distributivity and commutativity.

The computational complexity analysis of the power minimization problem showed that even highly simplified versions of the optimization tasks are NP-complete. Two widely used alternatives for the design of high quality suboptimal optimization algorithms are probabilistic and heuristic algorithms. Both heuristic and probabilistic algorithms have several distinctive advantages over each other. While the most important advantage of heuristic algorithms is a shorter run time, probabilistic algorithms are more robust and have stronger mechanisms for escaping local minimas.

The algorithm for power minimization using transformations has both heuristic and probabilistic components. While the heuristic part uses global transformations, the probabilistic component uses local moves. The heuristic part applies global transformations one at the time in order to provide good starting points for the application of the probabilistic algorithm. The probabilistic algorithm conducts a probabilistic search in a broad vicinity of the solution provided by the heuristic part. The underlying search mechanism of the probabilistic part is simulated annealing.

## 6. Results

A summary of power improvement after applying transformations relative to an initial solution that met the required throughput constraint at 5V for several representative examples is shown in Table 1. The results indicate that a large reduction in power consumption is possible (at the expense of area) compared to present-day methodologies. Also interesting was the fact that the “optimal” final supply voltage for all the examples was much lower than existing and emerging standards and was around 1.5V.

## 7. Conclusions

The problem of power minimization is becoming a very important problem with the increasing demand for “portable” computing and communication and we have presented a high-level synthesis system, HYPER-LP, for optimizing

Example	Power Reduction	Area Increase
RGB → YUV	8	5
FIR Filter	11	1.1
DCT (8 point)	8	5
Speech Filter	8	6.4
Elliptical Filter	9	2.7
Wavelet Filter	10	2
Volterra Filter	8.6	1

Table 1. Summary of results.

power consumption in application specific datapath intensive circuits using a variety of architectural and computational transformations. The synthesis approach consisted of applying transformation primitives (from a library of local and global moves) in a well defined manner in conjunction with efficient high-level estimation of power consumption. The results indicate that an order of magnitude reduction in power is possible over current-day design methodologies while maintaining the system throughput, and it was found that the optimal supply voltage for minimizing power was much lower than existing standards (present-day 5V and emerging 3.3V) and was around 1.5V for most of the examples investigated. While this work has addressed some key problems in the automated design of low-power systems, there are still many open research problems like detailed power estimation, module selection, partitioning, and scheduling for power optimization.

## Acknowledgments

This project was funded by DARPA. We would like to thank Shan-Hsi Huang for coding and supporting the library of local transformation primitives.

## References

- [1] A. Chandrakasan, S. Sheng, R.W. Brodersen, "Design Considerations for a Future Multimedia Terminal", in Third Generation Wireless Information Network, edited by D. Goodman and S. Nanda, Kluwer Academic Publishers, 1992.
- [2] A. Chandrakasan, S. Sheng, R. Brodersen, "Low-power CMOS Digital Design", IEEE Journal of Solid-state circuit, pp. 473-484, April 1992.
- [3] N. Weste and K. Eshragian, Principles of CMOS VLSI Design: A Systems Perspective, Addison-Wesley, MA, 1988.
- [4] F. Najm, "Transition Density, A Stochastic Measure of Activities in Digital Circuits", DAC, pp. 644-649, 1991.
- [5] A. Ghosh, S. Devadas, K. Keutzer, J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", DAC, pp. 253-259, 1992.

- [6] A. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, "An Approach to Power Minimization Using Transformations", IEEE VLSI Signal Processing Workshop, 1992.
- [7] R. W. Brodersen, (ed.), "Anatomy of a Silicon Compiler", Kluwer Academic Publishers, 1992.
- [8] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-level Simulator", Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 173-178, June 1989.
- [9] M. Potkonjak and J. Rabaey, "Optimizing the Resource Utilization Using Transformations", Proc. IEEE ICCAD Conference, Santa Clara, pp. 88-91, November 1991.
- [10] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Data Path Intensive Architecture". IEEE Design and Test, Vol. 8. No. 2, pp. 40-51, 1991.
- [11] D. Schultz, "The Influence of Hardware Mapping on High-Level Synthesis", M.S. report, U.C. Berkeley, 1992.
- [12] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, "Numerical Recipes in C", Cambridge University Press, 1988.

# POWER ANALYSIS OF EMBEDDED SOFTWARE: FIRST STEP TOWARDS SOFTWARE POWER MINIMIZATION

Vivek Tiwari, Sharad Malik and Andrew Wolfe

*Dept. of Electrical Engineering*

*Princeton University, Princeton, NJ 08544*

## Abstract

Embedded computer systems are characterized by the presence of a dedicated processor and the software that runs on it. Power constraints are increasingly becoming the critical component of the design specification of these systems. At present, however, power analysis tools can only be applied at the lower levels of the design – the circuit or gate level. It is either impractical or impossible to use the lower level tools to estimate the power cost of the software component of the system. This paper describes the first systematic attempt to model this power cost. A power analysis technique is developed that has been applied to two commercial microprocessors – Intel 486DX2 and Fujitsu SPARClite 934. This technique can be employed to evaluate the power cost of embedded software and also be used to search the design space in software power optimization.

## 1. Introduction

Embedded computer systems are characterized by the presence of a dedicated processor which executes application specific software. Recent years have seen a large growth of such systems. This growth is driven by several factors. The first is an increase in the number of applications as illustrated by the numerous examples of “smart electronics” around us. The second factor leading to their growth is the increasing migration from application specific logic to application specific code running on existing processors. The migration to software programmable solutions can often provide the competitive edge in terms of lower manufacturing costs and shorter time to market. Thus, we are seeing a movement from the logic gate being the basic unit of computation on silicon, to an instruction running on an embedded processor.

A large number of embedded computing applications are power critical, i.e., power constraints form an important part of the design specification. While there has been a significant research effort in power estimation and low power design, there is very little available in the form of design tools to help embedded system designers evaluate their designs in terms of the power metric. At present, power measurement tools are available for only the lower levels of the design - at the circuit level and the gate level. At the least these are very slow and impractical to use to evaluate the power consumption of software, and often cannot even

be applied due to lack of availability of circuit and gate level information of the embedded processors. The embedded processors currently used in designs take two possible shapes. The first is “off the shelf” microprocessors or digital signal processors (DSPs). The second is in the form of cores embedded in larger integrated circuits. In the first case, the processor information available to the designer is whatever is made available through data books. In the second case, the designer has logic/timing simulation models to help verify the designs. In neither case is there lower level information available for power analysis.

This paper describes a power analysis technique for embedded software. The goal is to develop and validate an instruction level power model for embedded software. Such a model can then be provided by the processor vendors for both off the shelf processors as well as embedded cores. This can then be used to evaluate embedded software, much as a gate level power model has been used to evaluate logic designs. This is useful in its own right to verify that a design meets its specified power constraints. In addition, it can also be used to search the design space in software power optimization. The technique has so far been applied to two commercial microprocessors – the Intel 486DX2 and the Fujitsu SPARClite 934. This paper uses the former as a basis for illustrating the technique. The application of this technique for the latter is described in a separate reference [4].

## 2. Experimental Method

While it is recognized that the power consumption of a processor varies from program to program, there is a complete lack of models and tools to analyze this variation. Traditional attempts to model the power consumption in the CPU rely on detailed physical layout of the processor and sophisticated power analysis tools that use the information provided by the layout.

In the case of embedded system design, detailed layout information of the CPU is often not available. Even if it is available, these techniques are expensive and difficult to apply. This is also the reason why the potential for power reduction through modification of software is so far unknown and unexploited. The thrust of our work is to overcome these deficiencies by developing a power estimation methodology based on actual laboratory measurements. Given a measurement setup to measure the current being drawn by the microprocessor, the only other information required can be obtained from the widely available manuals and handbooks specific to that microprocessor.

The main idea is to formulate an instruction level power model for the microprocessor. Given this model and an assembly/machine level program, the power consumption in the program can be efficiently estimated. The specifics of the measurement methodology are described next.

## 2.1 Power and Energy

The average power consumed by a microprocessor while running a certain program is given by:  $P = I \times V_{CC}$ , where  $P$  is the average power,  $I$  is the average current and  $V_{CC}$  is the supply voltage. Since power is the rate at which energy is consumed, the energy consumed by a program is given by:  $E = P \times T$  where  $T$  is the execution time of the program. This in turn is given by:  $T = N \times \tau$  where  $N$  is the number of clock cycles taken by the program and  $\tau$  is the clock period.

In common usage, the terms power consumption and energy consumption are often interchanged. However it is important to distinguish between the two when we talk of either of these in the context of programs running on mobile applications. Mobile systems run on the limited energy available in a battery. Therefore the energy consumed by the system or by the software running on it determines the length of the battery life. Energy consumption is thus the focus of attention. We will attempt to maintain a distinction between the two in the rest of the paper. However, in certain cases the term power may be used to refer to energy, in adherence to common usage.

## 2.2 Current Measurement

For this study, the processor used was a 40MHz Intel 486DX2-S Series CPU. The CPU was part of a mobile personal computer evaluation board with 4MB of DRAM memory. The reason for the choice of this processor was that its board setup allowed the measurement of the CPU and DRAM subsystem current in isolation from the rest of the system. *We would like to emphasize that while the numbers we report here are specific to this processor and board, the methodology used by us in developing the model is widely applicable.* The current was measured through a standard off the shelf, dual-slope integrating digital ammeter.

If a program completes execution in a short time, a current reading cannot be obtained visually. To overcome this, the programs being considered were put in infinite loops and current readings were taken. The current consumption in the CPU will vary in time depending on what instructions are being executed. But since the chosen ammeter averages current over a window of time (100ms), if the execution time of the program is much less than the width of this window, a stable reading will be obtained.

The main limitation of this approach is that it will not work for programs with larger execution times since the ammeter may not show a stable reading. However, in this study, the main use of this approach was in determining the current drawn while a particular instruction (instruction sequence) was being executed. A program written with several instances of the targeted instruction (instruction sequence) executing in a loop, has a periodic current waveform which yields a steady reading on the ammeter. This inexpensive approach works very well for

this. However the main concepts described in this paper are independent of the actual method used to measure average current. If sophisticated data acquisition based measurement instruments are available, the measurement method can be based on them.

For our setup,  $V_{CC}$  was 3.3V and  $\tau$  was 25ns, corresponding to the 40MHz internal frequency of the CPU. Given these constants, energy is proportional to the average current and number of cycles. Unless otherwise stated, the numbers reported in this paper correspond to average current in mA.

### 3. Instruction Level Modeling

A modern microprocessor like the 486DX2 is an extremely complex system consisting of several interacting functional blocks. However, this internal complexity is hidden behind a simple interface – its instruction set. Thus to model the energy consumption of this complex system, it seemed intuitive to consider individual instructions. Each instruction involves specific processing across various units of the CPU. This can result in circuit activity that is characteristic of each instruction and can vary with instructions.

This intuition was the starting point for the empirical study that led to the development of the final instruction-level energy model. Under this model each instruction in the instruction set is assigned a fixed energy cost called the *base energy cost*. The variation in base costs of a given instruction due to different operand and address values is then quantified. The base energy cost of a program is based on the sum of the base energy costs of each executed instruction. However, during the execution of a program, certain inter-instruction effects occur whose energy contribution is not accounted for if only base costs are considered. The first type of inter-instruction effect is the effect of circuit state. The second type is related to resource constraints that can lead to stalls and cache misses. The energy cost of these effects is also modeled and used to obtain the total energy cost of a program.

The instruction-level energy model described here is based on actual measurements and evolved as a result of extensive experimentation. It is comprehensive and provides all the information needed to evaluate programs in terms of their energy costs. The various components of this model are described in the subsections below.

#### 3.1 Base Energy Cost

The base cost for an instruction is determined by constructing a loop with several instances of the same instruction. The average current being drawn is then measured. This current multiplied by the number of cycles taken by each instance of the instruction is proportional to the total energy as described in Section 2.

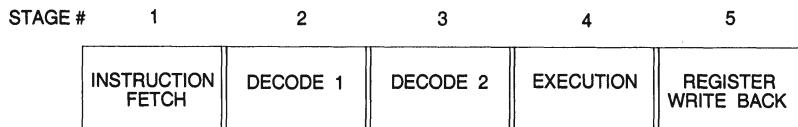


Figure 1. Internal Pipelining in the 486DX2

While this method seems intuitive if the CPU is executing only one instruction at a given time, most modern CPUs, including the 486DX2 are processing more than one instruction at a given time due to pipelining. However, the following discussion shows that the concept of a base energy cost per instruction and its derivation remains unchanged.

Number	Instruction	Base Cost (mA)	Cycles
1	NOP	275.7	1
2	MOV DX,BX	302.4	1
3	MOV DX,[BX]	428.3	1
4	MOV DX,[BX][DI]	409.0	2
5	MOV [BX],DX	521.7	1
6	MOV [BX][DI],DX	451.7	2
7	ADD DX,BX	313.6	1
8	ADD DX,[BX]	400.1	2
9	ADD [BX],DX	415.7	3
10	SAL BX,1	300.8	3
11	SAL BX,CL	306.5	3
12	LEA DX,[BX]	364.4	1
13	LEA DX,[BX][DI]	345.2	2
14	JMP label	373.0	3
15	JZ label	375.7	3
16	JZ label	355.9	1
17	CMP BX,DX	298.2	1
18	CMP [BX],DX	388.0	2

Table 1. Subset of the Base Cost Table for the 486DX2

The 486DX2 CPU has a five-stage pipeline [1] as shown in Figure 1. Let  $E_{jI_k}$  be the average energy consumed by pipeline stage  $j$ , when instruction  $I_k$  executes in that stage. Pipeline stages are separated from each other by latches. Thus, if we ignore the effect of circuit state and resource constraints for now, the energy consumption of different stages is independent of each other. Let us assume that in a given cycle, instruction  $I_1$  is being processed by stage 1,  $I_2$  by stage 2, and so on. The total energy consumed by the CPU in that cycle

data	0	OF	OFF	OFFF	0FFF
No. of 1's	0	4	8	12	16
Base Cost	309.5	305.2	300.1	294.2	288.5

Table 2. Base Costs of MOV BX, data

would be:  $E_{cycle} = E1_{I_1} + E2_{I_2} + E3_{I_3} + E4_{I_4} + E5_{I_5}$ . On the other hand, the total energy consumed by a given instruction  $I_1$ , as it moves through the various stages is:  $E_{ins} = \sum_j E j_{I_1}$ . This quantity actually refers to the base cost in the sense described above. Our method of forming a loop of instances of instruction  $I_1$ , results in  $E_{cycle} = E_{ins}$ , since in that case,  $I_1 = I_2 = I_3 = I_4 = I_5$ . The average current in this case is  $\sum_j E j_{I_1} / (V_{CC} \times \tau)$ , which is the same as the ammeter reading obtained.

Some instructions take multiple cycles in a given pipeline stage. All stages are then stalled. The reasoning applied above, however remains unchanged. The base energy cost of the instruction is just the observed average current value multiplied by the number of cycles taken by the instruction in that stage. For instance, consider a loop of instruction  $I_1$ , where  $I_1$  takes  $m$  cycles in the 4th stage. Therefore,  $E4_{I_1}$  is spread over  $m$  cycles. For sake of brevity assume that each stalled stage consumes zero energy. Then the current value observed on the ammeter will be  $\sum_j E j_{I_1} / (V_{CC} \times \tau \times m)$ . This quantity multiplied by  $m$  yields  $\sum_j E j_{I_1} / (V_{CC} \times \tau)$ , the base energy cost of the instruction.  $m$  represents the “number of cycles” parameter specified in instruction timing tables in microprocessor manuals.

Table 1 is a sample table of CPU base costs for some instructions for the 486DX2.<sup>1</sup> The numbers in Column 3 are the base cost in mA per clock cycle. The overall base energy cost of an instruction is the product of the numbers in Columns 3 and 4 and the constants  $V_{CC}$  and  $\tau$ .

Care should be taken in designing the experiments used to determine the base costs. The size of the loop has to be large enough to minimize the effects of the branch statement at the bottom of the loop and small enough not to cause any cache misses. Only the target instructions should execute on the CPU during the experiment and thus system effects like multiple time-sharing applications and frequent interrupts cannot be allowed.

**3.1.1 Variations in Base Cost.** As Table 1 shows, instructions with differing functionality and different addressing modes can have very different energy costs. This is to be expected since different functional blocks are being affected in different ways by these instructions. Within the same family of instructions, there is variability in base costs depending on the value of operands used. For example, consider the *MOV register,immediate* family. Use of different *registers* results in insignificant variation since the register file is prob-

ably a symmetric structure. Variation in the *immediate* value, however, leads to measurable variation. As an example, Table 2 shows the variation for MOV BX, *immediate*. The costs seem to be almost a linear function of the number of 1's in the binary representation of the immediate data – the more the 1's, the lesser the cost. Similarly, for the ADD instruction, the base costs are a function of the two numbers being added. The range of variation in all cases, however, is small. It is observed to be about 14, which corresponds to less than a 5% variation.

For instructions involving memory operands, there is a variation in the base cost depending upon the address of the operand. The variation is of two kinds. The first is due to operands that are mis-aligned [1]. Mis-aligned accesses lead to cycle penalties and thus energy penalties that are added to the base cost. Within aligned accesses there is variation in the base cost depending upon the value of the address. For example, for MOV DX, [BX], the base cost can be greater than the cost shown in Table 1 by about 3.5%. This variation is a function of the number of, and position of, 1's in the binary representation of the address.

Given the operand value and address, exact base costs can be obtained through direct measurements. However, these exact values will be of little use since typically a data or address value can be known only at run-time. Thus, from the point of view of program energy cost estimation, the only alternative is to use average base cost values. This is reasonable given that the variation in base costs is small and thus the discrepancy between the average and actual values will be limited.

## 3.2 Inter-instruction Effects

When sequences of instructions are considered, certain inter-instruction effects come into play, which are not reflected in the cost computed solely from base costs. These effects are discussed below.

**3.2.1 Effect of Circuit State.** The switching activity in a circuit is a function of the present inputs and the previous state of the circuit. Thus, it can be expected that the actual energy cost of executing an instruction in a program may be different from the instruction's base cost. This is because the previous instruction in the given program and in the program used for base cost determination may be different. For example, consider a loop of the following pair of instructions:

```
XOR BX,1  
ADD AX,DX
```

The base costs of the XOR and ADD instructions are 319.2 and 313.6. The expected base cost of the pair, using the individual base costs would be their average, i.e. 316.4, while the actual current is 323.2. It is greater by 6.8. The

reason is that the base costs are determined while executing the same instruction again and again. Thus each instruction executes in what we expect is a context of least change. At least, that is what the observations consistently seem to indicate. When a pair of two different instructions is considered, the context is one of greater change. The cost of a pair of instructions is always greater than the base cost of the pair and the difference is termed as the *circuit state overhead*.

As another example, consider the following sequence of instructions. The base cost and the number of cycles of each instruction is listed alongside:

Number	Instruction	Base Cost	Cycles
1	MOV CX,1	309.6	1
2	ADD AX,BX	313.6	1
3	ADD DX,8[BX]	400.2	2
4	SAL AX,1	308.3	3
5	SAL BX,CL	306.5	3

The measured cost was 332.8 (avg. current over 10 cycles). Using base costs we get

$$(309.6 + 313.6 + 400.2 \times 2 + 308.3 \times 3 + 306.5 \times 3) / 10 = 326.8 \quad (1)$$

The circuit state overhead is thus 6.0.

It is possible to get a closer estimate if we consider the circuit state overhead between each pair of consecutive instructions. This is done as follows. Consider a loop of the targeted pair, e.g., instructions 2 and 3. The estimated cost for the pair is  $(2 \times 400.2 + 313.6 \times 1) / 3 = 371.3$ , while the measured cost is 374.8. Thus, the circuit state overhead is 3.5. Now the overhead occurs twice in every 3 cycles, once between instructions 2&3, and once between 3&2. Since these two different cases cannot be resolved, let us assume that they are the same. Thus, the overhead each time it occurs would be  $3.5 \times \frac{3}{2} = 5.25$ . Similarly, the overhead between the pairs 1&2, 3&4, 4&5 and 5&1 is found to be 17.9, 12.25, 3.3 and 17.2 respectively. When these overheads are added to the numerator in Equation 1, we get an estimated cost of 332.38, which is within 0.12% of the measured value.

This example illustrates that by determining costs of pairs of instructions, it is possible to improve upon the results of the estimation obtained with base costs alone. However, extensive experiments with pairs of instructions revealed that the circuit state overhead has a limited range - between 5.0 and 30.0 and most frequently occurred in the vicinity of 15.0. This motivates an efficient yet fairly accurate way to account for the circuit state overhead. Calculate the average current for the program using the base costs. Then, add 15.0 to it, to account for circuit state overhead.

A specific manifestation of the effect of circuit state is the effect of switching that occurs on address and data lines. Our experiments revealed that the overall impact of this effect was small. For data reads from the cache, greater switching of the address values led to at most a 2% increase in the energy cost while for data writes (which go to the cache and the memory bus), the overhead due to greater switching was less than 4%.

The limited variation in the circuit state overhead is contrary to popular belief. In fact, a recent work [3], talks about scheduling instructions to reduce this overhead. But as our experiments reveal, the methods described in this work will not have much impact for the 486DX2. The probable explanation for the limited variation in circuit state overhead is that a major part of the circuit activity in a complex processor like the 486DX2, is common to all instructions, e.g., instruction pre-fetch, pipeline control, clocks etc. While the circuit state may cause significant variation within certain modules, its impact on the overall energy cost is swamped by the much greater common cost. However, we would not like to rule out the impact of circuit state overhead for all processors. It may well be that it is a significant part of the energy consumption in DSPs and processors with complex power management features. An investigation of this issue is the subject of our future study.

**3.2.2 Effect of Resource Constraints.** Resource constraints in the CPU can lead to stalls e.g. pipeline stalls and write buffer stalls [1, 2]. These can be considered as another kind of inter-instruction effect. They cause an increase in the number of cycles needed to execute a sequence of instructions. For example, a sequence of 120 MOV DX, [BX] instructions takes about 164 cycles to execute, instead of 120 due to pre-fetch buffer stalls. While determining the base cost of instructions, it is important to avoid stalls, since they represent a condition that ought not to be reflected in the base cost. Thus, for MOV DX, [BX] a sequence consisting of 3 MOV instructions followed by a NOP is used since there are no stalls during its execution [2]. Knowing the cost of the NOP and the measured value for the sequence, the base cost of the MOV is determined.

The energy cost of each kind of stall is experimentally determined through experiments that isolate the particular kind of stall. For example, an average cost of 250 per stall cycle was determined for the prefetch buffer stall.

To account for the energy cost of the above stalls during program cost estimation, the number of stall cycles has to be multiplied by the experimentally determined stall energy cost. This product is then added to the base cost of the program. The number of stall cycles is estimated through a traversal of the program code.

**3.2.3 Effect of Cache Misses.** Another inter-instruction effect is the effect of cache misses. The instruction timings listed in manuals give the

cycle count assuming a cache hit. For a cache miss, a certain cycle penalty has to be added to the instruction execution time. Along the same lines, the base costs for instructions with memory operands are determined in the context of cache hits. A cache miss will lead to extra cycles being consumed, which leads to an energy penalty. For experimentation purposes, a cache miss scenario is created by accessing memory addresses in an appropriate order. An average energy penalty of 216 per miss cycle has been experimentally obtained. This has to be multiplied by the average number of miss penalty cycles to get the average energy penalty for one miss. The average penalty multiplied by the cache miss rate is added to the base cost estimate to account for the cache misses during execution of a program.

#### 4. Estimation Framework

Program	Base Cost(mA)	Cycles
; Block B1		
main:		
mov bp,sp	285.0	1
sub sp,4	309.0	1
mov dx,0	309.8	1
mov word ptr -4[bp],0	404.8	2
;Block B2		
L2:		
mov si,word ptr -4[bp]	433.4	1
add si,si	309.0	1
add si,si	309.0	1
mov bx,dx	285.0	1
mov cx,word ptr _a[si]	433.4	1
add bx,cx	309.0	1
mov si,word ptr _b[si]	433.4	1
add bx,si	309.0	1
mov dx,bx	285.0	1
mov di,word ptr -4[bp]	433.4	1
inc di, 1	297.0	1
mov word ptr -4[bp],di	560.1	1
cmp di,4	313.1	1
jl L2	405.7(356.9)	3(1)
;Block B3		
L1:		
mov word ptr _sum,dx	521.7	1
mov sp,bp	285.0	1
jmp main	403.8	3

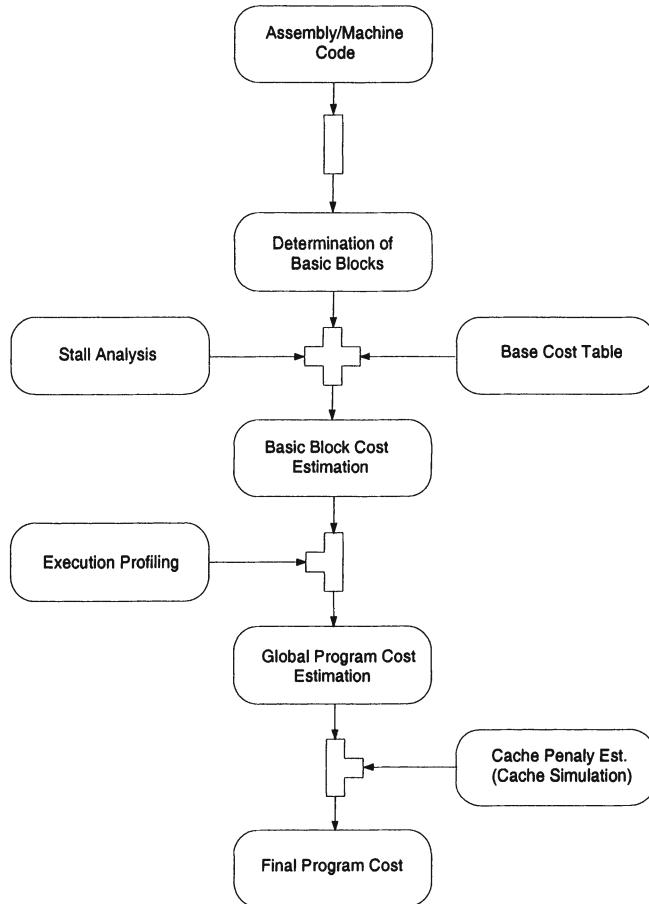
Table 3. Illustration of the Estimation Process

In this section we describe a framework for energy estimation of programs using the instruction level power model outlined in the previous section. We start by illustrating this estimation process for the program shown in Table 3. The program has three basic blocks as shown in the table.<sup>2</sup> The average current per cycle and the number of cycles for each instruction are given along with each instruction. With these numbers the base cost of the basic block B1 is 1713.4, B2 is 4709.8, and B3 is 2017.9. B1 is executed once, B2 4 times and B3 once. The `jmp main` statement has been inserted to put the program in an infinite loop. Cost of the `j1 L2` statement is not included in the cost of B2 since its cost is different depending on whether the jump is taken or not. It is taken 3 times and not taken once. Multiplying each basic block by the number of times it is executed and adding the cost of the unconditional jump `j1 L2`, we get a number proportional to the total energy cost of the program. Dividing it by the estimated number of cycles (72) gives us the cost per cycle (average current) of 369.1. Adding the circuit state overhead offset value of 15.0 we get 384.0. The actual measured average current is 385.0. This program does not have any stalls and thus no further additions to the estimated cost are required. If in the real execution of this program, some cold-start misses are expected, their energy overhead will have to be added.

To validate the estimation model described in the previous section, experiments were conducted with several programs. A close correspondence between the estimated and measured cost was obtained. The estimated cost was typically within 3% of the measured cost.

## 4.1 Overall Flow

The overall flow of the estimation procedure is shown in Figure 2. Given an assembly or machine level program, it is first split up into basic blocks. The base cost of each instance of the basic block is determined by adding up the base costs of the instructions in the block. These costs are provided in a base cost table. The energy overhead due to pipeline, write buffer and other stalls is estimated for each basic block and added to the basic block cost. Next, the number of times each basic block is executed has to be determined. This depends on the path that the program follows and is dynamic, run-time information that is obtained from a program profiler. Given this information, each basic block is multiplied by the number of times it will be executed. The circuit-state overhead is added to the overall sum at this stage, or alternatively, it could have been determined for each basic block using a table of energy costs for pairs of instructions. An estimated cache penalty is added to get the final estimate. The cache penalty overhead computation needs an estimate of the miss ratio, which is obtained through a cache simulator.



*Figure 2.* Software Energy Consumption Estimation Methodology

## 5. Memory System Modeling

The energy consumption in the memory system is also a function of the software being executed. The salient observations regarding the DRAM system current on our experimental setup are briefly described here. Details are provided in a separate reference [6].

The DRAM system draws constant current when no memory access is taking place. This current value was determined to be  $77.0mA$  or  $5.3mA$ , depending on whether page mode was active or not. Greater current is drawn during a memory access. The exact value of this current depends on the address of the present and previous memory access. For example, for writes, the cost of a page hit is 122.8 (for 3 cycles) and that of a miss is 247.8 (for 6 cycles). For page hits, a smaller

variation was observed depending on the number of bits that change from the previous address to the present.

Let  $X$  be the sum of the energy costs of each individual memory access. Let  $n$  and  $m$  be the number of memory idle cycles during which the page mode is active and inactive, respectively. The total memory system energy cost is given by  $X + 77.0 \times n + 5.3 \times m$ . As discussed above, the quantity  $X$  depends on the location and sequence of memory accesses made by the program. Along with  $n$  and  $m$ , this is dynamic, run-time information, which can only be loosely estimated by static analysis. Thus, modeling of memory system energy consumption is difficult if only static analysis is used. However, as the above discussion shows, analysis of this consumption is feasible. This is significant, given that for systems with tight energy budgets, it is important to understand all sources of energy consumption.

## 6. Software Power Optimization

In recent years, there has been a spurt of research activity targeted at reducing the energy consumption in systems. This research, however, has by and large not recognized the potential energy savings achievable through optimization of software. This was mainly due to the lack of practical techniques for analyzing the energy consumption of programs. This deficiency has been alleviated by the measurement and estimation methodology described in the previous sections. This methodology makes it possible to compare and evaluate programs in terms of their energy consumption and also to study the effect of compilation on the energy consumption of programs.

Using the results of this work, several possible avenues for energy reduction through code restructuring and compilation have been studied [5]. Examples with energy reduction of up to 40% on the 486DX2 based system, obtained by rewriting code, demonstrate the potential of these ideas. These ideas will be pursued further as part of the research in the area of software power optimization.

## 7. Analysis of SPARClite 934

The previous sections describe the application of the power analysis methodology for the 486DX2, a CISC processor. To verify the general applicability of this methodology, it was decided to apply the methodology to a processor with a different architectural style. The Fujitsu SPARClite 934, a RISC processor targeted for embedded applications was chosen for this purpose. A power analysis of this processor has been performed using the measurement and experimentation techniques described in the previous sections. The basic model of a base energy cost per instruction, enhanced by the inter-instruction effects remains valid for this processor, though the actual costs differ in value. The details of this analysis are described in a separate reference [4].

## 8. Summary and Future Work

This paper presents a methodology for analyzing the energy consumption of embedded software. It is based on an instruction level model that quantifies the energy cost of individual instructions and of the various inter-instruction effects. The motivation for the analysis methodology is three-fold. It provides insights into the energy consumption in processors. It can be used to help verify if an embedded design meets its energy constraints and it can also be used to guide the design of embedded software such that it meets these constraints.

The methodology has so far been applied to two commercial processors, a CISC and a RISC. Future work will extend this to other architecture styles, to characterize and contrast their energy consumption models. DSPs, superscalar processors and processors with internal power management will be considered.

## Acknowledgments

We would like to thank Deo Singh, Suresh Rajgopal and Tom Rossi of Intel Corp. for providing us with the 486DX2 evaluation board, Mike Tien-Chien Lee, Masahiro Fujita and Dinesh Maheshwari of Fujitsu for helping make the SPARClite analysis possible and Dan Markham and Pete Derosa of Princeton University for assistance in collecting the experimental data.

## Notes

1. All instructions are executed in "Real Mode". All registers contain 0, except in entry 11, where CL contains 1. Entry 15 is a "taken" jump while entry 16 is "fall through". Entries 5, 6 and 9 show *normalized* costs [6].

2. A basic block is defined as a contiguous section of code with exactly one entry and exit point.

## References

- [1] Intel Corporation. *i486 Microprocessor, Hardware Reference Manual*, 1990.
- [2] Intel Corporation. *i486 Microprocessor Family, Programmer's Reference Manual*, 1992.
- [3] Su, C.L., Tsui, C.Y., and Despain, A.M. "Low Power Architecture Design and Compilation Techniques for High-Performance Processors." In *Proceedings of the IEEE COMPCON*, February 1994.
- [4] Tiwari, V., and Lee, T.C. "Power Analysis of a 32-bit Embedded Microcontroller." *VLSI Design*, Vol. 7, No. 3, 1998.
- [5] Tiwari, V., Malik, S., Wolfe, A., and Lee, T.C. "Instruction Level Power Analysis and Optimization of Software." *Journal of VLSI Signal Processing Systems*, Vol. 13, No. 2, August 1996.
- [6] Tiwari, V. *Logic and System Design for Low Power Consumption*. Doctoral Thesis, Dept. of Electrical Engineering, Princeton University, September 1996.

# A METHODOLOGY FOR CORRECT-BY-CONSTRUCTION LATENCY INSENSITIVE DESIGN

Luca P. Carloni

*University of California at Berkeley  
Berkeley, CA 94720-1772*

Kenneth L. McMillan and Alexander Saldanha<sup>1</sup>

*Cadence Berkeley Laboratories  
Berkeley, CA 94704-1103*

Alberto L. Sangiovanni-Vincentelli

*University of California at Berkeley  
Berkeley, CA 94720-1772*

## Abstract

In Deep Sub-Micron (DSM) designs, performance will depend critically on the latency of long wires. We propose a new synthesis methodology for synchronous systems that makes the design functionally insensitive to the latency of long wires. Given a synchronous specification of a design, we generate a functionally equivalent synchronous implementation that can tolerate arbitrary communication latency between latches. By using latches we can break a long wire in short segments which can be traversed while meeting a single clock cycle constraint. The overall goal is to obtain a design that is robust with respect to delays of long wires, in a shorter time by reducing the multiple iterations between logical and physical design, and with performance that is optimized with respect to the speed of the single components of the design. In this paper we describe the details of the proposed methodology as well as report on the latency insensitive design of *PDLX*, an out-of-order microprocessor with speculative-execution.

## 1. Introduction

The advent of deep sub-micron (DSM) process technologies,  $0.13\mu$  and below, has generated a flurry of predictions on the effects of the inevitable dominance of wire delays on chip design. Although there is a certain amount of disagreement between the various studies on interconnect latencies in future design generations [9, 10], there is unanimity that the delay of a “long” wire will play a dominant role in logic synthesis and optimization. Recent ad-

---

<sup>1</sup>Author is currently with Softface, Inc.

vances on interconnect optimization techniques (such as interconnect topology optimization, optimal buffer insertion and sizing, optimal wire-sizing) can help to reduce interconnect delays significantly [8], but they are not able to reverse the trend of growing gap between device and interconnect performance [7]. In the current standard-cell design methodology, logic synthesis is performed using delay estimates for library modules that are parameterized to account for loading factors and transition (or slew) rates. As the delay of long wires become larger relative to gate delays, these estimates become increasingly sensitive to layout. Attempts have already been made to account for layout effects by performing floor-planning and wire-planning on register-transfer level (RTL) descriptions [25]. Such an approach requires extreme precaution in deriving constraints for synthesis tools, since any wire whose delay approaches a single clock may cause a failure to meet the timing constraints.

In this paper, we propose an alternative synthesis methodology that produces designs functionally insensitive to the latency of long wires. Given a synchronous design consisting of several communicating modules, automatic synthesis techniques are used to generate a functionally equivalent synchronous implementation that can tolerate arbitrary communication latency between modules. The overall goal is to achieve a robust design implementation that has as high a throughput as possible. As a preliminary assumption, each module must satisfy the *stallability* property, meaning that it can be stalled for an arbitrary amount of clock cycles without losing its internal state. In our implementation, the modules of the design communicate over channels, using a standard protocol that is insensitive to latency. This protocol allows a channel to run a number of clock cycles ahead of or behind other channels. The resulting system is guaranteed *by construction* to be functionally equivalent to it. The system maintains the appearance of a fully synchronous system despite the non uniform latencies along communication channels of the actual implementation.

The methodology is presented in Section 2 and discussed with respect to previous work in Section 3. In Section 4, we summarize the theory of latency insensitive protocols. In Section 5, we address some issues related to latency insensitive protocol implementation. In Section 6 we report on performance evaluation of the latency-insensitive design methodology for a fairly complex prototype system.

## 2. The Methodology

The proposed methodology is based on the automatic synthesis of a *communication architecture* implementing a *latency insensitive communication protocol*. It consists in a succession of five basic steps:

- 1 The designer starts with a completely synchronous specification of the system and with a collection of *modules*, which can be either acquired as

*intellectual property (IP) cores* from a (internal or external) third-party or can be specified as “synthesizable” code using a hardware description language such as VERILOG or VHDL.

- 2 Communicating modules are connected by means of *channels* as illustrated in Figure 1. Each channel operates using a latency-insensitive communication protocol and is made up of wires and logic blocks called *relay stations*. The wires of a channel are laid out together and share physical characteristics. The relay stations consist of latches together with logic gates implementing the functionality related to the latency-insensitive communication protocol.
- 3 Each module is encapsulated within a logic block called *shell*, playing the role of interface towards the communication architecture.
- 4 The layout is obtained using standard *place & route* tools.
- 5 A post-layout optimization step is performed to insert the necessary number of relay stations into each “critical channel” to ensure that the cycle time is met (*channel segmentation*). Some iterations may be required, but they are limited to each channel separately, while logic and layout of all modules remain untouched.

The essential point in this methodology is the *orthogonalization of concerns* between behavior and communication. Since the communication mechanism is automatically synthesized (as described later in this paper both relay stations and shells can be built with no intervention of the designer based only on the theory of latency insensitive protocols), the designer can focus on the choice of the modules that make up the functionality of the implementation without worrying about synchronization and latency of the overall design.

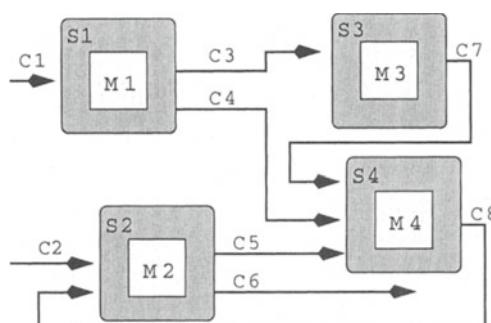


Figure 1. Shell Encapsulation and Communication Channels.

Communication design does not have any impact on the design and implementation of the modules provided that the modules and the relay stations share a fundamental property, *patience* (see Section 4). Requiring that an arbitrary module is patient at the onset is quite strong. This is the reason why we encapsulate the modules with an appropriate shell that has the task of making the module look patient. Such shells can be automatically generated for all modules if the output of the module is latched and each module is *stallable* [4]. “*Stallability*” means that a module can stall for an arbitrary amount of clock cycles without losing its internal state (and the overall state of the system) and is much weaker than *patience*<sup>1</sup>.

### 3. Related Work

The adoption of DSM process technologies and the increasing impact of interconnect delay are destined to exacerbate the *timing-closure problem*: the designer is forced to iterate many times between synthesis and layout, because the two steps are performed independently and synthesis uses statistical delay models which badly estimate the post-layout wire load capacitance [7, 19].

In [10] Sylvester and Keutzer discuss the impact of DSM geometries on the future of design automation methodologies and envision that future integrated circuits will be implemented hierarchically with large macro-blocks of approximately 50K to 100K gates. They conclude that traditional standard-cell design flow will be still used for the design of such macro-blocks, because “*interconnect delay will be small ( $\leq 25\%$ ) in block of 50K gates*”. These results are obtained from the analysis of detailed ASIC design data, such as average wire-lengths and average net fan-out. However, one must observe that the timing closure problem arises when the delay of the critical path in the design is excessive, and, therefore, it is by nature a worst-case problem and not an average-case problem. Most of the solutions proposed in literature so far call for tighter interaction between synthesis and physical design. A synthesis-driven methodology that optimizes for interconnect delay rather than gate delay during logic synthesis is presented in [14]. Unfortunately, the approach produces a large amount of logic duplication, which may lead to expensive area overheads. Floorplanning, technology mapping and gate placement are combined in [27], where, after placement has been completed, the critical paths are reduced one at a time to meet the timing requirements. Since to fix one critical path may generate new ones, this approach is unable to solve *by construction* the convergence problem. A series of layout-driven approaches suggest to fix the layout by extracting accurate physical informations which are used to guide different types of logic optimization, such as gate-resizing [16], fanout optimization [18], buffer insertion [28], and logic resynthesis [22].

All these approaches represent remedies to the effects of bad estimations made during logic synthesis and do not seem able to scale well with the shrinking of process geometries. Following the old adage that *an ounce of prevention is worth a pound of cure*, we believe that the time for a radical paradigm shift is approaching.

### 3.1 Latency Insensitive vs. Asynchronous Design

The latency insensitive design methodology is clearly reminiscent of many ideas which have been proposed in the asynchronous design community during the past three decades [11]. In particular, the idea of a design methodology which is inherently modular is already present in the work on *Macro-modular Computer Systems* by Clark and Molnar [5, 6]. To separate the design of these modules by the design of the system and make the entire process amenable to automation, the modules must be implemented as *delay-insensitive* circuits [24, 26]. A delay-insensitive circuit is designed to operate correctly regardless of the delays on its gates and wires (unbounded delay model) [32]. However, it has been proven that almost no useful delay-insensitive circuits can be built if one is restricted to a class of simple logic gates [2, 23]. To be able to build complex systems one must use more complex components, which are “externally” delay insensitive, while “internally” are designed by carefully verifying their timing and avoiding or tolerating metastability [13, 17, 26]. By slightly relaxing the unbounded delay model and allowing “isochronic forks”<sup>2</sup>, practical *quasi-delay-insensitive* circuits can be built using simple logic gates [3]. A further relaxation leads to *speed independent* circuits, which operate correctly regardless of gate delays, while wire delays are assumed to be negligible [1, 12, 20]. Both quasi-delay-insensitive and speed-independent circuits assume that the designer is able to control wire delays, and, therefore, do not appear as interesting alternatives when moving to DSM implementations. Instead, a methodology based on assembling complex modules which are “externally” delay-insensitive seems the right solution, on condition that the synthesis of such modules is not too cumbersome. However, it must be noted that asynchronous approaches do not address the fundamental problem of latency, because an asynchronous design simply slows down to accommodate the slowest component, e.g. the wires.

While a delay insensitive system is based on the assumption that the delay between two subsequent events on a communication channel is completely arbitrary, in the case of a latency insensitive system this arbitrary delay is a *multiple of the clock period*. The key point is that this kind of *discretization* allows us to leverage well-accepted design methodologies for the design and validation of synchronous circuits. In fact, the basic distinction between any of the previous asynchronous design methodologies and the latency-insensitive one is es-

sentially that a latency insensitive system is specified as a synchronous system. Notice that we say “specified” because, from an implementation point of view a latency-insensitive communication protocol can also be realized using *hand-shaking signaling* techniques (such as request/acknowledge protocols), which are typically asynchronous<sup>3</sup>. However, from a specification point of view, each module (as well as the overall system) is viewed as a synchronous system. Now, to specify a complex system as a collection of modules whose state is updated collectively in one “zero-time” step is naturally simpler than specifying the same system as the interaction of many components whose state is updated following an intricate set of interdependency relations. Furthermore, the synchronous specification allows us to slightly modify the traditional semi-custom design methodology, by simply inserting a step to encapsulate each synchronous module within a shell. Finally, the impact is very different also from a validation point of view because simulation is naturally a less complex task for a synchronous circuit than an equivalent asynchronous one. In conclusion, the proposed methodology can be implemented on top of the commonly-adopted standard-cell design flow, while all previous asynchronous approaches force the designer to use new tools and, more importantly, *to think the digital system in a completely different way*.

#### 4. Latency Insensitive Protocols

The proposed design methodology is based on the theory of *latency insensitive protocols*, which has been recently presented in literature [4]. This theory can be summarized as follows. A latency insensitive protocol is a communication protocol governing the exchange of information in a patient system. According to the Tagged-Signal Model [21] a system is a composition of processes communicating by exchanging signals, i.e. sequences of events, on a set of channels. A behavior of a system is unambiguously described by the set of signals which are exchanged among its processes. A *patient* system is a synchronous system whose functionality only depends on the order of the events of each signals and not on their exact timing. More specifically, a patient system is a collection of patient processes communicating by means of “point-to-point” channels whose latency may be arbitrary. Normally, at every cycle  $t_k$ , a generic patient process  $P_i$  receives a new *informative event* on each of its input channels and it emits informative events, which are the result of its internal computation up to the previous cycle  $t_{k-1}$ , on its output channels. However, due to channel arbitrary latencies, it may happen that at cycle  $t_k$  a *stalling event* (denoting the absence of an informative event) arrives on one or more of its input channels. If this is the case, process  $P_i$  (being *patient*) waits an arbitrary but finite amount of extra cycles until all informative events (which were expected at  $t_k$ ) have arrived on all input channels. During this wait,  $P_i$  emits stalling events. Any sequence

of stalling cycles does not affect the internal state of  $P_i$  (the process is patient) as well as the overall state of the system (the protocol guarantees that all processes awaiting data from  $P_i$  receive instead a stalling event).

If all channels in the system have unit latency then no stalling events are exchanged among its processes. Let  $S_{ref}$  be a patient system with such a characteristic. Then, let  $S_{stall}$  be another patient system which is composed by exactly the same processes as  $S_{ref}$ , while having some channels with latency greater than one clock cycle. Now, assume to apply to the two systems the same external stimulus yielding two corresponding behaviors  $\beta_{ref}$  and  $\beta_{stall}$ . If all stalling events are filtered away from  $\beta_{stall}$ , the resulting behavior is exactly equal to  $\beta_{ref}$ . The two behaviors are said *latency equivalent*. Further, if every behavior of  $S_{ref}$  is latency equivalent to some behavior of  $S_{stall}$  (and *vice versa*) then the two processes are said to be latency equivalent. It has been proven that, for patient processes, latency equivalence is compositional [4].

A *relay station* is a patient process communicating with two channels  $c_i$  and  $c_o$  such that if  $s_i$  and  $s_o$  are the signals associated to the channels and  $I(l, k, s_i), l \leq k$  denotes the sequence of informative events of  $s_i$  between the  $l$ -th clock cycle and the  $k$ -th one, then  $s_i$  and  $s_o$  are latency equivalent and for all  $k$

$$I(1, (k-1), s_i) - I(1, k, s_o) \geq 0 \quad (1)$$

$$I(1, k, s_i) - I(1, (k-1), s_o) \leq 2 \quad (2)$$

The following is an example of relay station behavior, where  $\tau$  denotes a stalling events and  $\iota_i$  a generic informative event:

$$\begin{aligned} s_i &= \iota_1 \iota_2 \iota_3 \tau \tau \iota_4 \iota_5 \iota_6 \tau \tau \iota_7 \tau \iota_8 \iota_9 \iota_{10} \dots \\ s_o &= \tau \iota_1 \iota_2 \iota_3 \tau \tau \iota_4 \tau \tau \iota_5 \iota_6 \iota_7 \tau \iota_8 \iota_9 \iota_{10} \dots \end{aligned}$$

Notice, that no further specification has been given on the signals  $s_i$  and  $s_o$ , (for instance saying that  $s_i$  is the input and  $s_o$  is the output). The definition of relay station simply involves a set of relations, i.e. a protocol, between  $s_i$  and  $s_o$  without any implementation detail. Still, it is clear that each informative event received on channel  $c_i$  is later emitted on  $c_o$ , while the presence of a stalling event on  $c_o$  may induce a stalling event on  $c_i$  in a later cycle. In fact, an informative event takes at least one clock cycle to pass through a relay station (minimum forward latency = 1), at most two informative events can arrive on  $c_i$  while no informative events are emitted on  $c_o$  (internal storage capacity = 2), and, finally, one extra stalling event on  $c_o$  will “move” into  $c_i$  in at least one cycle (minimum backward latency = 1). The double storage capacity of a relay station permits, in the best case, to communicate with maximum throughput (equal to one): a practical confirmation of this fact is given in Section 5, where an RTL implementation of a relay station is discussed.

Since relay stations are patient processes, their insertion in a patient system guarantees that the system remains patient. Further, since they have minimum latencies equal to one, they can be repetitively inserted on a channel to increase its latency. Therefore, the methodology is patterned after the theory as follows: (1) we start giving an abstract specification of a digital system as collection of synchronous modules without making any assumption on the latency of the wires (which are grouped in channels), then (2) we automatically synthesize a corresponding layout, (3) we segment every wire whose latency is greater than the desired clock period by distributing on it the necessary amount of relay stations, and (4) we build the shell around the modules to obtain patient processes that interact with the appropriate relay stations. Obviously, the final result will be satisfactory only to the extent that a sufficient throughput can be maintained in the presence of increased latency of wires. However, this is a general problem that will have to be faced in the design of large chips with DSM technologies, and not specific to the latency insensitive methodology. On the other hand, the latency insensitive methodology allows an easy early exploration of latency/throughput tradeoffs as illustrated in Section 6.

## 5. The Implementation of the Protocol

In this Section, we present a latency insensitive communication architecture consisting of channels, relay stations, and shells built according to our methodology.

### 5.1 Channels

Channels are point-to-point unidirectional links between a *source* module and a *sink* module. Data are transmitted on a channel by means of *packets*: a packet consists of a variable number of fields. Here, we consider only two basic fields: *payload* contains the transmitted data and *void* is a one bit flag which, if set to 1, denotes that no data are present in the packet (*void packet*). If a packet does contain “meaningful” payload data (i.e., void is set to 0) we will call it a *true packet*. A channel is made of wires and relay stations. The number of relay stations in a channel is finite and represents the buffering capability of the channel.

At each clock cycle, the source module may either put a new true packet on the channel or, in case no output data are available to be sent, put a void packet on it; on the other side, at each clock cycle the sink module retrieves from the channel the incoming packet and, on the basis of the void field value, decides whether to discard it or to store it on its input channel queue for later use. As a source module might not be ready to send a true packet, so a sink module might not be ready to receive it, for instance because its input queue is full. However, the latency insensitive protocol demands a fully reliable communication among

the modules, where no lossy communication link is allowed and all packets are properly delivered. Consequently, the sink module must have a way to interact with the channel (and ultimately with the corresponding source module) to stop momentarily the communication flow and avoid the loss of any packet. Therefore, we slightly relax our definition of a channel as unidirectional, to allow a bit of information, called the channel *stop flag*, moving in the opposite direction. By setting the stop flag equal to one during a certain clock cycle, the sink module informs the channel that the next packet can not be received and it must be held until the stop flag is reset. As the sink module also the channel has a limited amount of buffering resources: a channel dealing with a sink module that requires a long stall period may fill up all its relay stations and being forced to send a stop flag to the source module so that the latter will put its packet production on stall.

## 5.2 Relay Stations

Figure 2 illustrates a possible relay station implementation based on the following specification, which refines the abstract notion given in Section 4:

“At each clock cycle  $t$  it takes a packet  $packetIn^{t+1}$  and a stop signal  $stopIn^{t+1}$  as inputs and it emits a packet  $packetOut^{t+1}$  and a stop signal  $stopOut^{t+1}$  as outputs:  $stopOut^{t+1}$  is always equal to  $stopIn^t$ , while, according to the value of the internal variable  $stalling^t = stopIn^t \wedge stopIn^{t-1}$  the relay station decides whether to set  $packetOut^{t+1}$  equal to  $packetIn^t$  (case:  $stalling^t = 0$ ) or to stall by keeping  $packetOut^{t+1}$  equal to  $packetOut^t$  and saving  $packetIn^t$  value into an auxiliary register (case:  $stalling^t = 1$ ”).

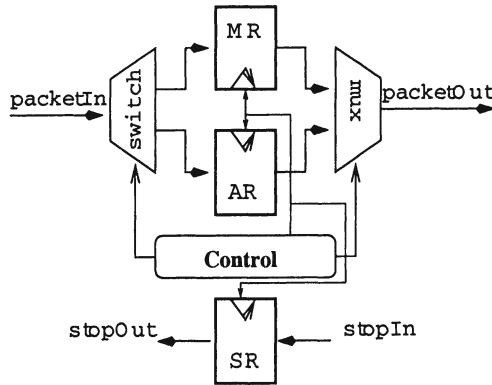


Figure 2. Relay Station Implementation.

Figure 3 illustrates two modules, *Fetch Unit* and *Instruction Cache*, communicating using two channels *Address Channel* and *Data Channel*. Both channels have been partitioned in 4 segments by the insertion of 3 relay stations and, as

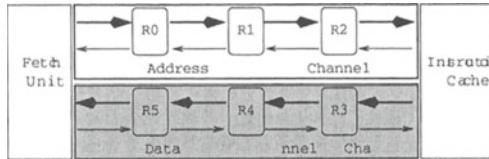


Figure 3. Channels between *Fetch Unit* and *Instruction Cache*.

a consequence, the lower bound on the latency of each channel has become 4 clock cycles. Figure 4 reports a snapshot of the waveforms obtained by simulating a VERILOG RTL description of the *Address Channel*: here, the source module is the *Fetch Unit* producing a sequence of addresses for a *Memory Block* which represents the sink module. The addresses are reported as hexadecimal numbers.

Beside the system clock having period  $T_{CLK}$  equal to 10ns, one can see 8 waveforms which, going from top to bottom, correspond respectively to the following signals of Figure 3:  $R2.packetOut$ ,  $R2.stopIn$ ,  $R1.packetOut$ ,  $R1.stopIn$ ,  $R0.packetOut$ ,  $R0.stopIn$ ,  $FU.packetOut$ ,  $FU.stopIn$ .

At time  $t = 75\text{ns}$  the sink module sets  $R2.stopIn$  equal to one and keeps it equal to one for three clock cycles. As a consequence,  $R2$  stalls two cycles as it maintains  $R2.packetOut = h'44$  for the next three cycles while storing  $R1.packetOut = h'45$  on a auxiliary set of registers. In the meantime, the stop signal is propagated to  $R1.stopIn$ . When, after three clock cycles, at time  $t = 105\text{ns}$ , the sink module can finally receive  $R2.packetOut = h'44$ , it resets  $R2.stopIn$  such that at the following clock cycle  $R2$  may set  $R2.packetOut = h'45$ . In the meantime, the three consecutive high values of the stop signal propagate back through the channel, provoking a stall of two cycles for each station while guaranteeing that no packets are lost. Notice that a characteristic of this implementation of the protocol is that when a *stopIn* signal is kept high for only one cycle, the relay station does not really stall: in Figure 4 this can be observed for the sequence of clock cycles starting at  $t = 165\text{ns}$ . This fact is simply a positive bi-product of the fact that the storing capacity of a relay station is double<sup>4</sup>.

### 5.3 Shells

As introduced in Section 2, given a particular module  $M$ , an instance of a shell can be automatically synthesized as a wrapper to encapsulate  $M$  and interface it with the channels so that  $M$  becomes a patient process. To do so the only necessary condition is that  $M$  be stallable.

At each clock cycle the module internal computation must be *fired* only if all inputs have arrived. Guaranteeing this *input synchronization* is the first task of the shell of a module. The second task is called *output propagation*: at each

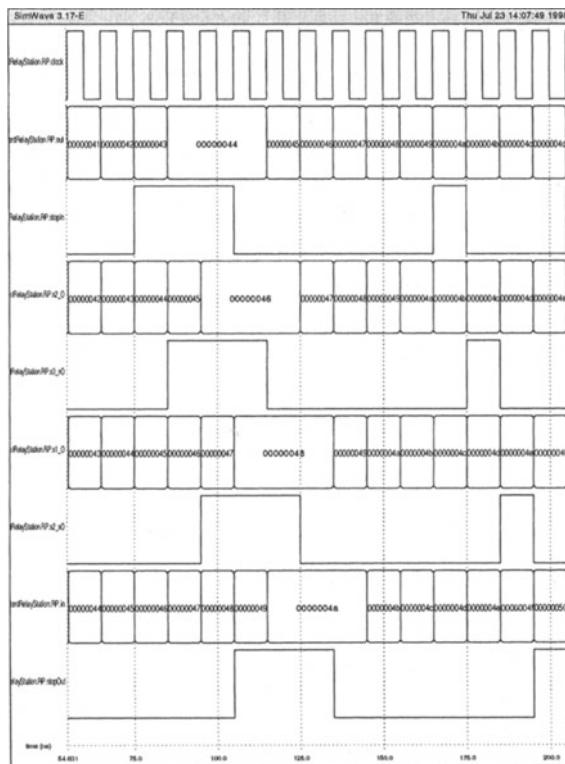


Figure 4. Waveforms on Address Channel.

clock cycle, if module  $M$  has produced new output values and no output channel has previously raised a stop flag, then these output values can be transmitted generating new true packets; if any of these two conditions is not verified, then the packet transmitted in the previous cycle is re-transmitted as a void packet.

In summary a shell for module  $M$  performs the following actions cyclically:

- 1 it gets the incoming packets from the input channels, filters away the void packets and extracts the input values for  $M$  from the payload fields of the true packets;
- 2 when all input values are available for the next computation, it passes them to  $M$  and fires the computation;
- 3 it gets the results of the computation from  $M$ ;
- 4 if no output channel has previously raised a stop flag, it routes the result into the output channels.

## 6. Case Study: The PDLX Microprocessor

To test our methodology, we performed a “latency insensitive design” of an out-of-order microprocessor (PDLX) with speculative execution. Its instruction set is the same of the DLX microprocessor, described in [15]. Its architecture is based on an extended version of the *Tomasulo's Algorithm* [31], which combines traditional dynamic scheduling with hardware-based speculative execution. The data-path of PDLX is similar to the one of some of the most advanced microprocessor available on the market today.

Figure 5 illustrates a simplified block diagram of the PDLX architecture: the *PC Unit* sends the current value of the *Program Counter (PC)* to the *Instruction Cache* and the *Fetch Unit*. After receiving the corresponding instruction, the *Fetch Unit* couples it with the PC value and sends it to the *Decode Unit*. Once instruction decoding is completed, the result arrives to the *Execution Unit* which performs the execution phase working with the *Data Cache* and the *Register File*. If the result of the execution is a “*branch taken*”, then the branch target address is sent to the *PC Unit*.

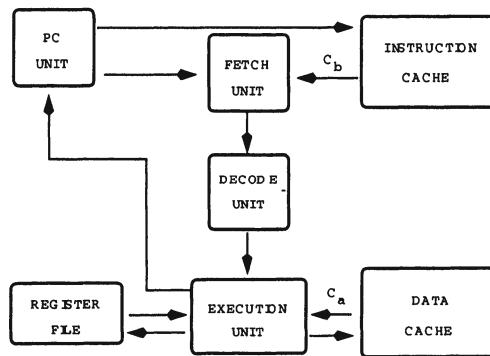


Figure 5. PDLX Microprocessor Block Diagram : top level view.

In our implementation, the 7 units correspond to 7 modules made patient by adding an appropriate shell. Obviously, this decomposition of the hardware implementing the PDLX, is not the only possible, let alone the best one. Still, while reasonably simple, it presents interesting challenges to the realization of the proposed latency insensitive communication architecture. In particular, the *Fetch Unit* shell merges two separate channels (likely they have different latencies), and each time a “*branch taken*” is executed a “*feedback path*” is activated between the *Execution Unit* and the *PC Unit*.

We performed a high-level cycle-accurate design of PDLX by using BONeS DESIGNER [29]. We first designed the PDLX modules illustrated in Figure 5, keeping in mind only the following informal rule to make the process stallable:

At each clock cycle the execution process of a module can always be frozen without affecting its internal state. Then, we designed the latency insensitive protocol library, containing as building blocks relay stations and shells. Finally, we encapsulated each module in a shell and we obtained the final system. To test our design, we took some simple numerical C programs (permutations, binary search,...) and we generated the corresponding DLX assembler code by using DLXCC, a publicly available DLX compiler [30]. Then, we loaded the assembler into the PDLX *Instruction Cache* and we executed it, while logging every read/write access to the *Data Cache*. Finally, we compared the “log file” with the one obtained executing the same assembler code on the DLX simulator DLXSIM to verify that the functional behavior was indeed the same.

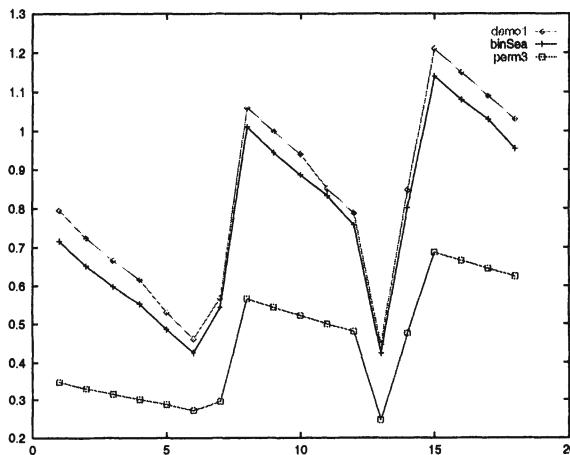


Figure 6. Effective Throughput vs. PDLX implementations.

For each program execution, we computed the total number of clock cycles  $T$  necessary to complete the execution of the assembler code: this number is equal to  $I + S + P$ , where  $I$  is the number of instruction which have been committed,  $S$  is the number of cycles lost due to a stall within the execution unit, and  $P$  is the number of cycles lost due to pipeline latency. Since the PDLX is a single-issue multiprocessor, the instruction throughput  $T = I/T$  is a quantity less than or equal to one. This quantity can be multiplied by the system clock frequency to obtain the *effective instruction throughput*  $ET = (I/T) * f_{CLK}$ , which allows us to compare the execution of the same assembler code on different PDLX implementations running at different speeds. Figure 6 illustrates the results obtained running three different assembler programs: the effective instruction throughput is reported on the y-axis, while each discrete point on the x-axis corresponds to a different PDLX implementation with a different fixed amount of latency on some channels. We focused on two specific channels on Figure 5: channel  $C_a$

between the *Execution Unit* and the *Data Cache* and channel  $C_b$  between the *Fetch Unit* and the *Instruction Cache*. We assumed that the wires grouped in these two channels represent the critical path of the PDLX design and that, after segmenting them (by inserting relay stations), we could afford to raise the clock frequency appropriately. We varied the latency on the two channels as follows: going from left to right on the  $x$ -axis, the 18 data-points represent 18 implementation cases which can be grouped in three subsets in correspondence to latency values  $L_a$  for  $C_a$  equal respectively to 0, 1, 2 clock cycles. Each of these subsets contains 6 data-points corresponding to latency values  $L_b$  for  $C_b$  going from 0 to 5 clock cycles. Finally, for each implementation case, we set the system clock frequency as  $f_{CLK} = \min\{L_a, L_b\} + 1$ . The plot illustrates how different PDLX implementations perform under the same data stimulus, showing that the throughput values are consistent across different benchmarks. All implementations are functionally equivalent by construction, being obtained simply by changing the number of relay stations on the channels and with no need of re-designing any PDLX module. The insertion of relay stations can be made at late stages in the design process, after detailed information can be extracted from the physical layout, to “fix” those channels whose latency is longer than the desired clock cycle.

## 7. Conclusions and Future Work

We proposed a new “*correct-by-construction*” synthesis methodology for designing very large digital systems by assembling IP functional modules. The modules communicate by exchanging data on communication channels according to an appropriate protocol, which guarantees a correct system behavior independently from channel latencies. As a consequence, a robust implementation is achieved in a shorter time by reducing the multiple iterations between logical and physical design. We developed a set of RTL libraries for a specific latency insensitive protocol and we used them to design a latency insensitive implementation of PDLX, an out-of-order microprocessor with speculative execution. There are several avenues for further investigations: (1) application to other designs, particularly in the multimedia domain, (2) study of the impact of our approach on other design metrics such as area and, especially, power, (3) extension of the theory to *speculation insensitive* protocols.

## Acknowledgments

The authors would like to thank Luciano Lavagno and Patrick Scaglia for their support and useful discussions.

## Notes

1. Observe that most hardware systems can be easily made stallable: for instance, consider any sequential logic block together with a *gated clock* mechanism, or a Moore finite state machine with an extra input that can force it to stay in the current state while emitting a “flag signal”.
2. A bounded skew is allowed between the different branches of a net.
3. But the communication bandwidth would be limited by the inverse of the longest of the round trip times between pairs of communicating relay stations.
4. Recall that the primary reason for this double capacity is the need of avoiding losing data while spending one cycle to propagate the stop signal.

## References

- [1] P. Beerel and T.H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 581–587. IEEE Computer Society Press, November 1992.
- [2] Janusz A. Brzozowski and Jo C. Ebergen. On the delay-sensitivity of gate networks. *IEEE Transactions on Computers*, 41(11):1349–1360, November 1992.
- [3] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.
- [4] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Latency Insensitive Protocols. In *Proc. of the 11th Intl. Conf. on Computer-Aided Verification (N. Halbwachs and D. Peled editors)*, pages 123–133. LNCS 1633, Springer, July 1999.
- [5] Wesley A. Clark. Macromodular computer systems. In *AFIPS Conference Proceedings: 1967 Spring Joint Computer Conference*, volume 30, pages 335–336, Atlantic City, NJ, 1967. Academic Press.
- [6] Wesley A. Clark and Charles E. Molnar. The promise of macromodular systems. In *Digest of Papers of the Six Annual IEEE Computer Society International Conference*, pages 309–312, San Francisco, CA, 1972. IEEE Press.
- [7] J. Cong. Challenges and Opportunities for Design Innovations in Nanometer Technologies. In *SRC Design Sciences Concept Paper*, December 1997.
- [8] J. Cong, L. He, K.Y. Khoo, C.K. Koh, and Z. Pan. Interconnect Design for Deep Submicron ICs. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 478–585. IEEE, November 1997.
- [9] D. Matzke. Will Physical Scalability Sabotage Performance Gains? *IEEE Computer*, 8(9):37–39, September 1997.
- [10] D. Sylvester and K. Keutzer. Getting to the Bottom of Deep Submicron. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1998.
- [11] Al Davis and Steven M. Nowick. Asynchronous circuit design: Motivation, background, and methods. In Graham Birtwistle and Al Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 1–49. Springer-Verlag, 1995.
- [12] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.
- [13] Jo C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3):107–119, 1991.
- [14] W. Gosti, A. Narayan, R.K. Brayton, and A. Sangiovanni-Vincentelli. Wireplanning in Logic Synthesis. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 26–33. IEEE, November 1998.

- [15] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 1996.
- [16] S. Hojat and P. Villarrubia. An Integrated Placement and Synthesis Approach for Timing Closure of Power PC Microprocessors. In *Proc. Intl. Conf. on Computer Design. VLSI in Computers and Processors*, pages 206–210. IEEE, October 1997.
- [17] M. B. Josephs and J. T. Udding. An overview of DI algebra. In *Proc. Hawaii International Conf. System Sciences*, volume I. IEEE Computer Society Press, January 1993.
- [18] L.N. Kannan, P.R. Suaris, and H. Fang. A Methodology and Algorithms for Post-Placement Delay Optimization. In *Proc. of the Design Automation Conf.*, pages 327–332, June 1994.
- [19] H. Kapadia and M. Horowitz. Using Partitioning to Help Convergence in the Standard-Cell Design Automation Method. In *Proc. of the Design Automation Conf.*, pages 592–597, June 1999.
- [20] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 56–62, June 1994.
- [21] E. A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Transactions on Computer-Aided Design*, 17(12):1217–1229, December 1998.
- [22] A. Lu, H. Eisenmann, G. Stenz G., and F.M. Johannes. Combining Technology Mapping with Post-Placement Resynthesis for Performance Optimization. In *Proc. Intl. Conf. on Computer Design. VLSI in Computers and Processors*, pages 616–621. IEEE, October 1998.
- [23] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.
- [24] Charles E. Molnar, Ting-Pien Fang, and Frederick U. Rosenberger. Synthesis of delay-insensitive modules. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 67–86. Computer Science Press, 1985.
- [25] R. H. J. M. Otten and R. K. Brayton. Planning for Performance. In *Proc. of the Design Automation Conf.*, pages 122–127, June 1998.
- [26] Fred U. Rosenberger, Charles E. Molnar, Thomas J. Chaney, and Ting-Pien Fang. Q-modules: Internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, C-37(9):1005–1018, September 1988.
- [27] A. Salek, J. Lou, and M. Pedram. A DSM design flow: Putting Floorplanning, Technology Mapping and Gate Placement Together. In *Proc. of the Design Automation Conf.*, pages 287–290, June 1998.
- [28] K. Sato, M. Kawarabayashi, H. Emura, and N. Maeda. Post-Layout Optimization for Deep Submicron Design. In *Proc. of the Design Automation Conf.*, pages 740–745, June 1996.
- [29] S.J. Schaffer and W.W. LaRue. BONeS DESIGNER: a Graphical Environment for Discrete-Event Modeling and Simulation. In *MASCOTS '94. Proc. of the 2nd. Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 371–374, Los Alamitos - CA, February 1994. IEEE.
- [30] The DLX Software. <ftp://max.stanford.edu/pub/hennessy-patterson/software>.
- [31] R. M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal Research and Development*, 11:25–33, January 1967.
- [32] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits. *Distributed Computing*, 1(4):197–204, 1986.

# EXPLORING PERFORMANCE TRADEOFFS FOR CLUSTERED VLIW ASIPS

Margarida F. Jacome, Gustavo de Veciana and Viktor Lapinskii<sup>1</sup>

*Department of Electrical and Computer Engineering*

*The University of Texas, Austin, TX, USA*

## Abstract

VLIW ASIPs provide an attractive solution for increasingly pervasive real-time multimedia and signal processing embedded applications. In this paper we propose an algorithm to support trade-off exploration during the early phases of the design/specialization of VLIW ASIPs with clustered datapaths. For purposes of an early exploration step, we define a parameterized *family of clustered datapaths*  $D(m, n)$ , where  $m$  and  $n$  denote *interconnect capacity* and *cluster capacity* constraints on the family. Given a kernel, the proposed algorithm explores the space of feasible clustered datapaths and returns: a datapath configuration; a binding and scheduling for the operations; and a corresponding estimate for the best achievable latency over the specified family. Moreover, we show how the parameters  $m$  and  $n$ , as well as a *target latency* optionally specified by the designer, can be used to effectively explore trade-offs among delay, power/energy, and latency. Extensive empirical evidence is provided showing that the proposed approach is strikingly effective at attacking this complex optimization problem.

## 1. Introduction

Real time multimedia and signal processing embedded applications often spend most of their cycles executing a few time critical code segments (kernels) with well defined characteristics, making them amenable to processor specialization. Moreover, these computation intensive kernels often exhibit a high degree of inherent instruction level parallelism (ILP). Thus, Very Large Instruction Word (VLIW) Application Specific Instruction Set Processors (ASIPs) provide an attractive solution for such embedded applications.

Traditionally, the datapaths of VLIW machines have been based on a single register file shared by all functional units (FUs). The central register file provides internal storage as well as switching, i.e., interconnection among the FUs and to/from the memory system. Unfortunately, this simple organization does not scale well with the large number of functional units typically required to take advantage of the ILP present in the embedded applications of interest. Indeed, it has been shown in [14] that, for  $N$  FUs connected to a register file, the area of the register file grows as  $N^3$ , the delay as  $N^{3/2}$ , and power dissipation as  $N^3$ .

---

<sup>1</sup>Author is currently with AmmoCore Technology, Santa Clara, CA.

In short, as the number of FUs increases, internal storage and communication between FUs quickly becomes the dominant, if not prohibitive factor, in terms of delay, power dissipation, and area.

A key observation is that the delay, power dissipation and area associated with the storage organization can be dramatically reduced by restricting the connectivity between FUs and registers, so that each FU can only read and write from/to a limited subset of registers[14]. Thus a key dimension of VLIW ASIP specialization is *clustering*, i.e., the development of datapaths comprised of clusters of FUs connected to local storage (the cluster's register file). Although by moving from a centralized to a distributed register file organization one can reap significant *delay*, *power* and *area* savings, this type of specialization may come at a cost. One may have to transfer data among these register files (i.e., datapath clusters), possibly resulting in increased *latency*.

More concretely, consider a family of clustered datapaths wherein each cluster has no more than a given number of FUs, irrespective of type. We shall refer to this constraint as a *cluster capacity* constraint. Intuitively, as the cluster capacity decreases (and thus the number of ports and size of the associated register file decrease), one expects combinational delay as well as power dissipation to decrease, while the number of clock cycles (latency) required to execute a given kernel to increase. In the limit, when comparing a clustered machine to a hypothetical centralized machine with the same number of FUs, one expects to be able to sustain higher clock rates in the clustered machine, but at the cost of increased latency, due to the need to move data among register files.

Moreover, as cluster capacity decreases, one also expects power dissipation to decrease with respect to the centralized machine. Indeed, clustered machines would have local register files that have fewer ports and are smaller than the single register file of the centralized machine, thus achieving a less costly (local) switching inside each cluster. Unfortunately, switching may also be needed among clusters, i.e., there may be a need to perform move (or copy) operations across register files of different clusters, with a corresponding undesirable effect in *energy consumption*.

Note that while performance and power/energy are a major concern in embedded applications, silicon area (*per se*) is not necessarily a concern, since with today's levels of integration one can cost-effectively place large numbers of transistors on a single chip[1, 13]. Thus, in exploring the design space with respect to the impact on performance and power/energy of different cluster capacities, one can allow for an unbounded number of clusters – at least during the early phases of the exploration. In fact, as observed in [5], in signal processing applications with high ILP, in order to achieve high throughput one should expect datapaths with a *large* number of functional resources and low resource sharing. Thus, placing an upper bound on the total number of functional resources was considered inadequate. One should however consider a constraint on the *inter-*

*connect capacity*, since congestion (during data transfers across clusters) may lead to major performance penalties, and the interconnect structure has a significant impact on the relevant figures of merit discussed above (i.e., delay and power/energy).

In summary, when considering the specialization of a datapath to a given kernel, one should seek solutions with a (possibly large) number of clusters working (quasi-) independently. Note that such configurations are the “ideal” ones, in that they decrease power and delay, by taking advantage of locality in the computations, while incurring no (significant) latency/energy penalties due to switching across clusters.

In this paper we propose an algorithm for estimating the minimum latency achievable by a *family of clustered machines* – the family is defined by the cluster and interconnect capacity constraints discussed above. In particular, given a kernel and capacity constraints, our algorithm explores the space of feasible clustered datapaths and returns: (1) an “optimal” datapath configuration; (2) a binding and scheduling for the operations; and (3) a corresponding estimate for the minimum achievable latency over the family of clustered datapaths.

The algorithm proposed in this paper is a fundamental tool for the *early exploration* required to design specialized clustered datapaths. To the best of our knowledge, this problem has not been addressed before, see §5. We formalize the problem under consideration in §2. Our novel approach is based on: (1) an effective decomposition of the problem into a sequence of simpler sub-problems; and (2) an aggressive heuristic pruning of the large design spaces defined by these sub-problems. This is discussed in §3. Extensive empirical evidence is provided in §4 showing that the approach is strikingly effective at attacking this exceedingly complex problem. Moreover, the discussions therein illustrate how the algorithm can be used within a general design space exploration framework. Conclusions are presented in §6.

## 2. Problem Definition

Our goal is to support early phases of the design of VLIW machines specialized to execute time critical segments (kernels) of target embedded applications. The identification of these time critical segments, represented as basic blocks, superblocks, etc., is thus performed prior to this exploration step[10, 9]. These kernels are represented as dataflow graphs (DFGs), i.e., in terms of a DAG,  $G(V, E)$ , where the nodes  $V$  represent *operations* to be carried out on datapath functional resources, e.g., adds, multiplies, etc., also called activities, and the edges  $E \subset V \times V$  represent *data objects* that are “produced” and “consumed” by activities during the flow of execution, see e.g., Figure 3. As discussed in the sequel, the DFG model of the application may be modified to include nodes corresponding to move/copy operations (i.e., data transfers across clusters) re-

quiring the interconnect resources. The location of such moves only becomes clear once a datapath and binding of functional operations to the datapath's resources begin to be specified.

The problem to be addressed is one of simultaneous allocation and binding, subject to coarse hierarchical "structural constraints." We parameterize families of clustered datapaths  $D(m, n)$  as follows. Each datapath may contain several *independent components*, see e.g., Figure 1. Each independent component, in turn, contains a collection of clustered FUs, i.e., ALUs and multipliers that share a common register file. The clusters within each component share a local interconnect structure with capacity not exceeding  $m$ . Each cluster has no more than  $n$  FUs, but no limit is placed on the number of components and associated clusters that can be instantiated in the datapath.

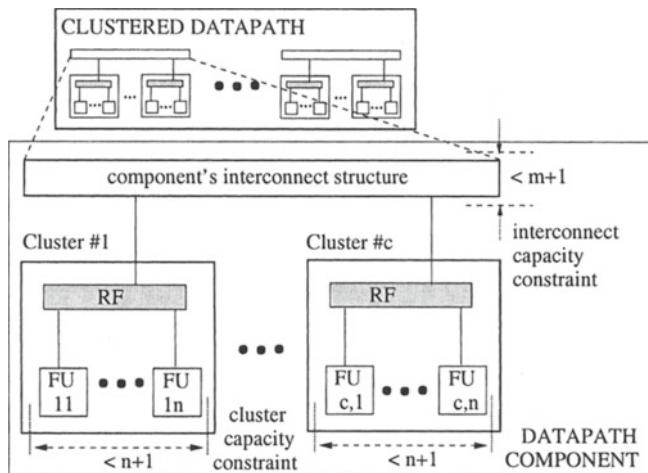


Figure 1. A component of a clustered VLIW datapath.

A feasible binding of a DFG to a clustered datapath specifies on which clusters activities will (and can) execute. Given a binding of a dataflow to a datapath one can schedule activities so as to minimize execution latency. The problem to be addressed can be stated as follows:

**Problem 1** *The problem  $P(m, n, DFG)$  is to find a datapath  $D^* \in D(m, n)$  and a binding and scheduling of the DFG to  $D^*$  that results in a small, if not minimal, execution latency. We let  $T^*(m, n, DFG)$  denote the minimal execution latency that can be achieved.*

Note that our coarse parameterization of datapaths is aimed at reducing the size/complexity of the design space for an *initial* exploration conducted at a high-level of abstraction. (This is not unlike approaches to similar high-complexity CAD / compilation problems, which typically resort to abstraction and problem

phasing, see e.g., [5, 10, 9].) Specifically, we do not model limitations on register file capacities for each cluster, and only consider data transfers of temporary values across clusters. Nevertheless, a solution to our abstract problem enables an effective exploration of a huge space of possible designs and thus datapath specialization along two critical dimensions: cluster and interconnect capacities. Promising datapath configurations are then considered in more detail, during subsequent phases of the VLIW ASIP design/specialization process[11].

### 3. Algorithm to support VLIW datapath specialization

The pseudo-code below exhibits the main high-level tasks of the proposed algorithm. The algorithm includes two main decompositions. The first, performed by the function **Gen-IDFGs**, corresponds to partitioning the DFG into a set of independent DFGs, or IDFGs, which can be addressed separately. Specifically, given a DFG we consider the associated undirected graph, and identify its connected components. Each connected component corresponds to an IDFG. Clearly, such IDFGs constitute ideal “chunks” of computation that can be performed on a single datapath component, requiring no local communication with other components. Thus for each IDFG the goal is to find an “optimal” clustered structure for the associated component.

```
Algorithm (m,n,DFG,TL) {                                     // initialization
    TL = max[ TL, ASAP(DFG)];
    solution = (datapath, binding, schedule, latency) = (0, 0, 0, TL);
    UpdateSolution(solution);
    Set-IDFGs = GenIDFGs(DFG);                                // generate a set of IDFGs
    for each IDFG ∈ Set-IDFG {                               // decomposition 1
        s1 = oneClusterSolution(m,n,IDFG);                  // try 1 cluster solution
        if ( latency(s1) ≤ TL){ UpdateSolution(s1); }
        else { s2 = multClusterSolution(m,n,IDFG);          // decomposition 2
            if (latency(s2) < latency(s1)) {                // choose min latency solution
                UpdateSolution(s2); }
            else { UpdateSolution(s1); }
        }
    } return (solution); }
```

The second decomposition, which will be explained in more detail below, is used to synthesize multi-cluster datapath components suitable for each IDFG. The key idea is to decompose an IDFG into the operations which are the most difficult to handle. Each operation is given a *difficulty ranking* assessing the likelihood that latency penalty steps will be incurred due to limited cluster or

interconnect capacity, i.e., resulting from serializing operations within a cluster due to limited FU capacity, or from introducing data transfers across clusters. Given such a ranking, we extract a set of operations with highest rankings and consider the induced sub-IDFG. Our approach then determines datapath clusters, bindings, and a partial schedule which are suitable for the induced sub-IDFG in the sense of minimizing latency penalties. The next IDFG sub-problem is addressed in a similar fashion.

In the sequel we focus on the key conceptual contributions of our approach, which are the decomposition of an IDFG into sub-problems, and a systematic method for synthesizing multi-cluster datapath solutions for such sub-problems.

### 3.1 Difficulty ranking function and decomposition

Our algorithm keeps track of, and updates, a global variable denoted target latency  $TL$ . The target latency is either specified by the designer or set to be the as-soon-as-possible (ASAP) latency bound for the DFG, denoted ASAP(DFG). Given  $TL$ , for each operation  $o$  in the DFG, we compute  $\text{mobility}(o) = \text{ALAP}(o) - \text{ASAP}(o)$ , see e.g., Figure 3.<sup>1</sup> The mobility corresponds to the operation's difficulty ranking. Clearly an operation with low mobility is likely to be difficult to handle as it has few temporal degrees of freedom to deal with possible serialization or data transfers among clusters.

We will be progressively constructing a global solution, i.e., a datapath, a binding, a schedule, and a feasible latency for the complete problem, based on considering several *sub-problems*. Each time a sub-problem is solved, the function **UpdateSolution()** is invoked to update the current global solution. This involves several tasks. First, if the sub-problem solution exceeds the current target latency, the global solution is updated accordingly. Then, the sub-problem operations and data transfers are *anchored* on the corresponding scheduling steps. Finally, the mobility of operations not yet anchored is recomputed.

The primary consideration driving the algorithm is to minimize execution latency. A secondary consideration is to minimize the number of data transfers among clusters. Thus for each IDFG (or IDFG sub-problem) the primary goal is to find a solution either within the current target latency, or resulting in a minimal increase in target latency. In each case, a *single* cluster solution, generated by **oneClusterSolution()**, and a *multiple* cluster solution, generated by **multClusterSolution()**, may be obtained. The function **latency()** returns the execution latency of a solution to a sub-problem. To satisfy the secondary goal, preference is always given to *single cluster solutions* that can achieve the current target latency, or provide the same or better latency than multi-cluster solutions. This is done in an attempt to reduce the number of clusters and data transfers in the final solution – see comments in §4.

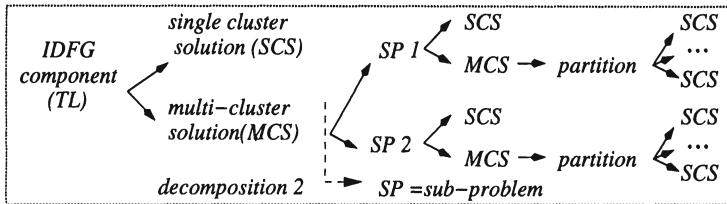


Figure 2. Simplified problem decomposition strategy.

When a multi-cluster solution is sought for a given IDFG, our second decomposition takes place, see Figure 2. As mentioned previously, multi-cluster solutions are obtained by first tackling a sub-problem associated with the most difficult set of operations in an IDFG, denoted sub-problem 1. The function **ExtractSubproblem1()**, called within **multClusterSolution()**, extracts an induced sub-IDFG associated with the operations with minimum mobility, denoted  $MM$ , and those with mobility  $MM + 1$ , if they have a direct producer or consumer with mobility  $MM$ . The rationale for including the second type of operations is that, if they were bound to a different cluster, they would incur additional data transfers reducing their mobility by at least one, and thus making them as difficult to handle as operations with mobility  $MM$ . For our benchmarks, this heuristic always selected more than 60 % of the IDFG's operations. For the example in Figure 3 the induced sub-IDFG associated with extracting the first sub-problem is shown on the right. The induced sub-IDFG includes the subset of nodes satisfying the criterion, and edges among those nodes.<sup>2</sup> In the simplest version of our algorithm, only one additional sub-problem is considered, namely **ExtractSubproblem2()**, associated with the operations not considered in the first sub-problem, see Figure 2.

The key property underlying this sub-problem decomposition for IDFGs is as follows. The first problem is associated with the most difficult operations, but includes only a relaxed set of sequencing constraints. This makes it usually easier to solve, i.e., to find a suitable clustered datapath component, binding and schedule resulting in a reduced latency. If the first sub-problem cannot be solved within the current target latency, then invariably the original IDFG would not be feasible within that target latency. Formalizing and proving this fact is simple and gives the following:

**Fact 3.1** Suppose a dataflow graph  $\text{subDFG}$  induced by a subset of operations in a DFG. Then  $T^*(m, n, \text{subDFG}) \leq T^*(m, n, \text{DFG})$ .

Thus, if the first sub-problem incurs a latency penalty, i.e., forces an increase in the current target latency, then that penalty will persist and typically make the second sub-problem easier, if not trivial, to solve.

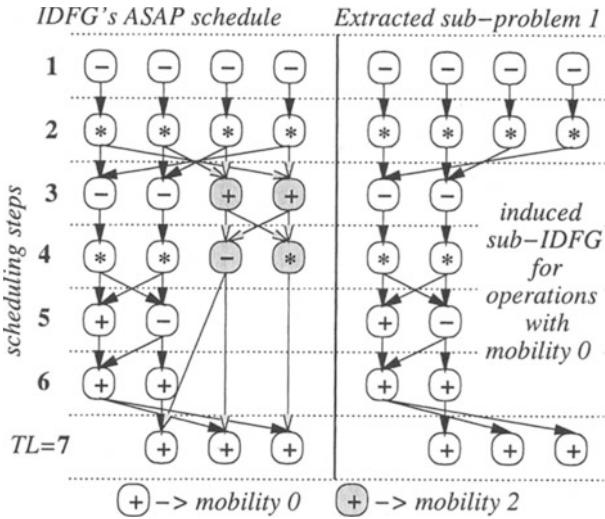


Figure 3. Ranking operations and extracting induced sub problems.

*Multi-cluster* datapath components for an IDFG are synthesized by decomposing the IDFG into several *single cluster* sub-problems, see Figure 2. These may correspond to the two IDFG sub-problems discussed above, or further decompositions of these to smaller sub-problems, as discussed in the next section. In progressively synthesizing a multi-cluster solution, several single cluster solutions to parts of the IDFG are composed. In an attempt to reduce the number of clusters in the datapath components, prior to instantiating a new cluster and assigning it to an IDFG sub-problem, we first attempt to bind and schedule the associated operations on existing clusters. If the capacity available in such clusters is insufficient, i.e., if we fail to meet the current target latency, a new cluster is allocated, and its FU's are selected to lead to the smallest latency penalty for the sub-problem under consideration.

In this process the execution latency is evaluated by scheduling operations and moves using a simple modification of the list scheduling algorithm[3] which is described in the sequel. Moves, i.e., data transfers requiring the use of the shared interconnect resource, are inserted when operations sharing an edge are bound to different clusters. Thus, when a multi-cluster component is being synthesized, the accrued load on the shared interconnect resource impacts the scheduling of move operations associated with each of the IDFG's sub-problems.

### 3.2 Finding multi-cluster solutions for IDFG sub-problems

Let us first consider a simple example where each cluster can have at most one FU, i.e.,  $n = 1$ . In this case, ideally, one identifies long independent strings of operations that require the same type of FU within the IDFG sub-problem, and binds such strings to independent clusters of the right type. By binding operations along such string to the same cluster, one avoids requiring data transfers for the edges (associated data objects) along the string. By ensuring the string includes operations of the same type, one avoids a mismatch between the load placed on the cluster and the capacity of the cluster. This suggests a general heuristic to determine datapath clusters and operation bindings for IDFG sub-problems that do not consist of independent strings and with nontrivial ( $n > 1$ ) cluster capacities. The idea is to find sets of operations corresponding to subtrees, rather than strings. We call these sets *vertical aggregations* and recognize that binding such aggregations to clusters with appropriate capacities might translate to reduced penalties due to data transfers. At the same time, it is of interest to identify sets of consecutive operations that have compatible resource requirements. We call these *horizontal aggregations* and recognize that binding these to *compatible* clusters might also avoid excessive serialization within limited capacity clusters.

Thus, our algorithm first determines vertical and horizontal aggregations of operations for IDFG sub-problems. Based on these, it creates possible partitions of its operations. By binding each element in the partition, i.e., set of operations, to a compatible cluster, we synthesize candidate clustered datapaths and bindings. The operations are then scheduled to evaluate the solution. We discuss these steps in more detail below.

**Vertical Aggregation.** Given an extracted sub-IDFG, vertical aggregation creates a collection of subsets of operations  $\mathcal{V}$  corresponding to *sub-trees* in the sub-IDFG, see e.g., Figure 4. Since vertical aggregation is attempted for a sub-IDFG when a single cluster solution appears to be inadequate/inferior (see Figure 2), one can assume at least two clusters are required. Thus, in attempting to partition the sub-IDFG into vertical aggregates, we ensure that there always exist *at least* two ongoing subtrees, i.e., cluster parallelism to be exploited. As explained below, this requirement translates to avoiding *full merging*, i.e., avoiding aggregating all the operations within a single tree in any given “layer” of the sub-IDFG.

Vertical aggregates are generated in both a top-down and bottom-up fashion, considering one layer at a time – a layer corresponds to the set of operations falling on a given step in the ASAP schedule for the sub-IDFG. For the top-down case, we begin by positing that each activity in the top layer (initialized

to the first step of its ASAP schedule) corresponds to an independent tree. At each step one considers growing and/or merging trees on the previous layer following the edges between operations in the two layers, see e.g., Figure 4. Such growing/merging takes place only if (1) the resulting aggregates correspond to subtrees and (2) have not resulted in a single aggregate, i.e., *full merging* of all the operations between the current layer and the top layer. When such growing/merging of trees violates one of these conditions (see e.g. the transitions from Layer 4 to 5 in Figure 4) the current subsets, e.g.,  $V1$  and  $V2$ , are added to the collection of vertical top-down aggregations  $\mathcal{V}_t$ , and the process restarts with the current layer as the new top layer. Thus, for our example, Layer 5 becomes the new top layer and the activities in the layer are assumed to correspond to independent trees. In our example the transition from Layer 5 to 6 again leads to a full merging, and thus the sets  $V3$  and  $V4$  with single operations are added to  $\mathcal{V}_t$ , and the process restarts on Layer 6, resulting in two additional vertical aggregation sets  $V5$  and  $V6$ . In the sequel we will refer to aggregations that contain only one operation, as is the case for  $V3$  and  $V4$ , as *trivial* and will attempt to merge these with larger vertical or horizontal aggregates.

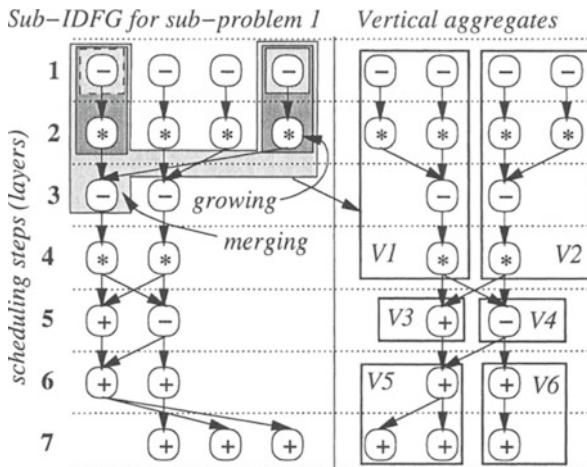


Figure 4. Vertical aggregation.

Bottom-up vertical aggregations, denoted by  $\mathcal{V}_b$ , are generated in the same fashion but starting from the bottom. Although for our example they result in the same aggregations, in general this is not the case. Since the sub-IDFG is a DAG, identification of the vertical aggregates is straightforward having only linear complexity in the number of nodes. Finally, we note that a more refined model for vertical aggregation could be obtained by hierarchically keeping track of all merge points seen in this process. Until now, we have found that in practice these fine grained partitions are not useful, i.e., even when data transfer delays

are small (1 cycle), scattering small aggregates across many clusters was never advantageous from a latency point of view, see §4.

**Horizontal Aggregation.** The horizontal aggregation step creates a collection of aggregates  $\mathcal{H}$  corresponding to sub-IDFG operations on consecutive layers with *compatible* loads. To do so, we determine the load profile for the operations on each layer, e.g., the multiplier load  $multload(i)$  on layer  $i$  is the sum of  $(mobility(o) + 1)^{-1}$  for all operations on layer  $i$  requiring a multiplier. We compute the ALU load,  $ALUload(i)$ , in a similar fashion.<sup>3</sup> Intuitively the resource load on two layers is compatible if there exists a cluster type that is a good match for both. Recall that a feasible cluster type is defined by the number of ALUs and multipliers it includes, so long as their sum does not exceed our constraint  $n$ . We define a notion of load *compatibility* for a constraint  $n$  as follows. For each layer  $i$  we determine the set of *all* feasible clusters types,  $CT(i)$  that would be able to support the layer's resource load in a minimum number of steps - non integral loads are rounded up. Two (or more) consecutive layers, say  $i$  and  $i + 1$ , are said to be compatible if  $CT(i) \cap CT(i + 1) \neq \emptyset$ , i.e., if there exists a cluster type that would be able to support their individual loads in a minimal number of steps. For illustration purposes, consider a hypothetical example including the following consecutive layers: Layer 1, with  $ALUload(1) = 2$  and  $multload(1) = 2$ ; Layer 2, with  $ALUload(2) = 2$  and  $multload(2) = 0$ ; and Layer 3, with  $ALUload(3) = 3$  and  $multload(3) = 0$ . Assuming a cluster capacity  $n = 3$ , the corresponding feasible cluster types would be  $CT(1) = \{2A1M, 1A2M\}$ ,  $CT(2) = \{2A1M, 3A\}$ , and  $CT(3) = \{3A\}$ . For this example, Layers 1 and 2 would be compatible since  $CT(1) \cap CT(2) = \{2A1M\} \neq \emptyset$ . Similarly, Layers 2 and 3 would be compatible since  $CT(2) \cap CT(3) = \{3A\} \neq \emptyset$ .

Figure 5 exhibits  $CT(i)$  when  $n = 2$  for our example. Layers  $i = 5, 6, 7$  have  $CT(i) = \{2A\}$  (i.e., clusters with 2 ALUs) and are thus jointly compatible, so a single horizontal aggregation of operations, the set  $H1$ , is placed in the collection  $\mathcal{H}$ . In general  $\mathcal{H}$  includes the largest sets of compatible horizontal aggregations. Since our notion of load compatibility among layers is not transitive, in some cases such aggregates may overlap. For example, in the hypothetical case introduced above, the horizontal aggregation formed by Layers 1 and 2 is not compatible with that formed by Layers 2 and 3, since  $\{2A1M\} \cap \{3A\} = \emptyset$ . Thus, two distinct (overlapping) horizontal aggregations would have been formed.

**Optimization Algorithm.** The third step in determining a multi-cluster solution is to jointly search for a “good” overall partition of sub-IDFG operations and corresponding binding of these partitions to suitable cluster types. Although this is a very complex task when a flat design space is considered,

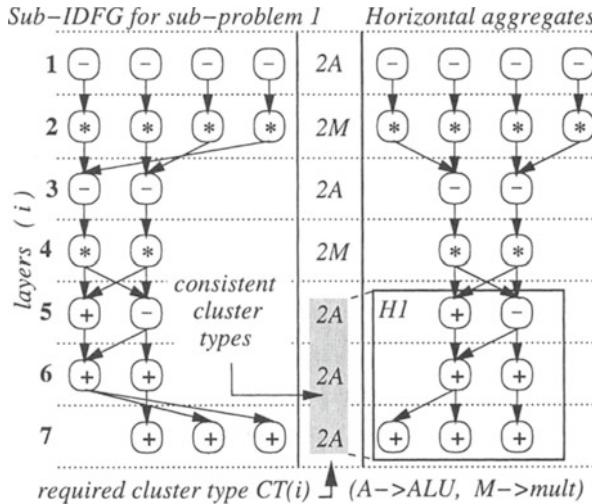


Figure 5. Horizontal aggregation.

in our approach the search space is dramatically reduced by the horizontal and vertical aggregates determined in the previous two steps.

Specifically, our optimization heuristic proceeds as follows.

Based on  $\mathcal{V}_t$ ,  $\mathcal{V}_b$  and  $\mathcal{H}$ , we create coverings of the sub-IDFG's operations.<sup>4</sup> In fact, we systematically generate several such coverings, as follows: (1) one based on  $\mathcal{V}_t$ ; (2) one based on  $\mathcal{V}_b$ ; and (3) several based on one or more sets in  $\mathcal{H}$ , covering each of the uncovered horizontal slices entirely with elements of either  $\mathcal{V}_t$  or  $\mathcal{V}_b$ . In most cases we have but a few horizontal aggregations, leading to a limited number of possible covers. In this process we ensure that no set in a cover is fully contained within another, but can not ensure that the obtained covers are in fact partitions.

In the next phase of our algorithm we exhaustively derive partitions from each of the obtained covers, i.e., any operation that is included in two or more sets in a cover is removed and assigned to only one of them. Further *boundary* perturbations creating additional partitions can be useful, e.g., merging *trivial* vertical aggregation sets with neighboring aggregates or shifting operations across aggregates in layers where the interconnect capacity has been exhausted. Because these perturbations involve only operations on the boundaries between large aggregates, the demands in generating these are not excessive. This process eventually transforms each cover into one of many possible partitions, see e.g., Figure 6. Although this process can grow exponentially in complexity, in practice the number of covers/partitions that were generated in all our case studies did not justify any further pruning. Specifically, given a partition, an exhaustive generation of boundary perturbations and scheduling of the corre-

sponding alternatives took no more than a few seconds on an Sun UltraSparc I for the benchmarks shown in Table 1.

Alternative multi-cluster datapaths are then generated based on these partitions. Each element in a partition corresponds to a single cluster problem, thus it makes sense to consider partitions with the fewest number of sets first. As discussed in §3.1 and shown in Figure 2, single cluster solutions to each partition are composed to generate multi-cluster solutions to the (sub)IDFG, which are compared based on the execution latency they achieve.

For our ongoing example, Figure 6 shows the best partition. The suitable cluster types when  $n = 2$  were determined to be 1A1M (i.e., 1 ALU and 1 multiplier) and 2A, as shown in the figure. The minimum latency schedule (for interconnect capacity  $m = 2$ ) was determined to be 10 steps, which is optimal for the given capacity constraints. At this point the solution for sub-problem 2 would be generated.

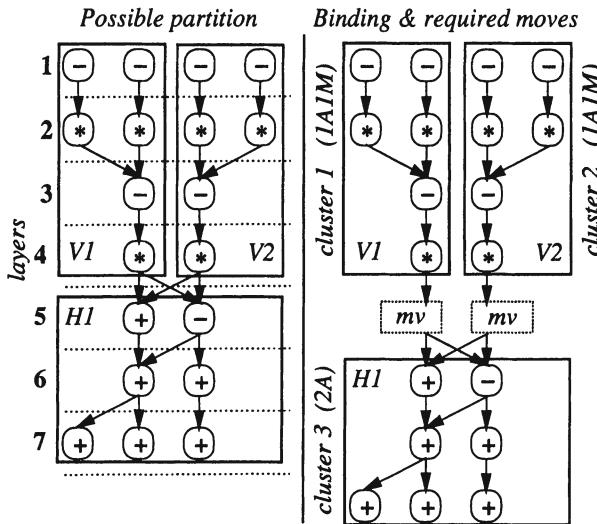


Figure 6. Generating covers, partitions, and deriving clusters types.

**Modified List Scheduling.** Execution latencies are determined using the following modified list scheduling algorithms. Given an IDFG, its *first* subproblem is scheduled (for the derived binding) using a standard list scheduling priority function (longest path to any sink operation), enhanced by a tie breaking policy. Specifically, in the case of a tie, operations that are ancestors of move operations are given higher priority. When an operation cannot be scheduled within its time frame<sup>5</sup>,  $TL$  is incremented, the time frames are updated, and the algorithm is repeated.

Recall that, when a solution for an IDFG's first sub-problem is found, the operations of the sub-IDFG, including moves, are anchored to their associated scheduling steps, and the time frames and mobility of the remaining operations are recomputed. Accordingly, scheduling for an IDFG's *second* sub-problem is performed as follows. A modified list scheduling algorithm traverses scheduling steps from 1 to  $TL$  and, at each step, schedules as many ready operations<sup>6</sup> as available resources permit, using mobility as the priority function. As in the previous case, if a tie occurs, priority is given to nodes which are ancestors of *unscheduled* moves. When an operation cannot be scheduled within its time frame, the scheduling process stops, the anchoring of *all* operations is released, and an overall scheduling is performed for the same binding function and the same initial target latency, using the list scheduling algorithm described in the previous paragraph.

## 4. Experimental Results

Table 1 shows the results produced by our algorithm for a number of representative benchmark kernels. For simplicity, operations and data transfers were assumed to take 1 cycle in all test cases, but our approach is general. The three Discrete Cosine Transform (DCT) algorithms (Lee, DIT and DIF [8]) typify complex kernels with high potential for ILP. The three filters (Elliptic, Autoregression and Avenhaus [6]) typify complex kernels with less potential for ILP. Information on the number of connected components (IDFGs), their critical path (i.e., absolute minimum latency), and number of operations for each IDFG is provided in Column 1. For each benchmark, we considered datapaths with interconnect capacity  $m = 2$  and cluster capacity constraints of  $n = 2, 3, 4$ . For each of the 18 problem instances considered, the derived clustered datapath and the associated achievable latency  $L$  are shown in the table. Also shown is the total number of data transfers, abbreviated DTs, with subtotals per datapath component.

We start by noting that our algorithm consistently found minimum latency penalty solutions for the specified capacity constraints, strongly suggesting that our aggressive design space pruning is effective.<sup>7</sup> Moreover, for all cases but one, the achievable latency was driven by *sub-problem* 1 of an IDFG, confirming the effectiveness of our decomposition heuristic based on difficulty rankings. The exception occurred for the DIT DCT benchmark, with constraints  $(m, n) = (2, 4)$ . In this case, contention for interconnect resources caused an additional latency penalty of 1 step for the second sub-problem associated with the IDFG. (Note that this benchmark has a single IDFG with 48 nodes and critical path of 7 steps, thus contention on the local interconnect for the datapath component was likely to occur, if a high degree of ILP was to be achieved.) Note however that this did not occur for cluster capacities  $n = 2, 3$ . In these two cases an increased

Benchmarks	(m,n)	L	Datapath	# DTs
<u>DCT-Lee:</u> 49 ops (29 add/subs, 20 mults) 2 IDFGs, CP=9 <u>IDFG1:</u> 28 ops, CP=9 <u>IDFG2:</u> 21 ops, CP=7	(2,4)	10	<u>IDFG1:</u> 2(2A2M)=2 <u>IDFG2:</u> (2A2M)=1	5 (5+0)
	(2,3)	12	<u>IDFG1:</u> (2A1M)+(1A1M)=2 <u>IDFG2:</u> (2A1M)=1	5 (5+0)
	(2,2)	12	<u>IDFG1:</u> 3(1A1M)=3 <u>IDFG2:</u> 2(1A1M)=2	11 (7+4)
	(2,4)	9	<u>IDFG1:</u> 2(2A2M)=2 <u>IDFG2:</u> (2A1M)=1	2 (2+0)
	(2,3)	10	<u>IDFG1:</u> 2(2A1M)=2 <u>IDFG2:</u> (2A1M)=1	2 (2+0)
	(2,2)	13	<u>IDFG1:</u> 2(1A1M)=2 <u>IDFG2:</u> 1(1A1M)=1	2 (2+0)
<u>DCT-DIF:</u> 41 ops (29 add/subs, 12 mults) 2 IDFGs, CP=7 <u>IDFG1:</u> 24 ops, CP=7 <u>IDFG2:</u> 17 ops, CP=5	(2,4)	9	<u>IDFG1:</u> 2(2A2M)=2 <u>IDFG2:</u> (2A1M)=1	2 (2+0)
	(2,3)	10	<u>IDFG1:</u> 2(2A1M)=2 <u>IDFG2:</u> (2A1M)=1	2 (2+0)
	(2,2)	13	<u>IDFG1:</u> 2(1A1M)=2 <u>IDFG2:</u> 1(1A1M)=1	2 (2+0)
	(2,4)	9	<u>IDFG1:</u> (4A)+ (3A1M)+2(2A2M)=4	9
	(2,3)	10	<u>IDFG1:</u> 2(3A)+ 2(2A1M)+(1A1M)=5	11
	(2,2)	11	<u>IDFG1:</u> 3(2A)+4(1A1M)=7	16
<u>5th order WDE Filter:</u> 34 ops (26 add/subs, 8 mults) 1 IDFG, CP=14 <u>IDFG1:</u> 34 ops, CP=14	(2,4)	14	<u>IDFG1:</u> (2A2M)+(2A1M)=2	3
	(2,3)	15	<u>IDFG1:</u> (2A1M)+(1A1M)=2	2
	(2,2)	15	<u>IDFG1:</u> 3(1A1M)=3	7
	(2,4)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,3)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,2)	11	<u>IDFG1:</u> 2(1A1M)=2	4
<u>Auto Regression Filter:</u> 28 ops (12 add/subs, 16 mults) 1 IDFG, CP=8 <u>IDFG1:</u> 28 ops, CP=8	(2,4)	8	<u>IDFG1:</u> (1A2M)+(1A1M)=2	3
	(2,3)	8	<u>IDFG1:</u> (1A2M)+(1A1M)=2	3
	(2,2)	9	<u>IDFG1:</u> 2(1A1M)=2	2
	(2,4)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,3)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,2)	11	<u>IDFG1:</u> 2(1A1M)=2	4
<u>Avenhaus Filter:</u> 18 ops (8 add/subs, 10 mults) 1 IDFG, CP=7 <u>IDFG1:</u> 18 ops, CP=7	(2,4)	8	<u>IDFG1:</u> (1A2M)+(1A1M)=2	3
	(2,3)	8	<u>IDFG1:</u> (1A2M)+(1A1M)=2	3
	(2,2)	9	<u>IDFG1:</u> 2(1A1M)=2	2
	(2,4)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,3)	10	<u>IDFG1:</u> 2(1A2M)=2	4
	(2,2)	11	<u>IDFG1:</u> 2(1A1M)=2	4

Table 1. Experimental results.

latency penalty resulted from sub-problem 1, which was sufficient to allow a solution for the rest of the IDFG without further latency increases/penalties.

The results in Table 1 show that solutions derived for larger capacity cluster constraints may have the same latency as those for smaller capacity constraints, see e.g., DCT Lee and WDE filter for  $n = 2, 3$ . Note however that the solu-

tions associated with the larger capacity clusters have fewer clusters (with more FUs), and typically have fewer data transfers. This clearly shows the bias of our algorithm towards serialization – solutions that add extra steps by serializing operations inside a cluster are favored with respect to solutions scattering these operations through various (possibly smaller) clusters, and yet paying the same latency penalty due to data transfer delays. More concretely, the proposed algorithm is biased towards solutions that use fewer clusters of higher capacity, as opposed to using more clusters of capacity smaller than that specified in the constraint. The underlying rationale is that, when latency is identical, the first solutions will typically lead to fewer data transfers. Finally, note that for the Autoregression and Avenhouse filters, the solutions with  $m = 3, 4$  are identical. This indicates that the extra cluster capacity does not help with these particular kernels.

Other interesting observations could be drawn from our case studies. Consider for example the two alternative DCT algorithms (DIT and DIF) shown in the table. Although the DCT-DIT algorithm has roughly 20 % more operations than the DCT-DIF algorithm, it executes faster on a family with cluster capacity 2, and has identical latency for larger capacities. Such non-trivial observations can be useful when performing algorithmic exploration in the context of a given embedded application.

We conclude by briefly discussing how the proposed algorithm can be used to support trade-off exploration. Consider again the solution for the DIT DCT with a cluster capacity of 4. The “minimum” latency solution generated by the algorithm has 4 clusters and 9 steps, i.e., 2 steps in excess of the critical path. If the designer finds this number of clusters to be excessive, s/he can increase the target latency and execute the algorithm once more for the same capacity constraints. As the designer increases the initial target latency, the number of clusters in the solution will eventually reduce. Thus, the designer can explore trade-offs between latency and cluster area.<sup>8</sup> Delay/power vs. latency trade-offs can be explored by considering different capacity constraints. Indeed, as the constraint on cluster capacity increases, fewer steps are typically required to execute the same kernel, yet the register file local to each cluster will have more ports and be larger, and thus delay and power consumption will increase<sup>9</sup> [14]. Our proposed algorithm can thus play a key role in a design space exploration environment/framework.

## 5. Related Work

A significant body of work is available in the area of datapath synthesis for digital signal processing applications, see e.g., [3, 5]. Our focus is on approaches geared towards *high throughput* applications, e.g., [5, 2, 4, 15]. The use of hierarchy in the DFG and in the datapath is an important common characteristic

of such approaches. Below we briefly contrast our work with the Cathedral compilers developed at IMEC[5]— a representative example.

Cathedral uses an *Application Specific Unit* (ASU) based *architectural style* [5]. ASUs are datapaths whose composition in terms of functional building blocks (i.e., FUs) and interconnection structure is customized to parts of the application flow graph, i.e., to judiciously selected clusters of operations. Below we argue that the design space defined by the ASU-based architectural style is not compatible with the problem handled in this paper. Specifically, our VLIW datapath clusters are fundamentally different from ASUs, and thus so are the objectives driving the aggregation of operations to be executed on these hierarchical datapath components. In the ASU-based design style, the emphasis is on performance optimization via FU chaining [5]. Switching of data among the FUs internal to an ASU is done only through interconnect and MUXES. Thus, no resource sharing is allowed within an ASU. By contrast, in our clustered VLIW datapaths, a cluster’s local register file is used to switch data among the cluster’s FUs. Moreover, resource sharing within a cluster (while executing an aggregate) is not only possible, but highly desirable. Similar contrasts can be made to other high-level synthesis approaches, showing that their adopted *structural hierarchy* defines a design space incompatible with the problem addressed in this paper.

Retargetable code generation has received significant attention lately, see e.g., [9, 10]. As in the previous case, the algorithms developed for code generation solve optimization problems different from ours. During code generation, operation assignment to clusters (i.e., binding) and other code generation tasks are performed assuming a *specific* target datapath. In contrast, in our approach the binding of operations (aggregates) to clusters is performed for an “optimal” datapath that is being *simultaneously generated*. Deriving optimal code for a specified/target VLIW clustered datapath is a different problem from that of *efficiently* finding a VLIW clustered datapath that can deliver “maximum” performance, under the specified capacity constraints.

Several lower bounds on latency have been proposed, see e.g. [12, 16, 7]. Such work usually assumes a pre-defined datapath with a ‘*flat*’ organization of FUs[12, 16]. An exception is [7]. In this case, a lower bound on latency is computed for a DFG bound to a specific clustered VLIW datapath. By contrast, the objective of our algorithm is to estimate the minimum latency that can be achieved for a given DFG over a *family* of clustered VLIW datapaths defined by the specified capacity constraints. Thus, once again the problems are quite distinct.

## 6. Conclusions

We proposed an algorithm to support trade-off exploration during the early phases of the design of VLIW ASIPs with clustered datapaths. Encouraging experimental results obtained for a number of benchmark kernels, assuming various cluster capacities, show that our aggressive heuristic decomposition and pruning strategies work quite well in practice. We are currently working on incorporating high-level memory system design issues in our design space exploration framework – these will be considered prior to the exploration step discussed in this paper.

## Acknowledgments

This work is supported in part by an NSF CAREER Award MIP-9624321 and NSF Grant CCR-9901255 and by Grant ATP-003658-0649 of the Texas Higher Education Coordinating Board.

## Notes

1. ALAP( $o$ ) denotes the as-late-as-possible step of  $o$  for a given  $TL$ .
2. We also remove edges that traverse a number of scheduling steps exceeding the mobility of the activities that were extracted.
3. This measure of load accounts for the fact that activities with higher mobility have more flexibility in their scheduling ranges, and thus should have lower importance in terms of assessing compatibility among layers.
4. A *covering* is a collection of sets such that the union includes all the operations in the sub-IDFG and a *partition* is a cover of disjoint sets.
5. The time frame of an operation  $o$  is given by [ASAP( $o$ ), ALAP( $o$ )].
6. An unscheduled operation is said to be ready at step  $s$  if  $s$  is in its time frame.
7. Due the high complexity of the optimization problem being tackled, verifying the optimality of a solution with respect to latency is virtually impossible. However, in all cases we have determined by inspection that for the given capacity constraints the latency penalty steps could not be reduced.
8. Cluster area estimation is beyond the scope of this paper.
9. Delay and power estimation are beyond the scope of this paper.

## References

- [1] D.C. Burger and J.R. Goodman. Billion-transistor architectures. *IEEE Computer*, 30(9), 1997.
- [2] C.M.Chu and J.Rabaey. Hardware selection and clustering in the HYPER synthesis system. In *Proc. IEEE European Conference of Design Automation*, March 1992.
- [3] G. de Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc, 1994.
- [4] D.S.Rao and F.J.Kurdahi. Partitioning by regularity extraction. In *Proc. IEEE/ACM Design Automation Conference*, June 1992.
- [5] W. Geurts, F. Catthor, S. Vernalde, and H. DeMan. *Accelerator Data-Path Synthesis for High-Throughput Signal Processing Applications*. Kluwer Academic Publishers, 1997.

- [6] E. Ifeachor and B. Jervis. *Digital signal processing: A practical approach*. Addison-Wesley, 1993.
- [7] M. Jacome and G. de Veciana. Lower bound on latency for VLIW ASIPs. In *Proc. of ACM/IEEE International Conference on Computer Aided Design (ICCAD)*, pages 261–269, November 1999.
- [8] K.R.Rao and P.Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, 1990.
- [9] C. Liem. *Retargetable compilers for embedded core processors*. Kluwer Academic Publishers, 1997.
- [10] P. Marwedel and Gert Goossens, editors. *Code Generation for Embedded Processors*. Kluwer Academic Publishers, 1995.
- [11] NOVA Project: ASIPs and retargetable compilers; CAD for embedded systems. Department of ECE, U.T. Austin. <http://horizon.ece.utexas.edu/~jacome/nova/>.
- [12] M. Rim and R. Jain. Lower bound performance estimation for the high-level synthesis scheduling problem. *IEEE Trans. on CAD of ICs and Systems*, 13(4):451–58, 1994.
- [13] S. Rixner, W. Dally, U. Kapasi, B. Khailany, A. Lopez-Lagunas, P. Mattson, and J. Owens. A bandwidth-efficient architecture for media processing. In *Proc. 31st Annual International Symposium on Microarchitecture*, pages 3–13., Nov.-Dec. 1998.
- [14] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens. Register organization for media processing. In *Proc. 26th International Symposium on High-Performance Computer Architecture*, May 1999.
- [15] E. A. Rundensteiner, D. Gajski, and L. Bic. Component synthesis from functional descriptions. *IEEE Transactions on Computer Aided Design*, 12(9), 1993.
- [16] G. Tiruvuri and M. Chung. Estimation of lower bounds in scheduling algorithms for high-level synthesis. *ACM Trans. on DAES (TODAES)*, 3(2):162–80, 1998.

## Part III

# LOGIC SYNTHESIS

Logic Synthesis Overview <i>Robert K. Brayton and John A. Darringer</i>	181
Multiple-Level Logic Optimization System ( <i>ICCAD 1986</i> ) <i>R. Brayton, E. Detjens, N. Phillips, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung and A. Sangiovanni-Vincentelli</i>	191
Exact Minimization of Multiple-Valued Functions for PLA Optimization ( <i>ICCAD 1986</i> ) <i>R. Rudell and A. Sangiovanni-Vincentelli</i>	205
Improved Logic Optimization Using Global-Flow Analysis ( <i>ICCAD 1988</i> ) <i>C. Leonard Berman and Louise H. Trevillyan</i>	217
A Method for Concurrent Decomposition and Factorization of Boolean Expressions ( <i>ICCAD 1990</i> ) <i>Jagadeesh Vasudevamurthy and Janusz Rajski</i>	227
An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs ( <i>ICCAD 1992</i> ) <i>Jason Cong and Yuzheng Ding</i>	235
Logic Decomposition during Technology Mapping ( <i>ICCAD 1995</i> ) <i>Eric Lehman, Yosinori Watanabe, Joel Grodstein and Heather Harkness</i>	249

# LOGIC SYNTHESIS OVERVIEW

Robert K. Brayton

*Department of Electrical Engineering and Computer Science*

*University of California at Berkeley*

*Berkeley, CA 94720*

John A. Darringer

*IBM Research Division*

*Thomas J. Watson Research Center*

*Yorktown Heights, NY 10598*

## 1. Introduction

The dream of generating efficient logic implementations from higher level specifications originated with the works of Boole,[57] Shannon [58], Quine and McCluskey[35, 36]. Interest in logic synthesis grew during the 60's and 70's, as the computers being designed became more complex. Although many theoretical advances were made, the first examples of practical synthesis did not occur until the later 70's. Programmable logic arrays, PLAs, were minimized with the program, MINI, [15] and used on many product chips in IBM. LSS[10, 9] was the first example of production synthesis of gate array chips. It was based on local transformations and compiler techniques for optimizing logic, mapping gates to a specific technology. It was used on hundreds of product chips in IBM and was rewritten later with many significant refinements as BooleDozer[11]. These optimization methods were also used in the first offerings from Synopsys[12, 13], a company formed in 1986 (originally *Optimal Solutions*) to market synthesis technology developed at General Electric. Synopsys succeeded in bringing synthesis to the commercial market enabling a dramatic advance in design productivity. The years that followed saw many important developments that produced improvements in execution speed, quality of results and ability to deal with real technologies. Today, logic synthesis is a critical part of almost all chip development projects.

The sub-areas of logic synthesis are many-fold, motivated by the increasing challenges of technology advancements. These include, technology independent methods including combinational and sequential synthesis [45, 48]; two-level, multi-level and algebraic methods [2, 40]; redundancy and related ATPG methods [46]; technology mapping for area, delay, power [44], testability and FPGAs; delay analysis [28, 33, 34], performance optimization [30, 55], and de-

lay testing [53, 54]; incremental synthesis; don't cares and other flexibilities [42, 39, 47, 41]; asynchronous synthesis; and multi-valued synthesis [38, 43]. A key improvement in synthesis technology was the development of the ROBDD [27] and the ever improving BDD packages[32]. These have become the modern-day truth table, and have made some of the initial theoretical advances in the '60s and early '70s quite practical. A similar development seems to be happening with the development of more efficient SAT solvers [29, 31]. Both BDD and SAT solving have had a major impact on the way logic expressions are represented and analyzed, which is at the very basis of efficient logic synthesis methods.

Many of the advances in logic synthesis appeared in their initial form at ICCAD. Six of these papers have been selected for inclusion in this volume, based on the combined originality and excellence<sup>1</sup> of their main ideas. The selection was difficult, since there were many qualified papers, some of which are mentioned below. In most cases, significant advances occurred incrementally over the years, producing ever improving results, often finally displacing the original developments. The advance of technology and the companion growth in system complexity continue to demand further innovations in logic synthesis to provide the productivity improvements needed for designers to take full advantage of technology's promise.

We discuss the papers selected in chronological order and try to give them an historical perspective. All the papers have in common the fact that they introduced concepts that have been adopted and used in commercial synthesis programs.

## 2. Selected Papers

### 2.1 Espresso

The first paper [23] from ICCAD 1986 and is one of several papers on the ESPRESSO PLA minimization programs [1]. These programs are still used extensively both commercially and academically. This paper represents a significant extension of ESPRESSO to both exact minimization and multi-valued logic [22, 24]. ESPRESSO, first developed in the summer of 1982 at IBM [37], went through several iterations before becoming the version used today. Frequently, it is used internally in many multi-level logic synthesis programs as a subroutine. ESPRESSO provides both exact and heuristic two-level minimization. The extension described in the selected paper led to a wider set of applications (due to the multi-valued extensions) and improved run-time (due to improved data structures). The exact minimization procedure follows Quine-McCluskey, generating primes and solving a unate covering problem to obtain a minimum sum-of-products expression. Important innovations in each of these areas resulted in faster execution and the ability to be used on larger examples. During its development, ESPRESSO accumulated over 130 benchmarks for test-

ing. It was surprising at the time that ESPRESSO-EXACT could solve most of them in reasonable time, with only a handful that had too many primes to be solved. Interestingly, several years later, these hard test cases motivated several new research efforts [8, 21], using BDDs and new methods for generating useful primes, which ultimately solved the complete benchmark suite exactly.

## 2.2 MIS

The paper “Multiple-Level Logic Optimization System”, MIS, is also from the 1986 ICCAD [4], and describes a program development that grew out of the Yorktown Logic Editor, developed at IBM Yorktown Heights in the early 80’s. MIS was the first program that used algebraic methods [2] for fast multi-level logic decomposition. It was based entirely on algorithmic methods rather than local transformations. The paper outlines the overall flow and the basic algorithms for extraction, re-substitution, factoring, decomposition and simplification. Several commercial logic synthesis programs were based on improved versions of MIS [5] and a follow-on program SIS [25, 3] with extensions to sequential synthesis. Over the years, MIS/SIS, which is available in source code form, has served as a platform for logic synthesis development and for comparing various algorithms.

## 2.3 Global Flow

LSS’s initial success at IBM was based on extensive use of local transformations. Over time these suites of transforms were augmented or in some cases replaced by more systematic and comprehensive algorithms. “Improved Logic Optimization Using Global Flow Analysis” [6] is an excellent example of this progress. The paper describes improvements in the original global flow methods [51]. These methods are related to later works on don’t cares [49, 52], redundancy addition and removal [46, 50], and recursive learning [47]. Here data-flow methods are employed to compute a “controlling” relation between signal pairs and then this summary data is used to enable a diverse set of optimizations. This paper describes significant improvements resulting in better logic, quicker runtime, an improved ability to handle don’t cares, and a method of reducing connections, often a critical issue in real chips.

## 2.4 Concurrent Optimization

“A Method for Concurrent Decomposition and Factorization of Boolean Expressions” (ICCAD 1990) [26] introduced a new algebraic technique. It focused only on “two-cube divisors” and provided an implementation where both single cube and two-cube divisors could be simultaneously evaluated for choosing the best one. This allowed for selecting a sequence of division steps which interleaved the use of single and multiple cube divisors. In addition, some divisors

could be evaluated in combination with their complements, when the complement was a single or two cube expression, further improving the result. The paper also showed how to incrementally update the data structure after a divisor was chosen, resulting in a very fast implementation. One advantage of this approach is that there can be no more than a quadratic (in the number of cubes in the expression) number of such divisors, whereas in the use of kernels, an exponential number is possible. These methods, were shown to be competitive in quality with the use of kernels, but are much faster in many cases. They have been implemented in MIS and SIS and are the basis for their extraction procedures, which find common algebraic divisors.

## 2.5 FPGA Synthesis

“An Optimal Technology Mapping Algorithm for Delay Optimization in Look-up-Table Based FPGA Designs” (ICCAD 1992) [7] represents an important contribution in the area of FPGA synthesis, which will likely have increasing relevance in the future. The work shows how to map logic to an FPGA where the depth of the network (and hence delay) can be minimized exactly, not just for trees, but for more general DAGs, in polynomial time. In contrast, technology mapping<sup>2</sup> for area for DAGs is NP-hard [17]. This paper, as well as the next one, inspired a later work [19], which showed, surprisingly, that technology mapping for optimum delay for DAGs, like tree mapping, can be done in linear time, (assuming that the delay is independent of the fanout load).

## 2.6 Technology Mapping

“Logic Decomposition During Technology Mapping” (ICCAD 1995) [20] describes a method for combining technology mapping and algebraic decomposition. The first breakthrough in technology mapping originated in the classical paper by Keutzer at Bell Labs[18], where the problem was shown to be similar to compiler optimization [14] and could be decomposed into tree mapping, providing a linear-time algorithm. For many years this has been the method of choice, with later developments focusing on optimization for delay and power, as well as area. However, the step before technology mapping required that the logic be decomposed into generic basis functions such as 2-input NAND gates. This biased the final result because it required a priori decisions on how to decompose the logic functions algebraically. The paper by Lehman et. al. solved this by defining a new structure, a mapping graph, which can represent all possible algebraic decompositions. It was made efficient by identifying sub-structures that represented the same logic, similar to BDDs. The final mapping used dynamic programming, visiting each node and finding a best match, similar to what is done in tree mapping. Later developments [16] improved on

area-delay tradeoffs and provided more details for an efficient implementation. The method was used in minimizing delay in DEC alpha machines. A detailed version of the work appeared in IEEE Trans. CAD 1997 [56] and received the best paper award for that year.

### 3. Additional ICCAD Contributions

The last twenty years of ICCAD included many important papers beyond the six included in this book. A select few of these are listed in chronological order along with a brief comment on each. The set represents solutions to a broad range of different problems arising in logic synthesis; two present elegant solutions for sequential optimization; however, such methods are rarely, if ever, used in practice, because of their impact on verification and other parts of a design methodology.

- *KISS: A program for Optimal State Assignment for FSMs* G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli 1984. This paper described a state assignment method for two-level implementation of finite state machines using multi-valued optimization and embedding of face constraints generated by the optimization result.
- *Performance-Oriented Synthesis in the Yorktown Silicon Compiler*, G. De Micheli 1986. This paper outlines a method for the overall optimization of delay in a combinational circuit by a sequence of re-synthesis, re-positioning and re-sizing steps as implemented in the YSC. This was probably the first attempt at combining logic synthesis and physical design optimizations, a technique essential for today's technologies and present in most commercial synthesis systems.
- *Multi-level Logic Optimization and the Rectangle Covering Problem*, R. Brayton, Rick Rudell, A. Sangiovanni-Vincentelli, and A. Wang 1987. This paper gave an elegant solution for extracting best kernels from a list of all kernels generated from a set of logic functions found in a Boolean network. The solution is stated in terms of finding rectangles in an incidence matrix of cubes versus kernels.
- *Observability Relations and Observability Don't Cares*, R. Brayton and H. Savoj 1991. This paper extended the theory of observability don't cares to multi-output nodes in a Boolean network and gave a method for computing the maximum flexibility for such nodes in terms of Boolean relations.

- *Calculating Resetability and Reset Sequences*, Carl Pixley 1991. This paper solved the problem of computing a reset sequence for a finite state machine after power up, when the machine could be in any state. It gives the condition when a machine is resettable and if it is, a method for computing a reset sequence.
- *Analysis of Cyclic Combinational Circuits*, Sharad Malik 1993. This paper re-examines the property of a circuit being *combinational* and analyzes cyclic circuits for this property using a method based on ternary simulation.
- *Switching Activity Analysis Considering Spatiotemporal Correlations*, R. Marculescu, D. Marculescu, and M. Pedram 1994. This paper looks at the problem of estimating power dissipation in a circuit and provides a method of computing switching activity using Markov chains to model both temporal correlations in a single signal and correlations between pairs of signals.
- *Logic Optimization by Output Phase Assignment in Dynamic Logic Synthesis*, R. Puri, A. Bjorksten, and T. Rosser 1996. This paper addresses the logic duplication problem encountered in dynamic logic. It solves, exactly and heuristically, the problem of minimizing the duplication required by assigning output phases to the logic nodes. This method has been used by many companies for dynamic logic design.
- *A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications*, S. Yamashita, H. Sawada, A. Nagoya 1996. This paper describes a new method, originally applied to LUT-type FPGAs, for computing flexibility in circuits. This flexibility, called SPFDS, is fundamentally different from don't cares and potentially provides more flexibility in implementing circuits.

## Notes

1. Like Samuel Johnson “Your manuscript is both good and original, but the part that is good is not original, and the part that is original is not good”, these qualities were required to be non-disjoint.
2. Technology mapping is the problem of mapping a net-list of logic functions into a given library of specific gates.

## References

- [1] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [2] R. K. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. *Proceedings of the International Symposium on Circuits and Systems*, May 1982.

- [3] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni- Vincentelli. Multilevel Logic Sythesis. *Proceedings of the IEEE*, vol.78, pages 264-300, Feb. 1990.
- [4] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeen, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, A. Sangiovanni- Vincentelli. Multiple-level logic optimization system *ICCAD 1986*, pp 356-359.
- [5] R. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Transactions on Computer-Aided Design*, pages 1062-1081, November 1987.
- [6] L. Berman and L. Trevillyan. Improved Logic Optimization Using Global Flow Analysis Extended Abstract. *ICCAD 1988*.
- [7] Jason Cong, Yuzheng Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *ICCAD 1992*, pages 48-53.
- [8] O. Coudert, H. Fraisse and J. C. Madre. Towards a symbolic logic minimization algorithm. *The Proceedings of the VLSI Design 1993 Conference*, pages 329-334, Jan. 1993.
- [9] John A. Darringer, Daniel Brand, John V. Gerbi, William H. Joyner Jr., Louise Trevillyan. LSS: A System for Production Logic Synthesis. *IBM Journal of Research and Development*, pages 537-545, Sept. 1984.
- [10] J. A. Darringer, J. W. Joyner, C. Berman, L. Trevillyan. Logic Syntesis Through Local Transformations. *IBM Journal of Research and Development*, Vol. 25, no.4., July 1981.
- [11] L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler. BooleDozer: Logic Synthesis for ASICs. *IBM Journal of Research and Development*, vol. 40, no. 4, pp. 407-430, July 1996.
- [12] A. J. deGeus and W. Cohen. A Rule-Based System for Optimizing Combinational Logic. *IEEE Design and Test*, pages 22-32 August 1985.
- [13] D. Gregory, K. Bartlett, A. J. deGeus. Automatic Generation of Combinatorial Logic from a Functional Specification. *Proc. 1985 Int. Symp. on Circ. and Syst.*, Kyoto, Japan, June 1985.
- [14] C. M. Hoffman and M. J. O'Donnell. Pattern Matching in Trees. *Journal of the Association for Computing Machinery*, pages 68-95, January 1982.
- [15] S. J. Hong, R. G. Cain, D. L. Ostapko. MINI: A Heuristic Approach for Logic Minimization. *IBM Journal of Research and Development*, pages 443-458, 1974.
- [16] D. Jongeneel, R. Otten, Y. Watanabe, R. K. Brayton. Area and Search Space Control for Technology Mapping. *Proceedings 2000 Design Automation Conference 37th DAC*, pages 86-91, June 2000.
- [17] K. Keutzer and D. Richards. Computational complexity of logic synthesis and optimization. *Proceedings of International Workshop on Logic Synthesis*, May 1989.
- [18] K. Keutzer. Technology binding and local optimization by dag mapping. *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 341-347, June 1987.
- [19] Y. Kukimoto, R. K. Brayton, P. Sawkar. Delay-optimal technology mapping by dag covering. *Proceedings of the 35th ACM/IEEE Design Automation Conference*, June 1998.
- [20] E. Lehman, Y. Watanabe, J. Grodstein, H. Harkness. Logic Decomposition during Technology Mapping. *ICCAD 1995*.
- [21] P. McGeer, J. Sanghavi, R. Brayton and A. Sangiovanni- Vincentelli. Espresso-signature: a new exact minimizer for logic functions. *Transactions of VLSI*, pages 432-440, Dec. 1993.

- [22] R. Rudell and A. Sangiovanni-Vincentelli. Espresso-MV: Algorithms for multiple-valued logic minimization. *Proceedings of the IEEE 1985 Custom Integrated Circuits Conference*, pages 230-4, May, 1985.
- [23] R. Rudell, and A. L. Sangiovanni-Vincentelli. Exact Minimization of Multiple-Valued Functions. *ICCAD 1986*, pp. 352-355
- [24] R. L. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transaction on CAD*, 1988.
- [25] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. *Technical Report UCB/ERL M92/41*, Electronics Research Laboratory, University of California, Berkeley, CA 94720, May 1992.
- [26] J. Vasudevamurthy and J. Rajski. A Method for Concurrent Decomposition and Factorization of Boolean Expressions *ICCAD 1990*.
- [27] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computing*, pages 677-691, Aug. 1986.
- [28] P. McGeer and R. K. Brayton, Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network, Circuits and Systems, *Design Automation Conference*, June 1989.
- [29] J. P. Marques-Silva and K. A. Sakallah. GRASP: A New Search Algorithm for Satisfiability, *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design* pp. 220-227, November 1996.
- [30] G. De Micheli, Performance-Oriented Synthesis in the Yorktown Silicon Compiler, *ICCAD*, pages 138-141 Nov. 1986.
- [31] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver, *Proc. of the Design Automation Conference*, pp. 530-535, June 2001.
- [32] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. *Design Automation Conf.*, pages 40-45, June 1990.
- [33] H.-C. Chen and D. Du. Path Sensitization in Critical Path Problem. *Proceedings of Int'l Conference on Computer-Aided Design*, pages 208-211, November 1991.
- [34] S. Devadas, K. Keutzer, and S. Malik. Delay computation for combinational logic circuits: theory and algorithms *IEEE International Conference on Computer-Aided Design*, Nov. 1991.
- [35] W. V. Quine. The problem of simplifying truth functions, *American Mathematical Monthly*, vol. 59 no. 8, pp. 521-531, Oct. 1952.
- [36] E. J. McCluskey, Jr., Minimization of Boolean Functions, *Bell System Tech. J.*, vol. 35 no. 6, pp. 1417-1444, Nov. 1956.
- [37] R. K. Brayton, G. D. Hachtel, L. Hemachandra, R. Newton and A. Sangiovanni- Vincentelli , A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization, *Proceedings of the International Symposium on Circuits and Systems*, pages 42-48, Rome Italy, April 1982.
- [38] L. Lavagno, S. Malik, R. K. Brayton and A. Sangiovanni-Vincentelli, MIS-MV: Optimization of Multi-Level Logic with Multiple-valued Inputs, *Proc. of ICCAD*, pp. 560-563, November 1990.
- [39] H. Savoj and R. K. Brayton, Observability Relations and Observability Don't Cares, *ICCAD*, November 1991.

- [40] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, Multi-Level Logic Optimization and the Rectangle Covering Problem *ICCAD* Nov. 1987
- [41] S. Yamashita, H. Sawada, and A. Nagoya A New Method to Express functional Permissibilities for LUT based FPGAs and Its Applications *ICCAD* Nov. 1996
- [42] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, The Transduction Method - Design of Logic Networks based on Permissible Functions *IEEE Trans. on Comp.*, Oct., 1989.
- [43] M. Gao, J.-H. Jiang, Y. Jiang, Y. Li, S. Sinha, and R. Brayton, MVSIS, *International Workshop on Logic Synthesis* June 2001
- [44] R. Marculescu, D. Marculescu, and M. Pedram Switching Activity Analysis Considering Spatiotemporal Correlations *ICCAD* Nov. 1994.
- [45] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli. KISS: A Program for Optimal State Assignment for FSMs, *ICCAD* Nov. 1984.
- [46] K.-T. Cheng and Luis A. Entrena, Multi-Level Logic Optimization by Redundancy Addition and Removal, *Proc. European Conf. on Design Automation*, pages 373-377, Feb. 1993.
- [47] W. Kunz and D. K. Pradhan, Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation Digital Circuits *Proc. Int'l Test Conference* pages 816-825, October 1992.
- [48] Carl Pixley Calculating Resetability and Reset Sequences *ICCAD* Nov. 1991
- [49] E. M. Sentovich and R. K. Brayton Don't Cares and Global Flow Analysis of Boolean Networks *ICCAD*, Nov. 1988
- [50] Chih-Wei Chang and Małgorzata Marek-Sadowska. Single-Pass Redundancy Addition and Removal, *ICCAD*, pages 606-609, Nov. 2001
- [51] L. Trevillyan, W. Joyner and L. Berman Global Flow Analysis in Automatic Logic Design *IEEE Trans. on Comp.* Jan. 1986
- [52] D. Brand Redundancy and Don't Cares in Logic Synthesis *IEEE Trans. on Comp.* Oct. 1983
- [53] Kwang-Ting Cheng, Srinivas Devadas and Kurt Keutzer. Robust Delay-Fault Test Generation and Synthesis for Testability Under A Standard Scan Design Methodology. *DAC*, pages 80-86, June 1991: 80-86
- [54] Alexander Saldanha, Robert K. Brayton and Alberto L. Sangiovanni-Vincentelli. Equivalence of Robust Delay-Fault and Single Stuck-Fault Test Generation. *DAC*, pages 173-176, June 1992
- [55] Kurt Keutzer, Sharad Malik and Alexander Saldanha. Is Redundancy Necessary to Reduce Delay. *DAC*, pages 228-234, June 1990
- [56] E. Lehman, Y. Watanabe, J. Grodstein and H. Harkness, Logic Decomposition During Technology Mapping, *IEEE Transactions on CAD/ICAS*, Vol. 16, No. 8, pp. 813-834, August 1997
- [57] G. Boole. An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities *London: Walton and Moberly*. 1854
- [58] Shannon, C.E. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers* pages 713-723, 1938

# MULTIPLE-LEVEL LOGIC OPTIMIZATION SYSTEM

R. Brayton

*IBM Research*

*Yorktown, NY*

E. Detjens

*Philips Research Labs*

*Sunnyvale, CA*

N. Phillips

*Digital Equipment Corp.*

*Hudson, MA*

S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal,  
A. Wang, R. Yung and A. Sangiovanni-Vincentelli

*University of California*

*Berkeley, CA*

## Abstract

MIS is a multi-level logic synthesis and minimization system and is an integral part of the Berkeley Synthesis Project. MIS starts from a description of a combinational logic macro-cell and produces an optimized set of logic equations which preserves the input-output behavior of the macro-cell. The system includes algorithms for minimizing the area required to implement the logic equations, and a global timing optimization step which is used to change the form of the logic equations along the critical path in order to meet system-level timing constraints. This paper provides an overview of the optimization system including the input language, the algorithms which minimize the area of the implementation, and the algorithms used to re-structure the logic network to meet the system-level timing constraints. Although the system is still under development, pieces of an industrially designed chip have been re-synthesized with MIS and the results compare favorably with the manual designs.

## 1. Introduction

MIS is a multi-level logic synthesis and minimization system and is an integral part of the Berkeley Synthesis Project. The program BDSYN starts from a high-level description of a combinational logic macro-cell, and extracts a set of logic equations. MIS optimizes the logic equations and produces a logic network which preserves the input-output behavior of the macro-cell. Communication between BDSYN and MIS, and between MIS and the module generation tools is through the OCT database [6]. Module generators which synthesize a symbolic

layout for a macro-cell directly from an optimized logic network are also part of the Berkeley Synthesis Project.

The optimization criterion is to minimize the area occupied by the logic equations (which is measured as a function of the number of gates, transistors, and nets in the final set of equations) while simultaneously satisfying the timing constraints derived from a system-level analysis of the chip. Timing constraints are also passed to the module generator tools to guide the placement and routing of the gates within a macro-cell, and similar constraints are passed to the floor-planning and placement and routing tools to guide the placement and routing of the macro-cells.

In this paper, we briefly present the major components of the multiple-level logic optimization system. These include the input language, the global optimization strategy for area minimization under timing constraints, and the local optimization step for mapping the resulting set of logic equations into an implementation. The system first minimizes the area without concern for the delay. Then MIS enters a timing pre-optimization step where the critical paths of the entire system are derived and the logic equations are re-structured along a cut-set of the critical paths to trade-off area for speed to meet the system timing requirements. This re-structuring consists of collapsing logic functions to fewer levels and duplicating logic functions. Final timing optimization is done by sizing transistors, and by passing constraints to the physical design tools to influence the placement and routing of the critical paths.

The system and the algorithms are general and hence largely independent of technology. Currently, MIS is targeted for complex-gate static CMOS designs. However, we have also considered the problems of fixed-gate technologies (i.e., fixed library cells) and dynamic CMOS design, and are confident that the same algorithms can be used for these other design styles.

## 2. Specification of the Design

MIS starts from a net-list description of a combinational logic network, where each block (i.e., *gate*) in the network is a multiple-input, single-output Boolean function. A network of this form is called a *Boolean Network*. A BLIF (Berkeley Logic Interchange Format) file is a textual representation of a Boolean network and can be used to communicate with tools outside the Oct environment.

As part of the Berkeley Synthesis Project, we have developed a program which translates a high-level description of a piece of logic into a Boolean Network. The designer partitions a digital design into combinational blocks and the latches (or bus transceivers) which connect the logic blocks. The combinational blocks are described by programs written in BDSYN, which is a subset of the hardware description language BDS. (BDS is the behavioral language used as part of Digital Equipment Corporation's mixed-mode simulation system.) The

latches and buses are connected to the logic blocks in a net-list. This net-list is the starting point for the floor-planning and placement and routing tools in the Berkeley Synthesis Project.

BDS is a programming language with the built-in data type of a bit-vector and the basic operations on bit-vectors (i.e., bit selection, logical operations, shift and rotate operations, and arithmetic operations). A program written in BDSYN has a well-defined set of inputs and outputs, and is assigned the semantics that the output is computed as a combinational function of the inputs. An arbitrary BDS description is allowed (including functions, loops, global variables, and multiple-assignment to variables). The program BDSYN extracts from this description a Boolean Network (i.e., a set of logic equations) with input/output behavior equivalent to the BDSYN description. It is important to note that these equations consist of many levels of logic thus removing any limitation on the type of logic described. We are currently investigating extensions to BDSYN which will allow the designer to explicitly specify don't-care conditions in the logic network. Our experience shows that allowing the designer to specify this information to the system can dramatically improve the optimization of the logic.

Figure 1 shows the BDSYN description of the register file decoder from the SPUR microprocessor [5]. The register file decoder is complicated by the use of the overlapping window scheme [7]. *cwp* is the current window pointer for the register file, and *reg* is the index of the register within the window. A one-hot signal *addr* selects one of the 138 registers in the register file. The operator & is bit concatenation, the operator ZXT performs an extension of an operand with zero fill, and the operator SL0 performs a shift-left with zero fill. Note the use of complex operations such as addition, comparison, and variable shift left (all of which are supported in BDSYN). Also, the variable *address* is assigned to more than once.

### 3. MIS

MIS is set up as an interactive logic synthesis system to allow us to experiment with different algorithms. MIS can also be run in a batch mode, as an automatic synthesis system would require, by a *script* file of MIS commands. The list of MIS commands is quite extensive and continues to grow as more development and experimentation is done. Currently, commands in MIS are concerned with eliminating nodes, simplifying nodes, substituting one node into another, factoring nodes, breaking nodes into smaller components, finding common kernels and common cubes, reading and writing logic files, factoring and complementing the functions, and displaying information about the Boolean Network. Many of these operations are broken down into different versions with varying efficiency. A typical MIS script file that may be used for automatic synthesis is

```

MODEL decoder addr<137:0> = cwp<2:0>, reg<4:0>;
BEHAVIOR;
CONSTANT NUMREGS = 138, NUMGLOBALS = 10;
ROUTINE main;
    STATE address<7:0>;
    ! Check for reference to global register
    if reg leq 9 then
        address = reg
    else begin
        ! compute address, check overflow
        address = (cwp & 0000#2) + reg;
        if address gtr (NUMREGS - 1) then
            address = address - (NUMREGS - NUMGLOBALS);
    end;
    ! Create the one-hot decode based on the address
    addr = (ZXT {width=138} 1) SL0 address;
ENDROUTINE main;
ENDBEHAVIOR;
ENDMODEL decoder;

```

*Figure 1.* BDSYN Example.

shown in Figure 2. A partial list of MIS commands and a short explanation of each is shown in Figure 3. Some of these operations are explained further below.

By changing the script file and changing some of the parameters of the operators, the file can be tailored to the kind of logic at hand. However, we need more experience to determine how to do this automatically.

### 3.1 Global Area Optimization

The goal of this step is to reduce the complexity of the logic equations using global techniques which allow significant re-structuring of the network, and which are independent of the particular design style or technology. Each node in the Boolean network is a completely specified Boolean function, and is stored in sum-of-products form. We use as a measure of the complexity of a logic equation the number of literals required to represent the function in a factored form. The cost for a Boolean network is the sum of the costs for each node. For example, the function

$$\begin{aligned}
 f_1 = & ab\bar{g} + abfg + ab\bar{e}\bar{g} + ace\bar{g} + acfg + ac\bar{e}\bar{g} + \\
 & d\bar{e}\bar{g} + dfg + d\bar{e}\bar{g} + bh + bi + ch + ci
 \end{aligned}$$

can be written in factored form using 13 literals:

$$f_1 = (a(b+c) + d)(e\bar{g} + g(f+\bar{e})) + (b+c)(h+i).$$

As will be mentioned later, there are several different algorithms which can be used to factor a single logic equation, each with a different performance and quality tradeoff. Because the cost function must be evaluated frequently during the synthesis, we favor a fast factoring algorithm during the early stages of the synthesis.

To explain the reasoning behind the cost function, consider that  $f_1$  may be implemented as a single CMOS complex-gate using 13 transistor pairs. Or, because of various technology reasons, the gate might require further decomposition into smaller gates. However, when this function is broken into smaller gates, it can be broken down in a manner corresponding to its factored form; hence, the transistor count does not increase greatly. The total number of literals in the factored form is a good measure of the number of transistors in the network.

Now assume that  $f_1$  was identified as a common factor to several other functions, and has been created to reduce the total complexity of the network. Regardless of how  $f_1$  is implemented (either as a single gate, or requiring further decomposition), it has reduced the number of transistors in the network by an amount corresponding to the total number of transistors that have been saved in the functions which it feeds minus the number of transistors in  $f_1$ .

**3.1.1 Identifying Global Commonality.** The most important (and most difficult) step in global minimization is to identify factors common to two or more functions which can be used to reduce the total number of literals in the network. To accomplish this, we use two basic operations: generate common algebraic factors from a set of logic functions (*extraction*), and checking whether an existing function is a factor of some other function (*resubstitution*).

**3.1.2 Extraction.** Based on the notion of a kernel [2], the set of all useful common algebraic divisors of a set of equations consists of all kernels and their intersections, and common single cubes. We thus limit our search space to first finding common kernels and common intersections of kernels, and then common single cube divisors. We can demonstrate that these two operations are computationally equivalent and can be formulated as the problem of finding a minimum rectangle cover of a 0-1 matrix.

For example, let us consider the problem of finding common single cube divisors. Suppose we have a number of functions  $f_1, \dots, f_n$  which are each described as a sum of products. Then we can construct a 0-1 matrix,  $B$ , such that row  $i$  represents the  $i^{th}$  product, and column  $j$  represents the  $j^{th}$  literal. A 1 is inserted in element  $(i, j)$  if literal  $j$  is in product  $i$ . A rectangle is defined as a subset of rows  $R$  and columns  $C$  such that for all  $(i, j)$  where  $i \in R$  and  $j \in C$  we have

$B_{ij} = 1$ . Then a rectangle represents a common subproduct which can be found in each of the products associated with the subset of rows  $R$ . An optimal rectangle covering of  $B$  is then an optimal set of subproducts which can be extracted and implemented as separate functions, and each of these used to simplify the functions  $f_j$ .

This covering problem has analogies to the problem of Boolean minimization. For example, we can define *prime rectangles*, and can solve the problem exactly using an algorithm very similar to the Quine-McCluskey algorithm for Boolean minimization. Because of the computational complexity of this exact solution to this problem, we are investigating adapting iterative, heuristic Boolean minimization algorithms (such as ESPRESSO-II [4]) to this problem.

**3.1.3 Resubstitution.** Once the factors have been determined, *algebraic resubstitution* is used to simplify the network. For example, suppose the network is:

$$\begin{aligned} f_1 &= ac + ad + bc + bd + e \\ f_2 &= a + b. \end{aligned}$$

Function  $f_1$  can be rewritten as:

$$f_1 = f_2(c + d) + e.$$

This operation is called algebraic resubstitution since  $a + b$  is an algebraic divisor of  $ac + ad + bc + bd + e = (a + b)(c + d) + e$ .

However, algebraic techniques alone do not exploit all of the Boolean properties of the logic equations. To improve the results, we also perform Boolean resubstitution, which is to check if any of the existing functions is a divisor of other functions in the network in the Boolean sense. Boolean resubstitution is capable of providing better results, but in general takes much longer than algebraic resubstitution. For example, algebraic resubstitution does not simplify the following network:

$$\begin{aligned} f_1 &= (ab + cd)\bar{e}\bar{f} + (ab + ef)\bar{c}\bar{d} + (cd + ef)\bar{a}\bar{b} \\ f_2 &= ab + cd + ef \end{aligned}$$

However, using Boolean resubstitution,  $f_1$  can be rewritten as

$$f_1 = f_2(\bar{a}\bar{b} + \bar{c}\bar{d} + \bar{e}\bar{f})$$

for a savings of 11 literals.

It is known that algebraic resubstitution can be performed in time  $O((|f| + |g|)\log(|f| + |g|))$  when the functions  $f$  and  $g$  have  $|f|$  and  $|g|$  product terms respectively [2]. However, Boolean resubstitution involves a Boolean minimization step with a proper don't-care set. Hence, heuristics are needed to decide when to use Boolean resubstitution in order to reduce the execution time.

**3.1.4 Phase Assignment.** Each function is represented as a positive logic, factored expression. Also, each intermediate variable is assumed present in both its true and complemented form. In practice, logic gates in static CMOS are inverting, and not all intermediate variables are needed in both phases. Hence, there is an optimization problem of choosing the phase of each intermediate function to minimize the number of inverters needed to implement the network. Global phase assignment is performed which determines for each function whether to implement the function or its complement. We are experimenting with several heuristic solutions to this problem.

## 3.2 Local Optimization

Local optimizations refer to operations performed on a single logic function, or locally in the surrounding neighborhood of the logic function. Our synthesis system uses the local transformations of: decomposing large gates into smaller ones, deriving better implementations of the gates, and simplifying each gate using knowledge of its local environment.

**3.2.1 Local Decomposition of a Gate.** We use two different algorithms to decompose a gate into a set of smaller gates. *Good algebraic decomposition* uses a greedy strategy based on selecting the best kernel and pulling out this kernel. *Quick decomposition* is done by generating only a single level-0 kernel at each step. *Quick decomposition* is computationally less expensive than generating all kernels and their intersections (in some cases, it is as much as 5 times faster) and has been shown to produce good results (often identical results). We will illustrate decomposition by referring to the examples given below for factoring, since decomposition and factoring are very similar operations.

**3.2.2 Factoring a Gate.** We are currently targeting a complex-gate CMOS design style. Hence, another local optimization is to factor the logic equation of a single gate in order to produce an optimal pull-down (and pull-up network) for the gate. Different factoring algorithms have been explored. Each has its own use and run time cost. *Quick factoring* (a modification of *Quick Decomposition*) is useful to estimate the cost of an implementation. The number of transistors in a quickly factored form gives a good estimate of the number of literals in a complex-gate CMOS implementation and the cost function is evaluated frequently during the synthesis. Two other factoring algorithms are also implemented called *good algebraic factoring* and *Boolean factoring*. Even though they are relatively expensive operations, they occasionally yield better factorizations, and are used to derive the final implementations of the gates.

To illustrate the different results that can be obtained by these various factoring algorithms, consider the function

$$f_1 = ac + ad + ae + ag + bc + bd + be + bf + ce + cf + df + dg.$$

Quick factoring (QF) and good factoring (GF) give different but similar quality factorings,

$$\begin{aligned} GF(f_1) &= (c+d+e)(a+b) + f(b+c+d) + g(a+d) + ce \\ GF(f_1) &= g(a+d) + (a+b)(c+d+e) + c(e+f) + f(b+d) \end{aligned}$$

but QF is much faster because it only needs to determine one level 0 kernel for each factor. However, for

$$\begin{aligned} f_1 &= ab\bar{g} + abfg + ab\bar{e}g + ace\bar{g} + acfg + ac\bar{e}g + \\ &\quad d\bar{e}g + dfg + d\bar{e}g + bh + bi + ch + ci \end{aligned}$$

the factored results are:

$$\begin{aligned} QF(f_1) &= (a(g(\bar{e}+f) + e\bar{g}) + i + h)(b+c) + d(g(\bar{e}+f) + e\bar{g}) \\ GF(f_1) &= (a(b+c) + d)(\bar{e}g + g(f+\bar{e})) + (b+c)(h+i). \end{aligned}$$

In this example, GF obtains the better result (13 literals versus 16 literals) by working harder to find the best kernel at the top level of the recursion. To illustrate Boolean factoring (BF), consider

$$f_1 = a\bar{b} + a\bar{c} + a\bar{d} + \bar{a}b + b\bar{c} + b\bar{d} + \bar{a}c + \bar{b}c + c\bar{d} + \bar{a}d + \bar{b}d + \bar{c}d.$$

We obtain

$$BF(f_1) = (a+b+c+d)(\bar{a}+\bar{b}+\bar{c}+\bar{d}),$$

whereas

$$QF(f_1) = GF(f_1) = \bar{a}(b+c+d) + (a+b)(\bar{c}+\bar{d}) + c(\bar{b}+\bar{d}) + \bar{c}d.$$

QF has become an important tool since it is so fast and effective. QF is used continuously during synthesis to estimate area and delay.

**3.2.3 Simplification.** We employ three forms of *simplification* another powerful local transformation. This step minimizes the function of a single equation using the algorithms of ESPRESSO-II [4]. Depending on the stage of the optimization, minimization algorithms of differing cost/performance trade-offs are used. (For example, quick simplification [4] (Chapter 3) is used at the beginning of the synthesis. Later, towards the end of the synthesis, the full power of

the ESPRESSO-II minimization algorithm is used with a don't-care set derived from the environment of the function [3]. For example, for the Boolean network

$$\begin{aligned} u &= \bar{b} + y + z \\ v &= z + \bar{a} \\ z &= yad \\ y &= b\bar{c} \end{aligned}$$

with outputs  $u, v$  and intermediate nodes  $y, z$ , if we simplify function  $z$ , we can write down a don't-care set which takes into account the environment of node  $z$  embedded in the Boolean network:

$$DC = y(\bar{b} + c) + \bar{y}b\bar{c} + (\bar{b} + y)\bar{a}.$$

Then using DC to simplify the function  $yad$  we obtain  $yd$ . A general method for simplifying a node in a Boolean network using two-level minimization and an appropriate don't-care set is given in [1].

### 3.3 Timing Pre-Optimization

To support timing optimization, we have a set of routines to break large gates into smaller ones, or to combine small gates into larger ones, in order to reduce the delay.

Our first problem is to estimate the delay of the network. Our goal is a reasonably accurate, relative measure of the speed of the circuit in terms of the number of gates, number of fanouts, and the size of the gates along a critical path. We took the following approach to estimate the delay through each gate. First, for each input, we translate a gate of arbitrary series-parallel transistors into an equivalent n-inputs NAND gate. A NAND gate is characterized by the number of inputs, transistor widths, and load capacitance. The delay is modeled with a polynomial function of these parameters. The coefficients of the equation are determined by the least square fitting of the curve to a large set of SPICE results for various NAND gates. Two sets of coefficients are used for CMOS circuits, one for the P-type transistors and one for the N-type. For each input to a gate, we compute the delay caused by the gate assuming that the given input is critical. This is done for both the pull-up phase and the pull-down phase. The critical paths are then traced through the appropriate input to output delays, alternating between gates from pull-up to pull-down and vice versa. By using this mechanism, we obtain better delay estimates and eliminate many false paths in the critical delay calculations.

The optimization loop involves, given a set of timing constraints, computing delays through each gate, identifying all the critical paths, finding a minimum weighted node cut set of the critical paths, and the re-synthesis of these nodes.

The weight of a node in the critical path is a measure of the number of literals on the critical path which are saved by having the node present. Finding a minimum weighted node cut set is equivalent of finding the maximum-flow/minimum-cut in a flow network. Because of the computational efficiency of the routines to compute delays and the cut-sets, we are able to iterate over these operations many times until the timing constraints are satisfied, or until it is apparent that they cannot be satisfied.

Because our timing estimates are necessarily inaccurate, we view the result of the optimization not as a precise delay calculation, but rather as a reasonable guide for re-structuring the architecture of the network to meet the timing constraints. Also, the timing estimates are reasonable specifications for the module generators to use when synthesizing and placing the gates. More accurate timing estimates and verifications may be employed later in the design cycle when the details of the gate designs and placements are known more precisely.

## 4. Results

Although work is continuing on MIS, Jacob Thomas of Advanced Micro Devices became interested in testing the performance of MIS on some industrial circuits. The function of each of these circuits was taken from an actual chip design, and had previously been designed by a human designer. In each case, the logic network was designed starting from a BDSYN description of the behavior of the network.

The target technology was a fixed library NAND/NOR with up to 4 inputs per gate – no complex gates were used. MIS is currently oriented towards complex gate CMOS, so a manual translation was performed on the output of MIS to this technology. No optimization or merging of gates was done during the translation so we expect these results to be an upper bound on the quality of the final network. (It is expected that better results could be obtained by performing local optimization over the final network taking into account the constraints of the technology.) The results of this experiment are given in Figure 4.

The results from the first two examples are very encouraging. The difference in the third example is explained, in part, from the fact that MIS is currently unable to use the don't-care information inherent in the network. In a second experiment, listed as *ex3b*, the network was minimized as a two-level PLA with a don't-care set, and then re-extracted into a multiple-level network. This network was not mapped into the technology for a direct comparison; however, it appears to be about two-thirds of the size. In a third experiment, listed as *ex3c*, we used MIS interactively, trying some of the more powerful operators, especially Boolean resubstitution, and obtained a further reduction to an estimated 10% increase over the manual design. We expect to improve on this result as MIS matures.

```
sweep
resub
eliminate -1
resub
eliminate -1
resub
fastdecomp *
resub
eliminate 0
resub
eliminate 0
simplify0 *
resub
kextract 5
eliminate 0
simplify1 *
resub
fastdecomp *
resub
cextract
eliminate 0
resub
verify 20
```

Figure 2. A Typical MIS Script.

## Acknowledgments

Many people participated in the synthesis class at the University of California, Berkeley, in the Spring of 1986 and we acknowledge the efforts of A. R. Newton, A. Sangiovanni-Vincentelli, and C. Sequin in bringing this class together. We acknowledge the financial support of SRC under contract 82-11-008, the Defense Advanced Research Projects Agency under contract N00039-86-R-0365, the California State Micro program, and NSF under a University of Colorado subcontract of grant ECS-8430435.

## References

- [1] K. Bartlett, R. Brayton, G. Hachtel, R. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. *Multilevel Logic Minimization using Implicit Don't-Cares*. In *Proceedings International Conference on Computer Design (ICCD)*, pages 552–557, October 1986.
- [2] R. Brayton and C. McMullen. *The Decomposition and Factorization of Boolean Expressions*. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, pages

**GLOBAL OPTIMIZATION**

collapse: collapse the network to two level  
 kextract: extract kernels  
 qextract: extract level 0 kernels  
 cextract: extract single cube factors  
 resub: resubstitute node to all other node  
 eliminate: eliminate nodes below threshold  
 sweep: eliminate 0-fanout or 1-fanin nodes  
 quickphase: quick phase assignment  
 goodphase: good phase assignment

**LOCAL OPTIMIZATION**

complement: compute the complement  
 decomp: decompose large gates  
 fastdecomp: fast decomposition  
 simplify0: simplify using SIMPCOMP  
 simplify1: simplify using ESPRESSO only  
 simplify2: simplify using ESPRESSO with don't care set  
 strongd: strong division one node by another  
 invert: invert nodes  
 qfactor: factor nodes using quick-factoring  
 gfactor: factor nodes using good-factoring  
 kernel: generate kernels  
 addinv: add inverters as needed

**TIMING OPTIMIZATION**

reduce: reduce the delay through a critical node  
 speedup: speed up the network by percentage  
 delay: calculate delays  
 setsslack: set the output slacks

**MISCELLANEOUS**

verify: verify current network against original  
 lopen: open a log file  
 lcclose: close the current log file  
 backup: backup the current copy of network  
 restore: restore the backup copy of the network  
 undo: undo last command that changed network  
 source: execute MIS commands in a file

*Figure 3.* Partial list of MIS commands.

Example	Manual Design			MIS DESIGN			Ratio of Devices
	Gate Count	Device Count	Logic Levels	Gate Count	Device Count	Logic Levels	
ex1	54	245	7	46	206	6	0.84
ex2	95	406	6	69	306	9	0.75
ex3	162	814	6	445	1965	12	2.50
ex3b	162	814	6	-	1250	-	1.53
ex3c	162	814	6	-	900	-	1.10

Figure 4. MIS results compared to manual design.

49–54, 1982.

- [3] R. Brayton and C. McMullen. *Synthesis and Optimization of Multistage Logic*. In *Proceedings International Conference on Computer Design (ICCD)*, pages 23–28, 1984.
- [4] R. Brayton, C. McMullen, G. Hachtel, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [5] R. Katz (ed). *Proceedings of CS292i: Implementation of VLSI Systems*. University of California, 1985.
- [6] D. Harrison, P. Moore, R. Spickelmier, and A. R. Newton. *Data Management and Graphics Editing in the Berkeley Design Environment*. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 20–24, November 1986.
- [7] D. Patterson and C. Sequin. *A VLSI RISC*. *IEEE Computer*, 15(9):8–21, 1982.

# EXACT MINIMIZATION OF MULTIPLE-VALUED FUNCTIONS FOR PLA OPTIMIZATION

R. Rudell and A. Sangiovanni-Vincentelli

*University of California*

*Berkeley, CA*

## Abstract

We present an algorithm for determining the minimum representation of an incompletely-specified, multiple-valued input, binary-valued output, function. The overall strategy is similar to the well-known Quine-McCluskey algorithm; however, the techniques used to solve each step are new. The advantages of the algorithm include a fast technique for detecting and eliminating from further consideration the essential prime implicants and the totally redundant prime implicants, and a fast technique for generating a reduced form of the prime implicant table. The minimum cover problem is solved with a branch and bound algorithm using a *maximal independent set* heuristic to control the selection of a branching variable and the bounding. Using this algorithm, we have derived minimum representations for several mathematical functions whose unsuccessful exact minimization has been previously reported in the literature. The exact algorithm has been used to determine the efficiency and solution quality provided by the heuristic minimizer Espresso-MV [11]. Also, a detailed comparison with McBoole [2] shows that the algorithm presented here is able to solve a larger percentage of the problems from a set of industrial examples within a fixed allocation of computer resources.

## 1. Introduction

Programmable Logic Arrays (PLA's) are an important design style in digital integrated circuit design [4]. The optimization of PLA's includes logic optimization steps (e.g., output phase assignment [4, 13], input variable assignment to decoders [4, 13] and classical minimization of the logic equations [7, 6, 1]) and topological optimization (e.g., PLA folding [5]). In this paper, we consider the problem of determining a minimum equivalent representation for *multiple-valued input, binary-valued output functions*. By stating the algorithms in terms of multiple-valued functions, we immediately consider the minimization of single- and multiple-output PLA's, the minimization of single- and multiple-output PLA's with input decoders of arbitrary size, and the minimization of arbitrary multiple-valued input, binary-valued output functions.

The primary goal for minimization algorithms for PLA design is to minimize the total number of terms needed for the sum-of-products representation of the logic function. This reflects the goal of minimizing the area (and to a first-order approximation, the delay) of the PLA. As a secondary concern, we seek the representation requiring the fewest transistors in the PLA to reduce the inter-

nal capacitance and increase possibilities for folding. Exact minimization algorithms consist of the following steps: (1) generate all of the *prime implicants* of the switching function and form the *prime implicant table*; and, (2) derive a minimum cover for the *prime implicant table*.

Many different algorithms have been presented for solving these two problems dating back to the early work of Quine [8]. The established Quine-McCluskey [7] procedure for Boolean minimization generates the prime implicants starting from a list of minterms. The prime implicant table has a row for each minterm and a column for each prime implicant. For each minterm row, a 1 is placed in a column if the corresponding prime implicant contains the minterm. The problem of selecting a minimum subset of primes has then been mapped into the problem of selecting a minimum cover of this matrix. (A cover is a row vector of 0's and 1's such that each row of the matrix shares a 1 in some column with the row vector, and a minimum cover is one with the fewest number of 1's.) Before the covering problem is solved, standard techniques such as detecting *essential columns*, and deleting *dominated rows* and *dominated columns* are used to create a reduced prime implicant table.

The basic terms are defined in Section 2, and the minimization procedure is described in detail in Section 3. Results from an implementation of the procedure are given in Section 4.

## 2. Definitions

**Definition 2.1:** Let  $p_i$  for  $i = 1, \dots, n$  be positive integers representing the number of values for each of  $n$  variables. Define the set  $P_i \equiv \{0, \dots, p_i - 1\}$  for  $i = 1, \dots, n$  which represents the  $p_i$  values that variable  $i$  may assume, and define  $B \equiv \{0, 1, *\}$  which represents the value of the function. A **multiple-valued input, binary-valued output function**,  $f$ , (hereafter known as a **multiple-valued function**) is a mapping

$$f : P_1 \times P_2 \times \dots \times P_n \rightarrow B$$

The function is said to have  $n$  multiple-valued inputs, and variable  $i$  is said to take on one of  $p_i$  possible values. Each element in the domain of the function is called a **minterm** of the function.

The value  $* \in B$  will represent a minterm for which the function value is allowed to be either 0 or 1. Hence, we allow functions which are *incompletely specified*.

**Remark:** An  $n$ -input,  $m$ -output switching function can be represented by a multiple-valued function of  $n + 1$  variables where  $p_i = 2$  for  $i = 1, \dots, n$ , and  $p_{n+1} = m$ . This special case is called a **multiple-output function**. It is easily proven that the minimization problem for multiple-output functions is equivalent to the minimization of a multiple-valued function of this form [12] (Theorem 4.1). Likewise, the minimization problem for a PLA with input decoders [4] is

also equivalent to the minimization of a multiple-valued function [12] (Theorem 2.1).

**Definition 2.2:** Let  $X_i$  be a variable taking a value from the set  $P_i$ , and let  $S_i$  be a subset of  $P_i$ .  $X_i^{S_i}$  represents the Boolean function

$$X_i^{S_i} = \begin{cases} 0 & \text{if } X_i \notin S_i \\ 1 & \text{if } X_i \in S_i \end{cases}$$

$X_i^{S_i}$  is called a **literal** of variable  $X_i$ . If  $S_i \equiv \emptyset$ , then the value of the literal is always 0, and the literal is called **empty**. If  $S_i \equiv P_i$ , then the value of the literal is always 1, and the literal is called **full**.

A **product term** is a Boolean product (AND) of literals. If a product term evaluates to 1 for a given minterm, the product term is said to contain the minterm.

If all literals in a product term are full, the product term contains all minterms, and is called the **universal product term**.

A **sum-of-products** (also called a **cover**) is a Boolean sum (or OR) of product terms. If any product term in the sum-of-products evaluates to 1 for a given minterm, then the sum-of-products is said to contain the minterm.

The **ON-set** is the set of minterms for which the function value is 1. Likewise, the **OFF-set** is the set of minterms for which the function value is 0, and the **DC-set** is the set of minterms for which the function value is unspecified.

An **algebraic expression** for  $f$  is a Boolean expression (written using Boolean sums and Boolean products of literals) which evaluates to 1 for all minterms of the ON-set, evaluates to 0 for all minterms of the OFF-set, and evaluates to either 0 or 1 for all minterms of the DC-set.

**Proposition 2.1:** An algebraic expression for  $f$  can always be written in sum-of-products form.

An **implicant** of a function  $f$  is a product term which does not contain any minterm in the OFF-set of the function. A **prime implicant** of a function  $f$  is an implicant which is contained by no other implicant of the function.

The **Boolean Minimization Problem** is to determine the minimum cost sum-of-products representation of a given multiple-valued function. For a PLA implementation, the cost function is to minimize the total number of product terms in the representation. This is the cost function which is considered here, although extension to a more general cost function is also possible.

## 2.1 Positional Cube Notation

Let  $X_1^{S_1}X_2^{S_2}\dots X_n^{S_n}$  be a product term. This product term can be represented by a binary vector:

$$c_1^0 c_1^1 \dots c_1^{p_1-1} - c_2^0 c_2^1 \dots c_2^{p_2-1} - \dots - c_n^0 c_n^1 \dots c_n^{p_n-1}$$

where  $c_i^j = 0$  if  $j \notin S_i$ , and  $c_i^j = 1$  if  $j \in S_i$ . This is called the **positional cube notation** or more simply a **cube** [17]. A cube is a convenient representation for a product term, and the terms cube and product term will often be used interchangeably.

The notation  $c_i$  represents the binary vector  $c_i^0 c_i^1 \dots c_i^{p_i-1}$ , and  $|c_i|$  represents the number of 1's in the binary vector.

### 3. Algorithms

The algorithms presented here for solving the basic steps of the minimization problem are based on the heuristic algorithms which are used in Espresso-MV [10, 11]. Hence, we will refer to this algorithm as Espresso-EXACT. We assume we are given a cover of the ON-set of a multiple-valued function (called  $F$ ), a cover of the DC-set (called  $D$ ), and a cover of the OFF-set (called  $R$ ). (If the complement is not provided, it may be efficiently computed using a fast multiple-valued complementation algorithm [15, 10]). The Espresso-EXACT Minimization Algorithm is:

- 1 Generate from  $R$  all of the prime implicants ( $P$ ) of the function.
- 2 Detect the essential primes ( $E$ ), the totally redundant primes ( $R_t$ ), and the partially redundant primes ( $R_p$ ) using a fast tautology algorithm.
- 3 Create the *prime implicant table* ( $A$ ) for the partially redundant primes using a modification of the tautology algorithm to record how a function avoids being a tautology.
- 4 Generate a minimum cover for the prime implicant table.
- 5 Select the primes which are in the cover for the minimum cover.
- 6 Heuristically remove redundant literals from the cover using the `MAKE_SPARSE` heuristic of Espresso-MV [10].

#### 3.1 Generation of Prime Implicants

Many techniques for determining all of the prime implicants of binary-valued single-output and multiple-output logic functions have been published in the past [7, 18]. More recently, an algorithm based on recursive decomposition of a function followed by a pairwise consensus operation has been reported [3], and has been improved upon in the program McBoole [2] to reduce the total number of consensus operations which need to be performed. In this paper, we mention another paradigm for generating all of the prime implicants, which yields two slightly different algorithms.

In order for a cube  $c$  to be an implicant of  $F$ ,  $c$  must not intersect each cube  $r^i \in R$ . This can be expressed by writing a Boolean expression. Let  $c_j^k$  be a

Boolean variable representing the condition that part  $k$  of variable  $j$  of cube  $c$  be set to 1. Let  $(r^i)_j^k$  have the value of 1 if part  $k$  of variable  $j$  of the cube  $r^i$  is a 1. The following Boolean expression asserts that  $c$  does not intersect the OFF-set of the function:

$$I = \bigcap_{i=1}^{|R|} \bigcup_{j=1}^n \bigcap_{\substack{k=0 \\ |r_j^i| \neq p_j}}^{p_j-1} ((\bar{r}^i)_j^k + \bar{c}_j^k)$$

We stress that the values of  $r^i$  written as  $(r^i)_j^k$  are known values of either 0 or 1, and that the variables in the above equation are  $c_j^k$ .

To form a sum-of-products representation of  $I$  requires that the product-of-sums-of-products expression be “multiplied-out”, that is, repeated intersection of sums-of-products covers. However, using DeMorgan’s law, we can directly write the expression for  $\bar{I}$ :

$$\bar{I} = \bigcup_{i=1}^{|R|} \bigcap_{j=1}^n \bigcup_{\substack{k=0 \\ |r_j^i| \neq p_j}}^{p_j-1} ((r^i)_j^k c_j^k)$$

An implicant of the function  $I$  corresponds to an assignment of 0, 1 to the variables  $c_j^k$  which results in an implicant of  $f$ . Further, a prime implicant of  $I$  corresponds to an assignment of 0, 1 to the variables  $c_j^k$  which is maximal in the sense that no other variable which is 0 can be made a 1; therefore, a prime implicant of  $I$  corresponds to a prime implicant of  $f$ .

By construction, the logic function  $I$  is a two-valued unate logic function. Hence, any prime cover for  $I$  consists of all of the prime implicants for  $I$  [1] (Prop. 3.3.7).

This construction proposes two techniques for generating all of the prime implicants of a function: one which involves repeated intersection of sum-of-products forms and one which involves the complementation of a sum-of-products form. We note here that the first formulation is equivalent to the technique outlined by Roth [9] for generating all of the prime implicants of a multiple-output logic function.

### 3.2 Separation of $E$ , $R_t$ and $R_p$

The primes  $P$  are split into the essential set  $E$ , the totally redundant set  $R_t$ , and the partially redundant set  $R_p$  according to the following rules:

$$\begin{aligned} E &\equiv \{c \in P \mid c \not\subset (P \cup D - c)\} \\ R_t &\equiv \{c \in P, c \notin E \mid c \subset (P \cup D - c)\} \\ R_p &\equiv P - (E \cup R_t) \end{aligned}$$

The cubes of  $E$  must belong to any cover of the function, (it is the set of essential prime implicants), and no cube of  $R_t$  can ever belong to a minimum cover of  $F$  (it is the set of implicants dominated by the essential prime implicants). The cubes of  $R_p$  are partially redundant because, although any single cube of  $R_p$  can be removed, it is not possible to simultaneously remove all of the cubes of  $R_p$  while still maintaining a cover of  $F$ .  $R_p$  causes the most difficulty in trying to extract a minimum subset of  $P$ .

The separation of  $P$  into the covers  $E$ ,  $R_t$ , and  $R_p$  is accomplished with a fast multiple-valued tautology algorithm [14, 10]. The basic test  $c \subseteq H$  is done by forming the cofactor  $H_c$ , and then testing if  $H_c$  is a tautology (i.e., if the function evaluates to 1 for all inputs). The tautology check uses a generalization of the Shannon Cofactor [10] as follows:

**Proposition 3.1:** Let  $c^i, i = 1, \dots, m$  be a set of cubes satisfying  $\cup_{i=1}^m c^i = 1$  and  $c^i \cap c^j = \emptyset$  for  $i \neq j$ . Then,

$$F = \bigcup_{i=1}^m c^i \cap F_{c^i}$$

It is easy to show that the tautology question for a function  $F$  can be answered by the following proposition:

**Proposition 3.2:**  $F \equiv 1 \Leftrightarrow F_{c^i} \equiv 1$  for  $i = 1, \dots, m$

The recursion is terminated when  $F_{c^i}$  is trivial enough to allow a direct check for tautology. In particular, if the function  $F_{subcubec_i}$  is weakly-unate [10], the recursion can be terminated immediately.

### 3.3 Creating the Reduced Prime Implicant Table

The techniques for extracting a maximal subset of primes from  $R_p$  is now described. The key in the algorithm is a simple modification of the multiple-valued tautology algorithm. Rather than testing whether the function is a tautology, we determine which subsets of cubes in a function would have to be removed to prevent a function from becoming a tautology.

For each cube  $c \in R_p$ , form  $H = E \cup R_p - c$ , and use a multiple-valued tautology algorithm to determine if  $H_c$  is a tautology. By definition of  $E$  and  $R_p$ ,  $H_c$  must be a tautology (every cube of  $R_p$  is covered by the union of  $E$  and the remaining cubes of  $R_p$ ). At each leaf in this tautology algorithm (i.e., where it is decided that the partial function is a tautology), the cubes which are in the cover at this leaf are examined. If there is a cube from  $E$  (or  $D$ ) which is the universe in the restriction of the function corresponding to this leaf, then this leaf will always give rise to a tautology regardless of which cubes of  $R_p$  are discarded. Otherwise, all of the cubes of  $R_p$  which are the universe (in this leaf) must be removed in order to avoid having  $H_c$  become a tautology in this leaf. In terms of determining how a cover covers the cube, this is equivalent to saying that  $H$

will fail to cover  $c$  if and only if all of the cubes of  $R_p$  which are the universe in this leaf are discarded. In this way, we enumerate all possible ways for subsets of  $R_p$  to avoid covering the original function.

A binary matrix is formed where each cube of  $R_p$  is associated with a column. At each leaf in the tautology algorithm where no cube from  $E$  is the universal cube, a row is added to the binary matrix with a 1 for each column where  $(R_p)^i$  is the universe. A minimal cover of this Boolean matrix corresponds to a minimal subset of the primes of  $R_p$  which must be retained in the cover for  $F$ .

The algorithm proceeds by forming  $H_c$  for each  $c \in R_p$ , and calling a modified version of the TAUTOLOGY procedure called FIND\_TAUTOLOGY. Note that after determining how  $c$  can be covered,  $c$  can be moved to the set  $E$  thus improving the performance of the algorithm (because it is known how all of the minterms of  $c$  can be covered by selecting primes from  $R_p$ ).

The binary matrix formed in this way can be related to the prime implicant table of the Quine-McCluskey algorithm. This binary matrix is a reduced form of the prime implicant table; rather than each row of the matrix corresponding to a minterm of the function, each row corresponds to a collection of minterms (i.e., a larger subspace) all of which are covered by the same set of prime implicants. In the limit, the tautology algorithm will terminate at each of the minterms of the function, thus producing precisely the prime implicant table. However, in practice, the algorithm is terminated much more quickly, leading to a more efficient creation of the prime implicant table.

### 3.4 Covering Algorithm

The standard branch-and-bound solution to the minimum cover problem involves the following steps:

- 1 Remove rows which are dominated by other rows (i.e., the row contains some other rows), and remove columns which are dominated by other columns (i.e., the column is contained in some other column). Detect essential columns (a row with a single 1 identifies an essential column) and add these to the selected set. Repeat until no new essential elements are detected.
- 2 If the size of the selected set exceeds the best solution so far, return from this level of the recursion. Or, if there are no elements left to be covered, declare the selected set as the best solution recorded so far.
- 3 Select (heuristically) a branching column.
- 4 Add this column to the selected set and recur for the submatrix resulting from deleting this column and all rows which are covered by this column. Then, recur for the submatrix resulting from deleting this column without adding it to the selected set.

We propose the following two heuristics to speed up this standard covering algorithm:

- 1 At step 2, determine a lower bound on the size of a cover for the current submatrix. This can be done by determining a maximal set of rows all of which are pairwise disjoint (called a *maximal independent set of rows*) using a straightforward, greedy algorithm. (Note that the problem of finding a maximum independent set of rows is itself NP-complete; however, we need only a large independent set of rows.) Because each row must be covered, and all of the rows in the maximal independent set share no column in common, the size of the maximal independent set is a lower bound on the number of columns needed to complete the cover. At step 2, the recursion can be bounded if the size of the selected set at step 2 *plus* the size of the maximal independent set of rows equals or exceeds the best solution known. Hence, quite often, unprofitable searches are terminated very high in the recursion.
- 2 To select the branching column, we use the following heuristics. First, form the union over all of the rows in the maximal independent set of rows. At least one of these columns must be in the selected set; hence, we limit the selection of a branching column to one of these columns. Second, assign each nonzero element of each row a weight equal to the reciprocal of the number of nonzero elements in the row. Sum the weight for each column, and choose as a branching column the column of maximum weight which is also in the union over the maximal independent set of rows. This weighting strategy gives the elements of the smaller sets a higher weight, as they are viewed as "harder" to cover. Another reason for favoring an element from a smaller set (for example, a set with two elements) is to create more essential elements in the subsequent recursion.

## 4. Results

A program implementing this minimization algorithm, Espresso-EXACT, has been compared against another exact minimization program McBoole [2], and against the heuristic minimization program Espresso-MV [10] for a large set PLA's drawn from industrial designs (111 PLA's), and a smaller set of standard mathematical functions (23 PLA's).

With a test set so large, it is a challenge to present the results in a meaningful manner. Measuring either the number of inputs and outputs, or the number of product terms is a notoriously inaccurate measure of the complexity of Boolean minimization for a specific example. Therefore, we first classify each of the 134 minimization problems as shown in Figure 1.

The classifications were determined by allowing Espresso-EXACT and McBoole to run a maximum of 5 hours for each example on an Apollo DN660.

Classification	Description
<i>trivial</i>	minimum solution consists of essential prime implicants
<i>noncyclic</i>	the covering problem contains no cyclic constraints
<i>cyclic and solved</i>	the covering problem contains cyclic constraints and the minimum solution is known
<i>cyclic and unsolved</i>	the covering problem contains cyclic constraints but the minimum solution is unknown
<i>too many primes</i>	there were too many primes to be enumerated

Figure 1. Problem Classification.

By examining the results for each program, we arrive at the classification. If the problem was solved by either of the two exact minimization algorithms, it is easy to classify it as *trivial*, *noncyclic*, or *cyclic and solved*. An example is classified as *too many primes* only if neither program was able to enumerate the set of prime implicants, and an example is classified as *cyclic and unsolved* only if neither program was able to complete the covering program (after having generated the set of all prime implicants).

#### 4.1 Comparison of Exact Minimization Algorithms

Figure 2 summarizes the comparison between Espresso-EXACT and McBoole for the 134 PLA's in the test set. *Number primes* refers to the number of examples for which each program could enumerate all of the prime implicants. *number solved* refers to the number of examples each program could solve. *time* gives the time on an Apollo DN660 taken to solve those examples which could be solved within the 5 hour limit.

type	total	Espresso-Exact			McBoole		
		number primes	number solved	time (sec)	number primes	number solved	time (sec)
<i>trivial</i>	9	9	9	120	9	9	271
<i>noncyclic</i>	56	55	54	26,524	56	56	35,956
<i>cyclic-s</i>	42	42	41	41,330	42	21	11,241
<i>cyclic-u</i>	10	7	0		10	0	
<i>primes</i>	17	0	0		0	0	
Totals	134	113	104	67,974	117	86	47,468

Figure 2. Comparison between Espresso-EXACT and McBoole.

For examples with no cyclic constraints (i.e., *trivial* or *noncyclic*), both Espresso-EXACT and McBoole are usually able to find the minimum solution. However, when there are cyclic constraints, the covering algorithm of Espresso-

EXACT is able to find the minimum solution for many more of the PLA's than McBoole. Sometimes the results are quite dramatic. For example, McBoole was allowed to run 58 hours on one example without terminating; however, Espresso-EXACT is able to complete this example in only 100 seconds.

The number of prime implicants ranged from 6 to 9179, and 36 of the examples had more than 1,000 prime implicants. In each of the cases where McBoole was able to finish enumerating all primes and Espresso-EXACT was unable to, there were more than 5,000 prime implicants. This indicates that the prime generation strategy of McBoole is superior when the number of prime implicants becomes very large. The execution time for the prime generation algorithms varied greatly, although the average time was comparable.

There were 83 examples which both programs could minimize, 3 examples which only McBoole could minimize, 21 examples which only Espresso-EXACT could minimize, and 27 examples which neither program was able to minimize. For the 83 examples which both programs could minimize, Espresso-EXACT used 38,198 seconds, and McBoole used 28,628 seconds. As expected, both returned the same number of solution terms. The Espresso-EXACT result had 51,821 literals and the McBoole result had 53,686 literals, indicating that the heuristic algorithm of Espresso-EXACT was more effective at reducing the number of literals than the heuristic of McBoole.

## 4.2 Espresso-MV Results

Next, we examine the effectiveness of the heuristic minimization algorithm Espresso-MV. To compare the results of Espresso-MV and Espresso-EXACT, we consider only those examples for which Espresso-EXACT was able to generate the minimum solution. The results are reported in Figure 3. The column *number minimum* reports the number of examples for which Espresso-MV achieved the minimum solution.

type	number	Espresso-MV			Espresso-Exact	
		number minimum	solution terms	time (sec)	solution terms	time (sec)
<i>trivial</i>	9	9	243	23	243	120
<i>noncyclic</i>	54	48	3,371	1,366	3,360	26,523
<i>cyclic-s</i>	41	19	3,463	2,532	3,395	41,329
Totals	104	76	7,077	3,920	6,998	67,973

Figure 3. Comparison between Espresso-MV and Espresso-EXACT.

We see that Espresso-MV provides a high quality result for all of the examples for which we can generate a minimum solution. In many cases, Espresso-MV is providing the minimum solution, and the total difference in terms between Espresso-MV and Espresso-EXACT is only one percent. The largest difference

for any single example was 7.5%. Not surprisingly, Espresso-MV has the most difficulty with the more difficult problems in the test set (i.e., those problems with cyclic constraints).

The comparison shows that Espresso-MV frequently produces results very near the minimum solution, and, as expected, the heuristic minimizer is much faster than the exact algorithm. However, Espresso-MV cannot guarantee that it has achieved the minimum solution. The large increase in time for the exact minimizer is primarily due to the effort needed to prove that the produced result is in fact minimum.

### 4.3 Exact Results for Mathematical Functions

Mathematical functions have long been used as standard benchmarks for minimization algorithms. We report here some new results on the minimum size of several mathematical functions. These examples have been examined by other exact minimizers without providing the minimum solution [2, 16].

The problems involve simple mathematical functions (multipliers *mlp3* and *mlp4*, norm function *dist4*, and a squaring circuit *sqr6*). They have been solved as a multiple-output minimization problem (*normal* in the table), and using two-input decoders (*bit-paired* in the table). These problems are deceptively small and have a small number of prime implicants. The difficulty lies in extracting the minimum cover for the prime implicant table. Figure 4 presents the results for these examples. Times are reported for a DEC MicroVax-II running Ultrix and are in seconds. Examples marked with '+' represent a new result.

Name	In/Out	Normal			Bit-paired		
		Primes	Minimum	Time	Primes	Minimum	Time
<i>mlp3</i>	6/6	90	+ 30	12	71	22	8
<i>mlp4</i>	8/8	606	+ 121	4,700	561	+ 83	119,000
<i>dist4</i>	8/5	401	120	100	493	+ 94	11,500
<i>sqr6</i>	6/12	205	+ 47	114	241	39	93

Figure 4. Results on previously unsolved mathematical function problems.

We note that the best previous results for *mlp4* were 123 and 88 for the normal and bit-paired cases, respectively. For the other examples marked with '+', we have verified that the previously known solutions are minimum.

### Acknowledgments

We wish to thank Michel Dagenais for providing access to his program McBoole for comparison, and for answering questions on the use of the program. We also wish to thank Bob Brayton of IBM Research for many interesting discussions on two-level minimization and the exact minimization problem. We

gratefully acknowledge the support of DARPA under contract number N00039-C-0107, and the California State Micro program with matching grants from GTE labs, Fairchild, Olivetti, Silicon Compilers, and MCC. Richard Rudell was supported by an IBM Graduate Research Fellowship while this work was performed.

## References

- [1] R. Brayton, C. McMullen, G. Hachtel, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [2] M. Dagenais, V. Agarwal, and N. Rumin. *McBoole: A New Procedure for Exact Logic Minimization*. *IEEE Transactions on Computer-Aided Design*, C-33:229–238, January 1986.
- [3] R. Brayton et al. *Fast Recursive Boolean Function Manipulation*. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, page 58, May 1982.
- [4] H. Fleisher and L. Maissel. *An Introduction to Array Logic*. *IBM Journal of Research and Development*, 19:98–109, March 1975.
- [5] G. Hachtel, A. R. Newton, and A. Sangiovanni-Vincentelli. *An Algorithm for Optimal PLA Folding*. *IEEE Transactions on Computer-Aided Design*, pages 63–76, January 1982.
- [6] S. Hong, R. Cain, and D. Ostapko. *MINI: A Heuristic Approach for Logic Minimization*. *IBM Journal of Research and Development*, 18:443–458, September 1974.
- [7] E. McCluskey. *Minimization of Boolean Functions*. *Bell System Technical Journal*, 35:1417–1444, April 1956.
- [8] W. Quine. *The Problem of Simplifying Truth Functions*. *American Mathematical Monthly*, 59:521–531, 1952.
- [9] J. Roth. *Computer Logic, Testing, and Verification*. Computer Science Press, 1981.
- [10] R. Rudell. *Multiple-Valued Logic Minimization for PLA Synthesis*. Master's thesis, University of California, Berkeley, June 1986. Memorandum UCB/ERL M86/65.
- [11] R. Rudell and A. Sangiovanni-Vincentelli. *Espresso-MV: Algorithms for Multiple-Valued Logic Minimization*. In *Proceedings Custom Integrated Circuits Conference (CICC)*, pages 230–234, May 1985.
- [12] T. Sasao. *An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays*. In *Proceedings 8th International Symposium on Multiple-Valued Logic*, 1978.
- [13] T. Sasao. *Input Variable Assignment and Output Phase Optimization of PLA's*. *IEEE Transactions on Computers*, C-33:879–894, October 1984.
- [14] T. Sasao. *Tautology Checking Algorithms for Multiple-Valued Input Binary Functions and Their Application*. In *Proceedings 14th International Symposium on Multiple-Valued Logic*, 1984.
- [15] T. Sasao. *An Algorithm to Derive the Complement of a Binary Function with Multiple-Valued Inputs*. *IEEE Transactions on Computers*, C-34:131–140, February 1985.
- [16] T. Sasao. Personal Communication, 1986.
- [17] Y. H. Su and P. T. Chueng. *Computer Minimization of Multi-Valued Switching Functions*. *IEEE Transactions on Computers*, C-21:995–1003, 1972.
- [18] P. Tison. *Generalization of Consensus Theory and Application to the minimization of Boolean Functions*. *IEEE Transactions on Computers*, C-16:446, August 1967.

# IMPROVED LOGIC OPTIMIZATION USING GLOBAL-FLOW ANALYSIS

C. Leonard Berman and Louise H. Trevillyan

*IBM T.J. Watson Research Center  
Yorktown Heights, NY 10598*

## Abstract

This paper is concerned with techniques for automatically reducing circuit size and improving testability. In an earlier paper [2], we introduced a new method for circuit optimization based on ideas of global-flow analysis. In this paper, we describe two extensions to the method. The first is a basic improvement in the primary result on which the earlier optimization was based, the second extends the applicability of the method to “conditional” optimizations as well. Together, these enhancements result in improved performance for the original algorithm, as well as the ability to handle designer-specified “don’t cares” and redundancy removal uniformly in the framework of a graph-based synthesis system, such as LSS[8].

## 1. Overview

There are two basic approaches to multi-level design: 1) the algebraic/boolean approach [6], and 2) the graph-based approach [4]. Although systems generally incorporate ideas from both approaches, it is fair to say that the boolean/algebraic method typically represents a function as a directed acyclic graph (dag) *whose nodes compute arbitrary functions* and performs optimizations using factoring and two-level minimization on the nodes. On the other hand, the graph-based approach represents a function as a dag *whose nodes compute simple functions* and performs optimizations using graph manipulation and data flow algorithms. This paper describes a significant enhancement to an optimization which belongs to the conceptual and algorithmic framework of the graph-based synthesis approach. We note that there has been progress toward applying these ideas in the framework of algebraic synthesis [7].

In [2] the authors introduced a new circuit optimization which used ideas from global-flow analysis to optimize a circuit by first computing circuit summary information and then using the information via the min-cut algorithm to reduce the number of connections in a circuit. This paper describes improvements to this method. There are two main contributions. The first is an improvement of the main theorem of [2] which guaranteed that the optimization was legal, i.e. that it left the function of the circuit unchanged. The second contribution is a refinement of the method that allows “conditional optimizations” to be handled uniformly. These conditional optimizations include connection-reduction optimizations which use designer-specified “don’t care” information [5], as well as redundancy-removal optimizations. Together, the extensions reported here re-

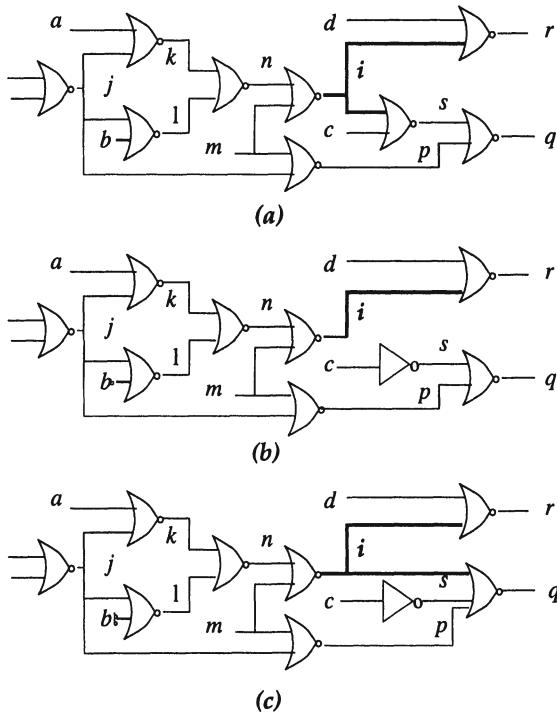


Figure 1. An example of Global-Flow Optimization.

sult in a significant conceptual, as well as practical, simplification to the logic optimization phase of LSS. This phase, which until recently included more than a dozen local transformations, can now be described entirely in terms of two primitive processes: global-flow methods for boolean logic optimization, and factoring to satisfy fan-in and fan-out constraints.

The global-flow method is described in detail in [2, 3]. At a high level, it can be described as follows. Iteratively for each net, the procedure determines how the terminals of the current net can be rearranged and chooses a legal rearrangement for implementation. These legal rearrangements are determined assuming the net carries the “controlling value” (e.g. the controlling value of a NOR is 1 and of a NAND is 0) and reflect the boolean nature of the operators (as opposed to the algebraic nature of factoring). These possible rearrangements are determined using summary information which consists of assertions concerning the state of other nets in the circuit. The correctness of the rearrangements is guaranteed by Theorem 1 of [2] which states that rearrangements of the terminals of a net that leaves the functions at its “frontier” unchanged are legal. (These concepts will be defined precisely later).

**Example:** We use the circuit in Figure 1 to illustrate throughout. In figure 1(a), when the procedure considers signal “i”, it performs deduction under the assumption that  $i = 1$ , since 1 is the controlling value for NOR gates. It derives the assertions:  $n = 0, m = 0, j = 0$  (because  $j = 1$  implies  $i = 0$ , so  $i = 1$  implies  $j = 0$ ),  $p = 1, r = 0, s = 0$  and  $q = 0$ . It then determines that  $\text{FRONTIER}(i) = \{q = 0, r = 0\}$ . The algorithm then changes the terminals of “i” to produce the circuit shown in Figure 1(c).

Since the optimizations involve rearranging the terminals of the net under consideration, it is important to differentiate between assertions which are independent of the precise connections of the current net and those which depend on where this net goes. Our first result addresses this distinction. In our earlier paper, this distinction was captured to some extent in the definition of the frontier. In this paper, this distinction is refined through the idea of an assertion concerning the circuit being “free”. The earlier theorem is extended to show that any rearrangement which leaves the non-free part of the frontier unchanged is legal. This results in significant connection reduction in some examples. Note that in the above example, the connection of “i” at the source of  $s$  is not necessary because the assertion  $q = 0$  is independent of where “i” is connected. It is this type of information which is captured in the notion of a “FREE” assertion.

The second enhancement to the method results from an improvement in the formalism which permits us to reason about individual terminals rather than entire nets. This does not require any fundamental change to the techniques, but it does permit “conditional” optimizations to be accommodated in the same theoretical framework. These conditional optimizations include connection-reduction optimizations that utilize designer-specified output “don’t care” information (this has not been possible before in graph-based synthesis systems) and in redundancy-removal optimizations. Connection-reduction optimizations are those

which leave the function unchanged at the most forward unchanged nodes, while redundancy-removal optimizations may change the function at the most forward nodes, but the change can not be seen at any output because of the circuit structure.

## 2. Summary of the Technical Material

### 2.1 Terminology and Foundations

Circuits are represented as dags. For simplicity, we assume the nodes of the dag to be NORs, INPUTs or OUTPUTs. We use terms gate, node, signal and terminal in the standard way. Since we assume each gate has a single output, we will also identify the net with the node which is its source. In what follows  $C$  is

an arbitrary circuit satisfying the above constraints.

**Definition:**

$A = \{(c_i, v_i) \mid c'_i s \text{ are terminals of } C \text{ and } v'_i s \text{ are associated values}\}$   
is called a *partial assignment* of  $C$ .

We can think of a partial assignment as a set of assertions (or deductions) about the state of the circuit. For example, if  $(c, 1) \in A$ , we may say that  $A$  contains the assertion  $c=1$ , or equivalently that  $A$  asserts that terminal  $c$  has value 1. We will write partial assignments as sets of pairs or sets of assertions interchangeably. Note that a partial assignment,  $A$ , determines a set of possible input values, i.e. all input choices which result in the assertions of  $A$  being true. We shall refer to such inputs as *input assignments of A*.

**Definition:** For any partial assignment  $A$  of  $C$ , let  $A^*$  denote the partial assignment consisting of all assertions implied by  $A$ . This is called the *closure of A*.

Although the results described here are independent of the method used to derive these assertions, for the sake of clarity in this paper, we use the following recurrences, which we introduced in [2], as our deductive system. Let  $X(s) = \{\text{inputs of the source of } s\}$ .

$$\begin{aligned} C_{10}(A) &= \{(s, 0) \mid \exists y \in X(S)[(y, 1) \in C_{11}(A)]\} \cup A \\ &\cup \{(s, 0) \mid \exists y[(y, 1) \in C_{11}(A) \wedge s \in X(y)]\} \\ &\cup \{(s, 0) \mid \exists x[(x, 1) \in A, (x, 0) \in C_{10}(\{s, 1\})]\} \\ &\cup \{(s, 0) \mid \exists c \text{ on the same net as } s, (c, 0) \in C_{10}(A)\} \\ C_{11}(A) &= \{(s, 1) \mid \exists y, (y, 0) \in C_{10}(A), s \in X(y), \forall t \in X(y)[t \neq s \Rightarrow \\ &(t, 0) \in C_{10}(A)]\} \\ &\cup \{(s, 1) \mid \forall y \in X(s)[(y, 0) \in C_{10}(A)]\} \cup A \\ &\cup \{(s, 1) \mid \exists [(x, 1) \in A, (x, 0) \in C_{00}(\{s, 0\})]\} \\ &\cup \{(s, 1) \mid \exists c \text{ on the same net as } s, (c, 1) \in C_{11}(A)\} \end{aligned}$$

Similarly for  $C_{00}$  and  $C_{01}$ .

The use of this deductive method results in the closure,  $A^*$ , being the least fixed point of these recurrences. We note that Hachtel et. al. [9] uses related deductive methods while Brayton et. al. [7] takes a different approach.

The definition of partial assignments permits different values to be associated with different terminals of a net. Such a partial assignment corresponds to inconsistent assertions about the state of the circuit. Inconsistent assignments are

ruled out through the notion of a compatible assignment.

**Definition:** A partial assignment  $A$  is said to be *compatible in  $C$*  if whenever  $(c_i, v_i)$  and  $(c_j, v_j) \in A^C$ , and  $c_j$  and  $c_i$  refer to terminals of the same net, then  $v_j = v_i$ .

The following notation will be useful. For any compatible  $A$ , we let  $C|A$  be a circuit identical to  $C$  but with all terminals of  $A$  removed, and each terminal of  $A$  which is the input to a gate replaced by a connection to a new primary input. Also, let  $B(A) = \{(c, v) \mid (c, v) \in A^C \text{ and there is no path in } C \text{ from any terminal in } A \text{ to any terminal on the same net as } c\}$ .

Observe that  $B(A)$  contains assertions which are dependent on how the signals in  $A$  are computed but independent of how they are used. This is true because any assignment to inputs that results in the terminal-value pairs of  $A$  must also result in the pairs contained in  $B(A)$ . In fact, for any input assignment of  $A$ , if  $C$  is changed by rearranging terminals in  $A$  (subject to some restrictions), the assertions of  $B(A)$  still hold. This is because no path exists from  $A$  to  $B(A)$ , so there can be no direct dependence of these assertions on the terminals in  $A$ . We illustrate this with the following example.

**Example:** Consider the circuit from Figure 1(a), and assume that the partial assignment,  $A$ , under consideration sets all terminals of signal “ $i$ ” to 1. We see that  $A^C = \{(n, 0), (m, 0), (j, 0), (p, 1), (r, 0), (s, 0), (q, 0)\}$ , with each signal-value pair replaced by all the appropriate terminal-value pairs. We also see that  $B(A)$  contains the assertions  $n=0, m=0, j=0$  and  $p=1$ , again with signals replaced by the appropriate terminals.

The independence is important and leads us naturally to our main definition.

**Definition:**  $\text{FREE}(A) = B(A)^{C|A}$ .

**Example:** If we continue the earlier example, we see that  $\text{FREE}(A)$  contains  $q=0$  as well as the assertions  $n=0, m=0, j=0$  and  $p=1$  which were also in  $B(A)$ .

We see intuitively that the assertions in  $\text{FREE}(A)$  are consistent with  $A^C$  and also independent of  $A$ . Consistency is established by:

**Lemma:** If  $A$  is compatible then

$$\{c \mid \{(c, 0), (c, 1)\} \subset \{\text{FREE}(A) \cup A^C\}\} = \emptyset.$$

**Proof:** From the definition of the closure operator, we see that for any set of assertions,  $X$ , and partial assignment,  $A$ ,  $X^{C|A} \subseteq X^C$  and that  $B(A)^C \subseteq A^C$ . Combining these observations shows that  $\text{FREE}(A) \subseteq A^C$ , and since  $A$  is compatible,

the result follows.

Intuitively, the independence of  $\text{FREE}(A)$  follows from the construction of  $C|A$ . By replacing connections of terminals of  $A$  which are inputs of gates by connections of new primary inputs, we prevent the establishment of any assertions which depend on the precise terminals of  $A$ .

Another idea which we need is that of the frontier of a signal in a set of nodes. Intuitively, the frontier is the subset of nodes closest to the outputs. More formally, given a signal “ $i$ ” in  $C$  and a partial assignment,  $A$ , we define the *frontier of i in A*,  $\mathcal{F}(A,i)$ , as the set of nets,  $j$ , for which

$$1 \quad (j,0) \in A^C$$

$$2 \quad \text{There is a path } j \rightarrow j_1 \rightarrow j_2 \rightarrow \dots \text{OUTPUT such that for no } j_k \text{ does } j_k \in A^C$$

$$3 \quad j \text{ is reachable in the circuit from } i.$$

When we say “frontier of a net”, we are referring to the partial assignment which assigns one to all terminals of that net.

The importance of the sets  $\mathcal{F}(A,i)$  and  $\text{FREE}(A)$  is illustrated by the following theorem:

**Main Theorem:** Let  $i$  be a net in  $C$  and let  $A$  be any partial assignment which sets  $i$  to 1. Let  $C'$  be identical to  $C$  except that terminals of  $i$  in  $A$  may be missing and additional terminals of  $i$  may be present. Assume that none of the connections of  $i$  in  $C'$  are to nodes in  $\text{FREE}(A)$ . If in the two circuits the sets  $\{x \mid x \in \mathcal{F}(A,i) \wedge x \notin \text{FREE}(A)\}$  are identical, then the two circuits compute the same function.

This theorem is very similar to the main theorem of [2]. In fact, it is established by showing that, in this case, the hypotheses of our earlier theorem hold. In our earlier work, we could guarantee that a rearrangement was legal only if the frontier sets in the two circuits were identical; while to apply this theorem, we require only that the non-“FREE” part of the frontier sets be identical. Since connections must be added to maintain the equivalence of these two sets, we see that our new result can, in principle, result in smaller circuits. We have found this to be true in practice.

**Example:** Continuing the example, we saw earlier that  $\mathcal{F}(A,j) = \{q = 0, r = 0\}$ . If we combine this with the value for  $\text{FREE}(A)$  computed above, we see that  $\{x \mid x \in \mathcal{F}(A,j) \wedge x \notin \text{FREE}(A)\} = \{r = 0\}$ . The above theorem then guarantees that the circuit shown in Figure 1(b) is equivalent to that shown in Figure 1(a). Note that in Figure 1(b), signal  $i$  has only two terminals. As mentioned

earlier, our previous method which did not make use of “FREE” connections would result in the circuit shown in Figure 1(c), in which net i has three terminals. We realize that this is a simple example which could be handled by other less sophisticated methods. It is meant only to illustrate the improvement in the new optimization.

As presented here, the method appears to be computationally intensive since, to compute the “FREE” connections, we must perform deductions for a new circuit. However, we note that a good approximation to the deductions embodied in the recurrences can be computed on a signal-by-signal basis. This is done by a straightforward use of dataflow propagation techniques [1, 10] which never compute information that will be invalidated before it is needed. The running time of this procedure is proportional to the product of the size of the controlling sets and the average fan-in of the circuit. These techniques enable us to compute the “FREE” connections with minimal extra cost during a single graph traversal.

## 2.2 Experimental Results

We performed a number of experiments to evaluate the impact of our main theorem on the effectiveness of global-flow optimization. We did this by creating two programs, one of which used our main theorem and one of which relied on Theorem 1 of [2]. Both programs utilized the theorems through the artifact of derived graphs. The method used to construct the derived graphs was somewhat different from that described in [2]; however, identical methods were used in both programs. In addition, because of the computational constraints alluded to above, we did not compute the entire fixed point of the recurrences; rather, we weakened the recurrences by dropping the term corresponding to the contrapositive throughout the experiment. When computing the results based on [2], we weakened the recurrences even more by including only the term corresponding to forward propagation. The result of these two approximations can only exaggerate the possible benefit due to our main theorem, and therefore, our results should be considered an upper bound on the usefulness of this idea. We feel strongly that other experiments are needed to evaluate the benefit which might be gained by utilizing the contrapositive or even using the entire  $F_{ij}$  sets defined in [2].

Our experiments began with logic which had been run through the high-level and AND/OR-level optimizations of LSS and then translated to NORs. (See [4, 8] for details about these optimizations.) We then ran one of the two programs: FREEOPT, which utilized our main theorem, or WEAKOPT, which was based on our earlier methods. These were followed by some “tidying up” programs which propagate constants, remove common sub-expressions, eliminate double negatives, etc.. We did not perform fan-in or fan-out correction.

Our most surprising result was that in one of the 15 cases for which comparisons were run, the size of the NOR-level circuit produced by WEAKOPT was smaller than that produced by FREEOPT. This must be due to the interaction of successive applications of the optimization, and suggests that the effect of signal ordering in the sequences of applications is important. Currently, we treat high fan-out signals first.

Other than this case, results were in accordance with our expectations. In many cases the two methods were identical. In those where there was a difference, there was a wide spread in effect; the improvement varied from 25% to 100% in the ratio of the number of connections removed by each program. This large variance is not too surprising; it suggests that if the style of specification is such that the main theorem applies, it may apply frequently.

### 2.3 Other Applications

As mentioned earlier, the refinement in our formalism extends the applicability of our method to conditional optimizations. This permits us to make use of "don't care" conditions directly. We do this by adding a new function which recognizes the appropriate "care" set to the circuit and a new pair setting the output of this function to '1' to partial assignments used for optimization. The resulting optimizations will be valid on all inputs in the "care" set. Both output "don't cares" and redundancy removal can be accommodated in this network.

## 3. Summary and Conclusion

In this paper we describe two enhancements to our earlier global-flow algorithm for connection reduction. We show how to make better use of all types of "don't care" information, and we show how to apply our methods to redundancy removal optimizations. From a practical standpoint, these enhancements result in superior performance for the algorithm. From a conceptual standpoint, they unify a wide variety of optimizations which have been part of the Logic Synthesis System.

The work presented suggests a number of avenues for further research:

- 1 Develop an incremental or on-line algorithm which maintains the full controlling sets
- 2 Determine the effect of using the forcing sets or the full controlling sets in global flow (See [9] for a beginning)
- 3 Investigate the effect of signal ordering on global flow.

## Acknowledgments

We would like to thank Larry Carter and Andrea LaPaugh, without whom this paper might have been written but could never have been read.

## References

- [1] F.E. Allen and J. Cocke, "A Program Data Flow Analysis Procedure", *CACM*, vol. 19, no. 3, pp. 137-147, March, 1976.
- [2] L. Berman and L. Trevillyan, "A Global Approach to Circuit Size Reduction", *Advanced Research in VLSI, 5th MIT Conference*, pp. 203-214, Cambridge, MA: MIT Press, March 28-30, 1988.
- [3] L. Berman, L. Trevillyan and D. Brand, "Applications of Global Flow Analysis in Logic Synthesis", *Proceedings of 1988 Int. Symp. on Circuits and Systems*, Helsinki, Finland, Jun 7-9, 1988.
- [4] D. Brand, "Logic Synthesis", *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, pp. 301-326, Martinus Nijhoff, 1987.
- [5] R.K. Brayton, "Algorithms for Multi-Level Logic Synthesis and Optimization", *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, pp. 197-248, Martinus Nijhoff, 1987.
- [6] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System", *IEEE Trans. on CAD*, vol. CAD-6, no. 6, November 1987.
- [7] R.K. Brayton, E.M. Sentovich and F. Somenzi, "Don't Cares and Global Flow Analysis of Boolean Networks", *Proceedings of the ICCAD*, November 1988.
- [8] J.A. Darringer, D. Brand, J.V. Gerbi, W.H. Joyner and L.H. Trevillyan, "LSS: A System for Production Logic Synthesis", *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 272-280, July, 1981.
- [9] G. Hachtel, R. Jacoby, P. Moceynas and C. Morrison, "Performance Enhancements in BOLD using "Implications""", *Proceedings of the ICCAD*, November 1988.
- [10] L. Trevillyan, W. H. Joyner, Jr, and C.L. Berman, "Global Flow Analysis in Automatic Logic Design", *IEEE Trans. on Computers*, vol C-25, no. 1, January 1986.

# A METHOD FOR CONCURRENT DECOMPOSITION AND FACTORIZATION OF BOOLEAN EXPRESSIONS\*

Jagadeesh Vasudevamurthy<sup>1</sup> and Janusz Rajski<sup>2</sup>

*VLSI Design Laboratory*

*Department of Electrical Engineering*

*McGill University, 3480 University Street*

*Montreal, Canada H3A 2A7*

## Abstract

This paper describes efficient algorithms for decomposition and factorization of Boolean expressions. The method uses only two-literal single-cube divisors and double-cube divisors considered concurrently with their complementary expressions. We demonstrate that these objects, despite their simplicity, provide a very good framework to reason about common algebraic divisors and the duality relations between expressions. The algorithm has been implemented and excellent results on several benchmark circuits illustrate its efficiency and effectiveness.

## 1. Introduction

Multi-level logic synthesis is one of the key problems in automatic synthesis of digital circuits [1, 2, 4, 6, 8]. The basic operations involved in multi-level synthesis are *decomposition* and *factoring*. Decomposition [6] of a network refers to identifying subexpressions common to one or more functions such that they can be implemented once and shared across the entire design. Factoring refers to representing a logic function in a minimum form using parenthesis with the least number of literals. The operation of decomposition is similar to factoring except that each subexpression is formed as a new intermediate variable and substituted into the functions being decomposed. Both the decomposition and factoring concern with identification of common subexpressions and rewriting logic functions in factored form. A factored form is a parenthesized algebraic expression that has many attractive properties [6].

The basic problem in decomposition and factoring is the identification of frequently used common subexpressions. Sharing of such expressions across the entire design reduces the complexity of the synthesized network. Dietmeyer and Su [7] proposed a recursive top down factoring algorithm: Given  $F$ , the method

---

\*This work was supported by strategic grant MEF0045788 from the "Microelectronics Fund of Natural Sciences and Engineering Research Council of Canada".

<sup>1,2</sup>Authors are currently with <sup>1</sup>Symplicity, Inc. and <sup>2</sup>Mentor Graphics Corporation.

computes common factors which are single-cube divisors. The main disadvantage of this method is that common factors which consist of more than one cube (multiple cube divisors) are not considered.

Brayton and McMullen [5, 6] proposed a bottom up technique for detecting common subexpressions. They introduced the notion of *kernels* in algebraic expressions and showed how to use kernels to find multiple cube factors, which are common to two or more expressions. These techniques, with minor modifications, are used in all the systems [1, 2, 4, 6, 8] for global optimization. It is believed that through the intersection of sets of kernels, it is possible to find common subexpressions that would be just as good a group of candidates for divisors, as those obtained from intersecting the complete sets of all algebraic divisors. Although the set of kernels of an expression is usually smaller than the set of all its algebraic divisors, the number of kernels of an expression can grow exponentially in the number of literals in the support of the expression [6]. The computation to find the intersections of all kernels is not trivial, since the problem of detecting intersections in a set of kernels, and detecting common subcubes in a set of functions, are computationally equivalent to finding the kernels of an expression. Due to these facts, the advantage of finding common multiple cubes by the intersection of kernels, is nullified in many big practical examples. The algorithm [6] is greedy and selects a kernel with greatest *value*, and the kernel is substituted in the network. However, after one kernel is selected and substituted, it is possible that the value of the remaining kernels may change. The recomputation of kernels after every substitution is very expensive. Hence the method uses the set of kernels up to a certain point, where the value of kernels become inaccurate. This simplification does not guarantee that the best kernel is always selected and therefore the method is not entirely greedy.

Our approach is to identify and extract useful multiple-cube divisors and single-cube divisors concurrently with their complements, that together provide the greatest cost reduction in terms of the number of literals. In order to make the extraction process very simple, and to ensure that the operations are in the polynomial time domain, the method during any iteration looks at objects which are either multiple-cube divisors having exactly two cubes called double-cube divisors or single-cube divisors having exactly two literals. Using objects of size two, whose numbers are always in the polynomial domain, algebraic divisors of arbitrary size and their complements are obtained. The relationships between double-cube divisors, single-cube divisors, algebraic divisors and their complements are studied in this paper. These properties are exploited in designing efficient concurrent decomposition algorithm.

Based on the above concepts a program called Pendulum has been implemented. Excellent results on several benchmark circuits [3, 10] illustrate the effectiveness and efficiency of our algorithm in terms of the literal count of the synthesized multi-level logic and the CPU time required by the algorithm.

## 2. Definitions

The definitions used in this paper are consistent with [5]. A *variable* is a symbol representing a single coordinate of Boolean space. A *literal* is a variable or its negation. A *cube* is a set  $C$  of literals such that  $x \in C$  implies  $\bar{x} \notin C$ . A cube represents the conjunction of the literals. A *sum of products* (SOP) form, also called an expression, is a set of cubes. An expression represents the disjunction of its cubes. Given two Boolean expressions  $f$  and  $g$ ,  $g$  is called an *algebraic divisor* of  $f$  if  $f = gh + r$ , where  $h$  and  $r$  are expressions and  $h$  is not null. Also  $\text{sup}(g) \cap \text{sup}(h) = \emptyset$ . If  $g$  has exactly one cube, then  $g$  is called a *single-cube divisor*. If  $g$  has more than one cube, then  $g$  is called a *multiple-cube divisor*. For example, if  $f = abc + abd + p$ ,  $g = ab$  is called a single-cube divisor and  $g = bc + bd$  is called a multiple-cube divisor. A Boolean expression  $f$  is *cube-free*, if the only cube dividing  $f$  evenly is 1. Note that a cube-free expression must have more than one cube. For example,  $ab + c$  is cube-free but not  $ab + ac$  and  $abc$ . *Double-cube divisors* of a Boolean expression  $f$  are cube-free, multiple-cube divisors having exactly two cubes.

The set of all double-cube divisors of  $f$  is written as  $D(f)$ , where

$$D(f) = \{d | d = \{c_i \setminus (c_i \cap c_j), c_j \setminus (c_i \cap c_j)\}\} \quad (1)$$

for  $i, j = 1, \dots, n$ ,  $i \neq j$ , where  $n$  is the number of cubes in  $f$ .

For example, let  $f = ade + ag + bcde + bcg$ . The double-cube divisors are  $de + g$ ,  $a + bc$ ,  $ade + bcg$  and  $ag + bcde$ . It is evident that the total number of double-cube divisors for a Boolean function with  $n$  cubes is  $O(n^2)$ .  $(c_i \cap c_j)$  is called the *base* of cubes  $c_i$  and  $c_j$  and the double-cube divisor definition allows double-cube divisors with an empty base.

## 3. Properties of double-cube divisors

As the name indicates, double-cube divisors have exactly two cubes. A subset of double-cube divisors is represented by  $D_{x,y,s}$ , where  $x$  is the number of literals in the first cube,  $y$  is the number of literals in the second cube and  $s$  is the number of variables in the support of any double-cube divisor  $d$ . Note that  $\max(x, y) \leq s \leq (x + y)$ . For any  $d \in D_{x,y,s}$ ,  $d \in D_{y,x,s}$  and hence without loss of generality, we can assume  $x \leq y$ . For example, a double-cube divisor  $xy + \bar{y}zp \in D_{2,3,4}$ .

Let  $S$  denote the set of all single-cube divisors. A subset of single-cube divisors is denoted by  $S_k$ , where  $k$  is the number of literals in the single-cube divisor. For example, a single-cube divisor  $ab \in S_2$ .

With this notation, some useful relations between single-cube divisors, double-cube divisors and complements among them are given below. We assume, in the discussion that follows, that double-cube divisors are extracted from functions which are prime and irredundant with respect to every output. For proofs, readers are referred to [9].

**Lemma 1.**  $D_{1,1,1}$  and  $D_{1,2,2}$  are null set.

**Lemma 2.** For any  $d \in D_{1,1,2}$ ,  $\bar{d} \in S_2$ .

**Example.** The complement of  $a + \bar{b}$  is  $\bar{a}\bar{b} \in S_2$ .

**Lemma 3.** For any  $d \in D_{1,2,3}$ ,  $\bar{d} \notin D$ .

**Example.**  $a + \bar{b}c \in D_{1,2,3}$ . Its complement is  $\bar{a}\bar{b} + \bar{a}\bar{c}$ . Since  $D$  is a set of cube-free divisors.  $\bar{a}\bar{b} + \bar{a}\bar{c}$  cannot be in  $D$ .

**Lemma 4.** For any  $d \in D_{2,2,2}$ ,  $d$  is either exclusive-or or exclusive-nor expression and  $\bar{d} \in D_{2,2,2}$ .

**Example.**  $a\bar{b} + b\bar{a} \in D_{2,2,2}$ . Its complement is  $a\bar{b} + \bar{a}\bar{b} \in D_{2,2,2}$ .

**Lemma 5.** For any  $d \in D_{2,2,3}$ ,  $\bar{d} \in D_{2,2,3}$ .

**Example.**  $\bar{a}\bar{b} + ac \in D_{2,2,3}$ . Its complement is  $\bar{a}\bar{b} + a\bar{c} \in D_{2,2,3}$ .

**Lemma 6.** For any  $d \in D_{2,2,4}$ ,  $\bar{d} \notin D$ .

Given two Boolean expressions  $f$  and  $g$ , the important task during decomposition is to establish whether (a) function  $f$  has a complement cube divisor in  $g$  and (b) function  $f$  has a common cube-divisor in  $g$ . Theorems 1 and 2, below, establish such relations between two expressions.

**Theorem 1 [9].** Let  $f$  and  $g$  be two Boolean expressions. Then if

- (a)  $d_i \neq \bar{s}_j$  for every  $d_i \in D_{1,1,2}(f)$ ,  $s_j \in S_2(g)$ , and
- (b)  $d_i \neq \bar{d}_j$  for every  $d_i \in D_{exor}(f) \subseteq D_{2,2,2}(f)$ ,  $d_j \in D_{exnor}(g) \subseteq D_{2,2,2}(g)$ , and
- (c)  $d_i \neq \bar{d}_j$  for every  $d_i \in D_{exnor}(f) \subseteq D_{2,2,2}(f)$ ,  $d_j \in D_{exor}(g) \subseteq D_{2,2,2}(g)$ , and
- (d)  $d_i \neq \bar{d}_j$  for every  $d_i \in D_{2,2,3}(f)$ ,  $d_j \in D_{2,2,3}(g)$ , and
- (e)  $d_i \neq \bar{s}_j$  for every  $d_i \in D_{1,1,2}(g)$ ,  $s_j \in S_2(f)$ ,

then  $f$  has neither a complement double-cube divisor nor a complement single cube-divisor in  $g$ .

One of the important tasks during the decomposition is to detect whether two or more expressions have any common algebraic divisors other than single cubes. A theorem was presented in [11, 5, 6] using the concept of intersection

of set of kernels. This theorem is used for detecting if two or more expressions have any common algebraic divisors other than single cubes. The method is to compute the set of kernels for each logic expression, and forming nontrivial (more than one term) intersections among kernels from different functions. If this intersection set is empty, it is certain that there are no common multiple cubes between expressions. This theorem is important, since the set of all kernels is much smaller than the set of all algebraic divisors. We will now demonstrate that the same argument is valid for detecting common multiple cube from a set of double-cube divisors rather than a set of kernels. This should lead to an improved run time efficiency, since the set of all double-cube divisors is much smaller than the set of all kernels.

**Theorem 2.** Expressions  $f$  and  $g$  have a common multiple-cube divisors if and only if  $D(f) \cap D(g) \neq 0$ .

**Proof.** **If** If  $D(f) \cap D(g) \neq 0$ , then there exists a  $d \in D(f) \cap D(g)$  which is a double-cube divisor that divides both  $f$  and  $g$ .

**Only if** Now suppose, that  $f$  and  $g$  have a common multiple-cube divisor  $C$ ,  $C|f, C|g$ . ( $C|f$  denotes  $C$  divides  $f$ .) Let  $C = \{c_1, c_2, \dots, c_m\}$ . Take any  $e = \{c_i, c_j\}$  such that  $c_i, c_j \in C$ . If  $e$  is cube-free, then  $e \in D(f) \cap D(g)$ . If  $e$  is not cube-free, then  $e' = \{c_i \setminus (c_i \cap c_j), C_j \setminus (c_i \cap c_j)\}$  exists since  $f$  and  $g$  are non-redundant. Hence  $e' \in D(f) \cap D(g)$ . Q.E.D

## 4. Concurrent Decomposition Procedure

The basic objects used in the decomposition of Boolean equations are single-cube divisors having exactly two literals, double-cube divisors, and their complements. These two objects help us to find single-cube divisors of arbitrary sizes, and multiple-cube divisors. With each of the generated double-cube divisors and single-cube divisors, a weight function is associated which indicates the number of literals that can be saved, if this divisor is selected. The complement of the double-cube divisor is also considered in the weight computation. The concurrent, greedy decomposition method is given in Algorithm 1 [9], which identifies and extracts either a double-cube divisor jointly with its dual expression, or a single-cube divisor that provides the greatest cost reduction in terms of the total number of literals.

## 5. Experimental Results

Based on the above proposed algorithm a program called Pendulum has been implemented in C under UNIX. The largest PLAs from [10, 3] were selected as benchmark circuits to measure the performance of the algorithm. All the PLA's were minimized using the logic minimizer ESPRESSO [3] before decomposi-

tion. The resulting synthesized circuits were verified using the *verify* option of MISII [6]. Columns 2 and 3 of Table 1 summarizes the results obtained using Pendulum and compares to those generated with MISII (both run on the same SUN 3/260) using the script given in Figure 1.

```

time
sweep
eliminate 5
simplify -m nocomp -d
resub -a
gkx
resub -a; sweep
gcx
resub -a; sweep
gkx
resub -a; sweep
gcx
resub -a; sweep
gkx
resub -a; sweep
gcx
resub -a; sweep
eliminate 0
decomp -g *
print_stats -f
time

```

*Figure 1.*

Note that only “gkx” and “gcx” commands are used to run MIS without any expensive options. These scripts ensure that only level-0 kernels [6] are used and the more efficient “ping-pong” algorithm [11] is used to find a good (but not necessarily the best) kernel intersection and single-cube divisor. These options, in MIS, take less CPU time compared to the standard script which finds all the kernels and then chooses the best kernel. The effectiveness of the method can be summarized by the total literal count and the total time taken to synthesize the circuits. These numbers are 8,871 literals and 6,076 CPU seconds for Pendulum and 11,488 literals and 53,648 CPU seconds for MISII. For these benchmarks, Pendulum synthesized circuits with, on the average, 20 percentage fewer literals and using one ninth of CPU time required by MISII. The results for MIS2.1 were reported in the MCNC 1989 workshop poster session [10] and are given in the last two columns of Table 1. *Single* is the *factored form* literal count using a single execution of a standard script. *Best* is the best factored form literal count using several executions of a standard script. It can be seen that for large benchmark circuits like *apex1*, *apex3*, *apex4*, *apex5* and *seq* Pendulum produces smaller networks than that of the best results obtained by MIS and for *9sym*,

	Final literals in SOP		CPU seconds SUN 3/260		Literal count in factored-form			
	pla	Pend <ul style="list-style-type: none">ulum</ul>	MIS 2.1*	Pend <ul style="list-style-type: none">ulum</ul>	MIS 2.1*	Pend <ul style="list-style-type: none">ulum</ul>	MIS2.1 Single**	MIS2.1 Best**
5xp1	163	186	1.7	32.0	150	114	104	
9sym	263	255	95.5	484.7	225	229	216	
add6	85	104	220	1087	83	-	-	
apex1	1101	1268	75	990	1065	1247	1247	
apex2	452	540	4498	37509	426	278	246	
apex3	1381	1876	184.5	2410	1380	1450	1401	
apex4	1976	2594	346.9	4512	1924	2592	2592	
apex5	893	1112	183	507	849	908	890	
chkn	469	452	184.2	289	408	-	-	
duke2	435	493	13.0	82.5	420	442	393	
e64	253	253	2.7	140.3	253	253	253	
rd73	106	117	11.8	294.8	98	90	74	
rd84	143	446	136.6	784.6	135	148	124	
sao2	171	213	4.9	40	149	122	118	
seq	948	1547	63.8	2059	896	1176	1176	
xor9	32	32	55	2427	32	-	-	
TOTAL	8871	11488	6076.6	53648.9				

\* Results obtained by using the script given in Figure 1

\*\*Results reported in MCNC '89 workshop poster session

- Results not available

Table 1. Summary of experimental results.

*duke2*, *e64* and *rd84* Pendulum produces smaller networks than that of the single execution of MIS. The effectiveness of our approach is due to the fact that Pendulum works with divisors and their complements which are in polynomial time domain. These divisors are extracted only once and updated incrementally as the synthesis proceeds. The duality relation between various intermediate nodes is also determined immediately during the synthesis. The advantage of the method is the speed over the MIS approach and the ability to compute the weights of potential divisors more accurately and dynamically during decomposition and factorization. Either single-cubes having two literals or double-cube divisors are selected at each step, which provides an added advantage over performing the "gcx" and "gkx" commands separately as done in MIS. As the cube divisors are extracted concurrently based on the cost reduction they yield, no resubstitution or simplification is required.

## 6. Conclusions

We presented a new, very efficient method for decomposition and factorization of Boolean expressions. The method proposed uses only two-literal single-

cube divisors or double-cube divisors concurrently with consideration for use of complements. The advantage of this approach is the speed over the methods based on kernels [5] and the ability to compute the weights of potential divisors more accurately and dynamically during decomposition and factorization. Either single-cubes having two literals or double-cube divisors are selected at each step, which provides an added advantage over extracting multiple-cube divisors and single-cube divisors separately as done in [6]. In fact, the results obtained by this method match the best known literal counts for most benchmark circuits, in many cases the method generates Boolean networks with much smaller number of literals. In another paper [9], we demonstrate, both theoretically and experimentally, that the decomposition and factorization transformations introduced in this paper preserve testability, which implies that a complete test set developed for an input network gives also complete coverage of faults in the synthesized multi-level network.

## References

- [1] Bartlett, K., W. Cohen, A. de Geus, and G. Hatchel (Oct 1986). "Synthesis and optimization of multilevel logic under timing constraints," *IEEE Trans. on Computer-Aided Design*, Vol. 5, no. 4, pp. 582–596.
- [2] Bostick, D., and G. D. Hatchel (1987). "The Boulder optimal logic design system," in *Proc. of the Int. Conf. on Computer Aided Design*, pp. 62–65.
- [3] Brayton, R. K., G. D. Hatchel, C. McMullen, and A. Sangiovanni-vincentelli (1984). "Logic Minimization Algorithms for VLSI Synthesis," Kluwer Academic Publishers.
- [4] Brayton, R. K., N. Brenner, C. Chen, G. Hatchel, C. McMullen, and R. Otten (1985). "The Yorktown silicon compiler," in *Proc. of the Int. Symp. on Circuits and systems*, pp. 391–394.
- [5] Brayton, R. K., C. McMullen (1982). "The decomposition and factorization of Boolean expressions," in *Proc. of the Int. Symp. on Circuits and systems*, pp. 49–54.
- [6] Brayton, R. K., R. Rudell, A. Sangiovanni-vincentelli, and A. R. Wang (Nov 1987). "MIS: A multiple-level logic optimization system," *IEEE Trans. on Computer-Aided Design*, Vol. 6, no. 6 pp. 1062–1081.
- [7] Dietmeyer, D. L., and Y. H. Su (Jan 1969). "Logic design automation of fan in limited NAND networks," *IEEE Trans. on Computers*, Vol. C-18, no. 1, pp. 11–22.
- [8] Mathony, H., and U. G. Baitinger (March 1988). "CARLOS: An automated multilevel logic design system for CMOS semi-custom integrated circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 7, no. 3, pp. 346–355.
- [9] Rajski, J., and J. Vasudevamurthy (1990). "Testability preserving transformations in multi-level logic synthesis," in *International Test Conference*, accepted for publication.
- [10] "Introduction to Synthesis Benchmarks," in *Second international workshop on logic synthesis*, North Carolina, USA, May 23–26, 1989.
- [11] Brayton, R. K. (March 1987). "Factoring logic function," *I.B.M. Journal of Research and Development*, pp. 187–198.

# AN OPTIMAL TECHNOLOGY MAPPING ALGORITHM FOR DELAY OPTIMIZATION IN LOOKUP-TABLE BASED FPGA DESIGNS

Jason Cong and Yuzheng Ding<sup>1</sup>

*Department of Computer Science*

*University of California*

*Los Angeles, CA 90024*

## Abstract

In this paper we present a polynomial time technology mapping algorithm, called Flow-Map, that optimally solves the LUT based FPGA technology mapping problem for depth minimization for general Boolean networks. This theoretical breakthrough makes a sharp contrast with the fact that conventional technology mapping problem in library based designs is NP-hard. A key step in Flow-Map is to compute a minimum height  $K$ -feasible cut in a network, solved by network flow computation. Our algorithm also effectively minimizes the number of LUTs by maximizing the volume of each cut and by several postprocessing operations. We tested the Flow-Map algorithm on a set of benchmarks and achieved reductions on both the network depth and the number of LUTs in mapping solutions as compared with previous algorithms.

## 1. Introduction

The short design cycle and low manufacturing cost have made FPGA an important technology for VLSI ASIC designs. The LUT-based FPGA is a popular architecture used by several FPGA manufacturers, including Xilinx and AT&T [1, 2]. In an LUT-based FPGA chip, the basic programmable logic block is a  $K$ -input lookup table ( $K$ -LUT) which can implement any Boolean function of up to  $K$  variables. The technology mapping problem in LUT-based FPGA designs is to transform a general Boolean network (obtained by technology independent synthesis) into a functionally equivalent network of  $K$ -LUTs. This paper studies the LUT-based FPGA technology mapping problem for delay optimization.

The previous LUT-based FPGA mapping algorithms can be roughly divided into three classes. The algorithms in the first class emphasize on minimizing the number of LUTs in the mapping solutions [3, 4, 5, 6, 7, 8]; The algorithms in the second class emphasize on minimizing the delay of the mapping solutions

---

<sup>1</sup>Author is currently with Mentor Graphics Corporation.

[9, 10, 11]. The algorithms in the third class maximize the routability of the mapping solutions [12, 13]. Although many of the existing mapping methods showed encouraging results, these methods are heuristic in nature, and there is no way to determine how far away the mapping solutions of these algorithms are from the optimal solution in terms of the number of LUTs or the depth of the LUT network.

This paper presents a theoretical breakthrough which shows that the LUT-based FPGA technology mapping problem for depth minimization can be solved optimally in polynomial time for general Boolean networks. A key step in our algorithm is to compute a *minimum height K-feasible cut* in a network, which is solved optimally in polynomial time based on efficient network flow computation. Our result makes a sharp contrast with the fact that the conventional technology mapping problem in library-based designs is NP-hard for general Boolean networks [14, 15]. Due to this inherent difficulty, most conventional technology mapping algorithms decompose the input network into a forest of trees and then map each tree optimally [14, 15]. Such a methodology was also used in some existing FPGA mapping algorithms [3, 16]. However, our result shows that optimal solutions can be produced efficiently for *general* Boolean networks in LUT-based FPGA technology mapping for depth minimization.

## 2. Problem Formulation and Preliminaries

A Boolean network can be represented as a directed acyclic graph (DAG) where each node represents a logic gate, and a directed edge  $(i, j)$  exists if the output of gate  $i$  is an input of gate  $j$ . A primary input (PI) node has no incoming edge and a primary output (PO) node has no outgoing edge. We use  $\text{input}(v)$  to denote the set of fanins of gate  $v$ . Given a subgraph  $H$  of the Boolean network,  $\text{input}(H)$  denotes the set of *distinct* nodes outside  $H$  which supply inputs to the gates in  $H$ . For a node  $v$  in the network, a *K-feasible cone at v*, denoted  $C_v$ , is a subgraph consisting of  $v$  and its predecessors such that any path connecting a node in  $C_v$  and  $v$  lies entirely in  $C_v$ , and  $|\text{input}(C_v)| \leq K$ . The *level* of a node  $v$  is the length of the longest path from any PI node to  $v$ . The level of a PI node is zero. The *depth* of a network is the largest node level in the network. A Boolean network is *K-bounded* if  $|\text{input}(v)| \leq K$  for every node  $v$ .

We assume that each programmable logic block in an FPGA is a *K-LUT* that can implement any  $K$ -input Boolean function. Thus, each *K-LUT* can implement any *K*-feasible cone of a Boolean network. The technology mapping problem for *K-LUT* based FPGAs is to cover a given Boolean network with *K*-feasible cones. (Note that we allow these cones to overlap, which means that certain nodes in the original network can be duplicated when generating *K-LUTs*.) A technology mapping solution  $S$  is a DAG where each node is a *K*-feasible cone (equivalently, a *K-LUT*) and the edge  $(C_u, C_v)$  exists if  $u$  is in

$\text{input}(C_v)$ . Our main objective is to compute a mapping solution that results in the minimum delay. The *unit delay model* is used where the delay is determined by the depth of the mapping solution. We say that a mapping solution is *optimal* if its depth is minimum. The main objective of our algorithm is to find an optimal mapping solution; the secondary objective is to reduce the number of  $K$ -LUTs used in the solution.

Several important concepts about *cuts* in a network will be used in this paper. Given a network  $N = (V(N), E(N))$  with a source  $s$  and a sink  $t$ , a *cut*  $(X, \bar{X})$  is a partition of the nodes in  $V$  such that  $s \in X$  and  $t \in \bar{X}$ . The *node cut-size* of  $(X, \bar{X})$ , denoted as  $n(X, \bar{X})$ , is the number of nodes in  $X$  that are adjacent to some node in  $\bar{X}$ , i.e.

$$n(X, \bar{X}) = |\{x : (x, y) \in E, x \in X \text{ and } y \in \bar{X}\}|.$$

A cut  $(X, \bar{X})$  is  *$K$ -feasible* if its node cut-size is no more than  $K$ , i.e.,  $n(X, \bar{X}) \leq K$ . Assume that each edge  $(u, v)$  has a non-negative capacity  $c(u, v)$ . Then, the *edge cut-size* of  $(X, \bar{X})$ , denoted  $e(X, \bar{X})$ , is the sum of the capacities of the edges that cross the cut, i.e.

$$e(X, \bar{X}) = \sum_{u \in X, v \in \bar{X}} c(u, v).$$

Throughout this paper, we assume that the capacity of each edge is one unless specified explicitly. The *volume* of a cut  $(X, \bar{X})$ , denoted  $\text{vol}(X, \bar{X})$ , is the number of nodes in  $\bar{X}$ , i.e.,  $\text{vol}(X, \bar{X}) = |\bar{X}|$ . Moreover, assume that there is a given label  $l(v)$  associated with each node  $v$ . Then, the *height* of a cut  $(X, \bar{X})$ , denoted  $h(X, \bar{X})$ , is defined to be the maximum label in  $X$ , i.e.

$$h(X, \bar{X}) = \text{MAX}\{l(x) : x \in X\}.$$

Figure 1 shows a cut  $(X, \bar{X})$  in a network with given node labels, where  $n(X, \bar{X}) = 3$ ,  $e(X, \bar{X}) = 10$ ,  $h(X, \bar{X}) = 2$ , and  $\text{vol}(X, \bar{X}) = 9$ .

### 3. An Optimal LUT-Based FPGA Mapping Algorithm for Depth Minimization

Our algorithm is applicable to any  $K$ -bounded Boolean network. Given a general Boolean network as input, if it is not  $K$ -bounded, there are a number of ways to transform it into a  $K$ -bounded network. For example, the Roth-Karp decomposition [17] was used in [5] to obtain a  $K$ -bounded network. In our system, we first transform the given Boolean network into a network of simple gates (i.e. AND, OR, NAND, and NOR gate): We represent each complex gate in the sum-of-products form and then replace it with two levels of simple gates. Then, we transform the resulting Boolean network into a two-input Boolean network. There are two reasons for carrying out such a transformation. First, we want to limit the number of inputs of each gate to be no more than  $K$  so that

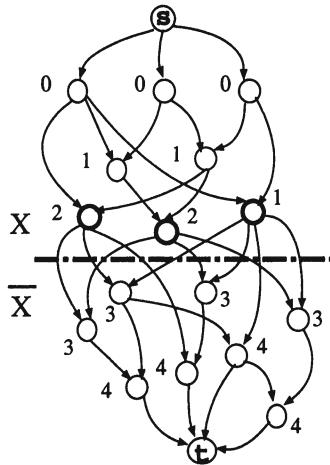


Figure 1. A 3-feasible cut of height 2.

we do not have to decompose gates during technology mapping. Second, if we think of FPGA technology mapping as a process of packing gates in a given network into  $K$ -LUTs, then, intuitively, smaller gates will be more easily packed, with less wasted space in each  $K$ -LUT. As proposed in [9, 18, 19], we use an algorithm based on the Huffman coding tree construction to decompose each multiple input simple gate into a tree of two-input simple gates. According to the result in [9], such a decomposition procedure increases the network depth by at most a small constant factor. Although our system transforms the original network into a network of two-input simple gates, the optimality of our algorithm does not depend on the fact that each node in the Boolean network is a two-input simple gate. The optimality of our mapping result holds as long as the input network is  $K$ -bounded, in which the gates need not to be simple.

Our optimal mapping algorithm, named Flow-Map, runs in two phases. In the first phase, it computes a label for each node which reflects the level of the  $K$ -LUT implementing that node in an optimal mapping solution. In the second phase, it generates the  $K$ -LUT mapping solution based on the node labels computed in the first phase. Due to the length restriction of the paper, the results in this section and next section are stated without proof. The proofs of these results can be found in [20].

### 3.1 The Labeling Phase

Given a  $K$ -bounded Boolean network  $N$ , let  $N_v$  denote the subnetwork consisting of node  $v$  and all the predecessors of  $v$ . We define the *label* of  $v$ , denoted

as  $l(v)$ , to be the depth of the optimal  $K$ -LUT mapping solution of  $N_v$ . Clearly, the level of the  $K$ -LUT containing  $v$  in the optimal mapping solution of  $N$  is at least  $l(v)$ , and the maximum label of all the POs of  $N$  is the depth of the optimal mapping solution of  $N$ . The first phase of our algorithm computes the labels of all the nodes in  $N$ , according to the topological order starting from the PIs. For each PI node  $v$ , we assign  $l(v) = 0$ . Suppose  $t$  is the current node being processed. Then, for each node  $v \neq t$  in  $N_t$ , the label  $l(v)$  has been computed. By including in  $N_t$  an auxiliary node  $s$  and connecting  $s$  to all the PI nodes in  $N_t$ , we obtain a network with  $s$  as the source and  $t$  as the sink. For simplicity we still denote it as  $N_t$ . Figure 2(a) shows part of a Boolean network in which gate  $t$  is being labeled, and Figure 2(b) shows the construction of the network  $N_t$ . Let  $LUT(t)$  be the  $K$ -LUT that implements node  $t$  in an optimal  $K$ -LUT mapping solution of  $N_t$ , and let  $\bar{X}$  denote the set of nodes in  $LUT(t)$  and  $X$  denote the remaining nodes in  $N_t$ . It is easy to see that  $(X, \bar{X})$  forms a  $K$ -feasible cut between  $s$  and  $t$  in  $N_t$  because the number of inputs of  $LUT(t)$  is no more than  $K$ . Moreover, let  $u$  be the node with the maximum label in  $X$ , then, the level of  $LUT(t)$  is  $l(u) + 1$  in the optimal mapping solution of  $N_t$ . Recall the definition of the height of a cut in Section 2, we have  $h(X, \bar{X}) = l(u)$ . Therefore, in order to minimize the level of  $LUT(t)$  in the mapping solution of  $N_t$ , we want to find a *minimum height K-feasible* cut  $(X, \bar{X})$  in  $N_t$ . (We exclude the cuts  $(X, \bar{X})$  where  $\bar{X}$  contains a PI node. Our algorithm to be shown later on guarantees that such kind of cuts are not generated.) In other words,

$$l(t) = \text{MIN}_{(X, \bar{X}) \text{ is } K\text{-feasible}} h(X, \bar{X}) + 1.$$

Figures 2(b) and 2(c) illustrate our labeling method. Since in 2(b) we have a minimum height 3-feasible cut in  $N_t$  whose height is 1,  $t$  is labeled 2, and the optimal  $K$ -LUT mapping solution of  $N_t$  is shown in Figure 2(c).

There was no known polynomial time algorithm for computing a minimum height  $K$ -feasible cut. One important contribution of our work is that we have developed an  $O(Km)$  time algorithm for computing a minimum height  $K$ -feasible cut in  $N_t$ , where  $m$  is the number of edges in  $N_t$ .

First, we show that the node labels defined by our labeling scheme satisfy the following property.

**Lemma 1** *The label of  $t$  satisfies  $l(t) = p$  or  $l(t) = p + 1$ , where  $p$  is the maximum label of the nodes in  $\text{input}(t)$ .*

According to Lemma 1, our algorithm first checks if there is a  $K$ -feasible cut  $(X_t, \bar{X}_t)$  of height  $p - 1$  in  $N_t$ . If there is such a cut, we assign  $l(t) = p$  and node  $t$  can be packed with the nodes in  $\bar{X}_t$  into the same  $K$ -LUT in the second phase of our algorithm. Otherwise, the minimum height of the  $K$ -feasible cuts in  $N_t$  is  $p$  and  $(V(N_t) - \{t\}, \{t\})$  is such a cut. In this case, we assign  $l(t) = p + 1$  and we shall use a new  $K$ -LUT for node  $t$ .

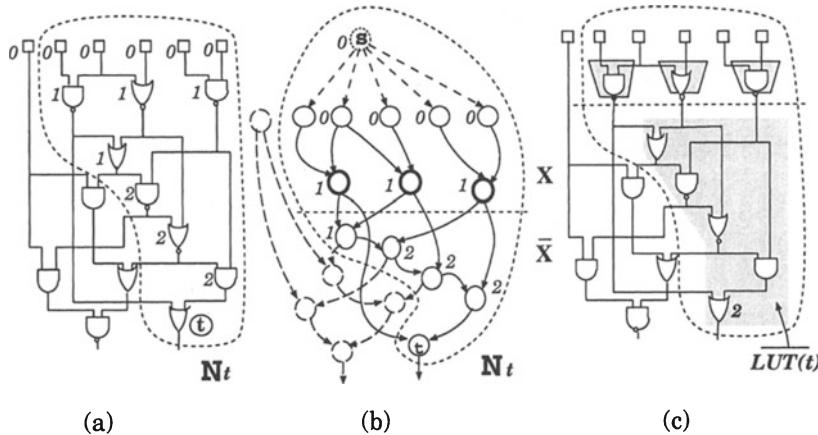


Figure 2. Computing the label  $l(t)$  of node  $t$  ( $K = 3$ ). (a) The partial network; (b) Construction of  $N_t$  and the highest 3-feasible cut; (c) Determining  $l(t)$ .

Whether  $N_t$  has a  $K$ -feasible cut of height  $p - 1$  or not can be tested efficiently using the following method. Let  $p$  be the maximum label of the nodes in  $N_t$ . We first apply a network transformation on  $N_t$  that collapses all the nodes in  $N_t$  with label  $\geq p$ , together with  $t$ , into the new sink  $t'$ . Let  $N'_t$  be the resulting network, we have the following result.

**Lemma 2**  $N_t$  has a  $K$ -feasible cut of height  $p - 1$  if and only if  $N'_t$  has a  $K$ -feasible cut.

For example, Figure 3(a) shows the network  $N_t$  for node  $t$  in Figure 2(a), and Figure 3(b) shows the induced network  $N'_t$ .

In order to determine if  $N'_t$  has a  $K$ -feasible cut, we apply another network transformation, which reduces the node cut-size constraint to an edge cut-size constraint by splitting nodes into edges. Specifically, we construct a new network  $N''_t$  from  $N'_t$  as follows. For each node  $v$  in  $N'_t$  other than  $s$  and  $t'$ , we introduce two nodes  $v_1$  and  $v_2$  and connect them by an edge  $(v_1, v_2)$  in  $N''_t$ , which is called a *bridging edge*. The source  $s$  and sink  $t'$  are also included in  $N''_t$  (with  $t'$  renamed as  $t''$ ). For each edge  $(s, v)$  in  $N'_t$ , there is an edge  $(s, v_1)$  in  $N''_t$ ; and for each edge  $(v, t')$  in  $N'_t$  there is an edge  $(v_2, t'')$  in  $N''_t$ . Moreover, for each edge  $(u, v)$  in  $N'_t$  ( $u \neq s$  and  $v \neq t'$ ), we introduce an edge  $(u_2, v_1)$  in  $N''_t$ . We assign the capacity of each bridging edge to be one, and the capacity of each non-bridging edge to be infinity. Figure 3(c) shows the resulting  $N''_t$  obtained from  $N'_t$  in Figure 3(b). Regarding this transformation, we have the following fact:

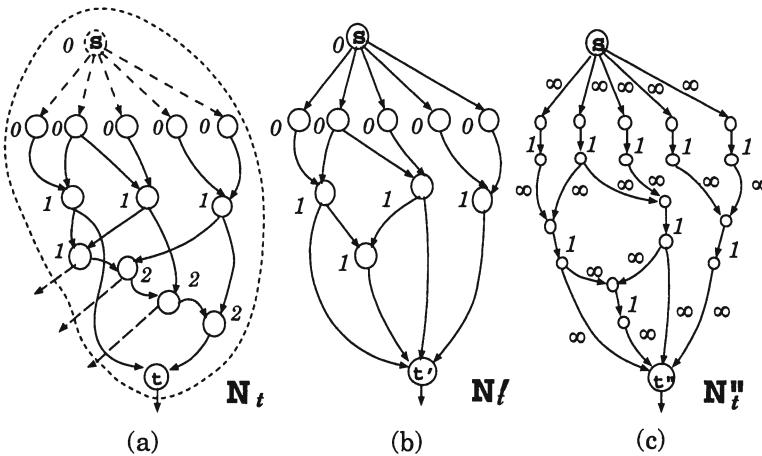


Figure 3. Network transformations in computing a minimum height 3-feasible cut in  $N_t$ .

**Lemma 3**  $N'_t$  has a  $K$ -feasible cut if and only if  $N''_t$  has a cut whose edge cut-size is no more than  $K$ .

According to the Max-flow Min-cut Theorem [21],  $N''_t$  has a cut whose edge cut-size is no more than  $K$  if and only if the maximum flow between  $s$  and  $t''$  in  $N''_t$  has a value of  $K$  or smaller. We apply the augmenting path algorithm in  $N''_t$  to compute a maximum flow. Since each bridging edge in  $N''_t$  has unit capacity, each augmenting path in the flow residual graph of  $N''_t$  from  $s$  to  $t''$  increases the flow by one unit. If we can find  $K+1$  augmenting paths, then  $N''_t$  has a maximum flow of a value larger than  $K$  and we can conclude that  $N''_t$  does not have a cut  $(X'', \bar{X}'')$  with  $e(X'', \bar{X}'') \leq K$ . Otherwise, the residual graph is disconnected before we find the  $(K+1)$ -th augmenting path, and the disconnected residual graph induces a cut of edge cut-size no more than  $K$ . Moreover, we can find such a cut  $(X'', \bar{X}'')$  by performing a depth first search starting at the source  $s$ , and including in  $X''$  all the nodes which are reachable from  $s$ . Since finding an augmenting path takes  $O(m)$  time, where  $m$  is the number of edges in the residual graph of  $N''_t$  (which is in the same order as the number of edges in  $N_t$ ), we can determine in  $O(Km)$  time whether  $N''_t$  has a cut of edge cut-size no more than  $K$  and find one if such a cut exists. Such a cut  $(X'', \bar{X}'')$  in  $N''_t$  induces a  $K$ -feasible cut  $(X', \bar{X}')$  in  $N'_t$ , which in turn gives a minimum height  $K$ -feasible cut  $(X, \bar{X})$  in  $N_t$ . (It is clear that for the resulting cut  $(X, \bar{X})$  in  $N_t$ ,  $\bar{X}$  does not contain any PI nodes since any outgoing edge of the source  $s$  in  $N''_t$  has infinite capacity.)

Based on the above discussions, we have

**Theorem 1** *A minimum height  $K$ -feasible cut in  $N_t$  can be found in  $O(Km)$  time where  $m$  is the number of edges in  $N_t$ .*

Applying Theorem 1 to each node in network  $N$  in our labeling algorithm, we have

**Corollary 1** *The labels of all the nodes in  $N$  can be computed in  $O(Kmn)$  time, where  $n$  and  $m$  are the number of nodes and edges in  $N$ , respectively.*

In the current LUT-based FPGA architecture, the typical value of  $K$  is 4 or 5. Moreover, if the number of fanins (or the fanouts) of each node in  $N$  is bounded by a constant (which is two in our implementation), we have  $m = O(n)$ . Therefore, the complexity of the labeling phase of our algorithm is  $O(n^2)$  in practice.

In fact, the result in Theorem 1 can be generalized to compute the minimum height  $K$ -feasible cut in a general network with arbitrary node labels. Details can be found in [20].

### 3.2 The Mapping Phase

The second phase of our algorithm is to generate the  $K$ -LUTs in the optimal mapping solution. Let  $L$  be the set of outputs which are to be implemented using  $K$ -LUTs. Initially,  $L$  contains all the PO nodes. We process the nodes in  $L$  one by one. For each non-PI node  $v$  in  $L$ , assume that  $(X_v, \bar{X}_v)$  is the minimum height  $K$ -feasible cut in  $N_v$  that we computed in the first phase by the labeling algorithm. We generate a  $K$ -LUT  $v'$  to implement the function of gate  $v$ , using the input signals from  $X_v$  to  $\bar{X}_v$ . That is, the  $K$ -LUT  $v'$  includes all the gates in  $\bar{X}_v$  and  $input(v') = input(\bar{X}_v)$ . (Since the cut is  $K$ -feasible, we have  $|input(\bar{X}_v)| \leq K$ .) Then, we update the set  $L$  to be  $(L - \{v\}) \cup input(v')$ . It is possible that a gate  $w$  belongs to both  $\bar{X}_v$  and  $\bar{X}_u$  for two different gates  $v$  and  $u$  in  $L$ . In this case, gate  $w$  is automatically replicated and is included in both  $v'$  and  $u'$ . It is also possible that no  $K$ -LUT is generated for a gate  $w$  since it has been completely covered by the  $K$ -LUTs generated for some of its successors. In general, a  $K$ -LUT has to be generated for a gate  $w$  if  $w$  belongs to  $input(v')$  of some  $K$ -LUT  $v'$  which has been generated.

The second phase ends when  $L$  consists of only PI nodes of the original network. It is clear that at the end of the execution we get a network of  $K$ -LUTs which is logically equivalent to the original network.

Combining the first and second phases, we have

**Theorem 2** *For any  $K$ -bounded Boolean network  $N$ , the Flow-Map algorithm produces a  $K$ -LUT mapping solution with the minimum depth in  $O(Kmn)$  time, where  $n$  and  $m$  are the number of nodes and edges in  $N$ .*

In our implementation,  $K = 5$  and  $m = O(n)$ , so the complexity of the Flow-Map algorithm is  $O(n^2)$  in practice.

## 4. Enhancement of the Flow-Map Algorithm for Area Optimization

The secondary objective of our technology mapping algorithm is to minimize the number of  $K$ -LUTs in the mapping solution. In Flow-Map, this is considered by maximizing the volume of each cut during the mapping process and by postprocessing operations for  $K$ -LUT reduction.

### 4.1 Maximizing Cut Volume During Mapping

From the discussion in the preceding section, for each node  $t$  in the input network  $N$ , the Flow-Map algorithm computes a minimum-height  $K$ -feasible cut  $(X, \bar{X})$  in  $N_t$  and the nodes in  $\bar{X}$  will be packed into the same  $K$ -LUT with  $t$  if a  $K$ -LUT is generated to implement  $t$ . In general, minimum-height  $K$ -feasible cut is not unique. Intuitively, the larger  $\text{vol}(X, \bar{X}) = |\bar{X}|$  is, the more nodes we can pack into a  $K$ -LUT, and the fewer  $K$ -LUTs we use in total. Therefore, our algorithm wants to maximize the volume of the cut that it finds at each node.

It can be shown that maximizing the volume of a  $K$ -feasible cut  $(X, \bar{X})$  in  $N_t$  is equivalent to maximizing the volume of the corresponding cut  $(X'', \bar{X}'')$  in  $N''_t$  [20]. Therefore, according to the algorithm presented in the previous section, we want to find a *min-cut* in  $N''_t$  (i.e. a cut  $(X'', \bar{X}'')$  with the minimum  $e(X'', \bar{X}'')$ ) of the maximum volume. (Since for every signal crossing the cut in  $N_t$  we need to generate the signal using a  $K$ -LUT, minimizing the node cut-size in  $N_t$  will also lead to reduction of the number of  $K$ -LUTs. Consequently, we look for a min-cut with maximum volume in  $N''_t$ , instead of any cut with  $e(X'', \bar{X}'') \leq K$ .)

First, we can show the following results.

**Lemma 4** *There is a unique maximum volume min-cut in any network. Moreover, if  $(X, \bar{X})$  is the maximum volume min-cut and  $(Y, \bar{Y})$  is another min-cut different from  $(X, \bar{X})$ , then  $X \subset Y$ .*

**Lemma 5** *Let  $R_f$  be the residual graph of a maximum flow  $f$ . Let  $X$  be the set of nodes in  $R_f$  reachable from the source  $s$ , and  $\bar{X}$  be the set of the remaining nodes. Then,  $(X, \bar{X})$  is a maximum volume min-cut.*

Combining Theorem 1 and these results, we have

**Theorem 3** *A maximum volume min-cut in  $N''_t$  can be found in  $O(Km)$  time, where  $m$  is the number of edges in  $N_t$ .*

Therefore, Flow-Map maximizes the number of gates covered by each  $K$ -LUT by maximizing the volume of each min-cut in  $N''_t$ . As a result, area minimization is also achieved in the depth optimal mapping of Flow-Map.

## 4.2 Postprocessing Operations for K-LUT Reduction

After obtaining a  $K$ -LUT mapping solution using the Flow-Map algorithm, we want to further reduce the number of  $K$ -LUTs used in the mapping solution without increasing the depth. In [9], two depth-preserving operations were developed to minimize the number of  $K$ -LUTs in the mapping solutions of DAG-Map. One is called *predecessor packing* and the other is called *gate decomposition*. Although these operations lead to substantial reduction in the number of  $K$ -LUTs, they take only local information into consideration during minimization.

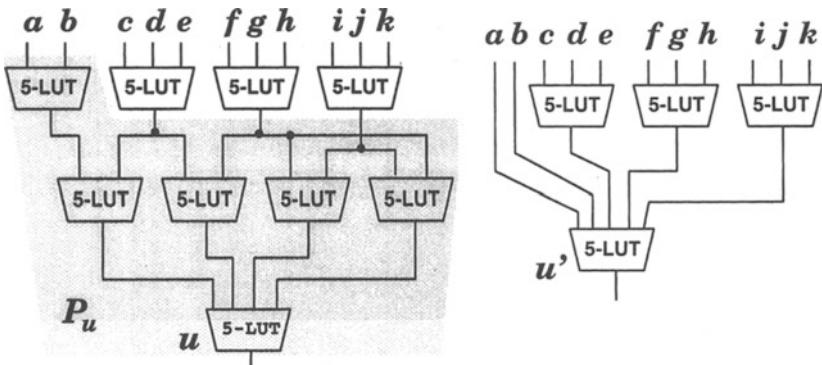


Figure 4. The flow-pack operation ( $K = 5$ ).

In this subsection, we introduce a new postprocessing operation called *Flow-Pack*. Given a  $K$ -LUT  $u$  in the mapping solution, it tries to pack a *set of predecessors* of  $u$  (including  $u$ ), denoted  $P_u$ , into a single  $K$ -LUT. (See Figure 4.) Clearly, we need to guarantee the condition that  $| \text{input}(P_u) | \leq K$  in order to carry out the packing. Let  $M$  be the current mapping solution and  $M_u$  be the subnetwork of  $M$  consisting of  $K$ -LUT  $u$  and all its predecessors. It is easily seen that  $P_u$  can be packed into a single  $K$ -LUT if and only if  $(V(M_u) - P_u, P_u)$  forms a  $K$ -feasible cut in  $M_u$ . Moreover, the larger  $| P_u |$  is, the more  $K$ -LUTs we reduce in the mapping solution  $M$ . Therefore, we want to find a  $K$ -feasible cut with the maximum volume in  $M_u$ .

Since no polynomial time optimal algorithm is known for the maximum volume  $K$ -feasible cut problem in a network, we have developed a heuristic algorithm to solve this problem. We start with a cut of the minimum node-cut size and maximum volume, which can be computed using the algorithm described

in the previous subsection. Then, we gradually increase the volume of the cut (and the node-cut size will increase accordingly as well) until the node-cut size exceeds  $K$ . The last  $K$ -feasible cut in the sequence is recorded as an approximate solution to the maximum volume  $K$ -feasible cut problem. The details of this algorithm can be found in [20]. The time complexity of the algorithm was shown to be  $O(K^3 m)$ .

Based on this approximation algorithm to the maximum volume  $K$ -feasible cut problem, the Flow-Pack operation is implemented as a part of postprocessing step of the Flow-Map package. During the postprocessing phase, we first carry out the matching based gate-decomposition operation as described in [9]. Then, we apply the Flow-Pack operation to each  $K$ -LUT  $u$  in the mapping solution so that  $u$  is collapsed with a maximal subset of its predecessors into a single  $K$ -LUT.

The advantage of the Flow-Pack operation is clear: the Flow-Pack operation takes the global information about the entire subnetwork  $M_u$  into consideration during the packing process. Therefore, it leads to more substantial reduction of the number of  $K$ -LUTs than the predecessor-packing operation defined in [9]. Our experimental results show that on average the postprocessing phase reduced the number of  $K$ -LUTs in the mapping solution by 13.0%, and the Flow-Pack operation alone reduced the number of  $K$ -LUTs by 11.6%.

## 5. Experimental Results

We have implemented the Flow-Map algorithm and its preprocessing and postprocessing steps and tested them on a set of MCNC benchmark examples. We chose  $K = 5$  and compared our results with those produced by previous algorithms including Chortle-d [10], DAG-Map [9], and MIS-pga delay optimization algorithm [11].

Table 1 compares the performance of Flow-Map with Chortle-d and DAG-Map, using the input networks that were used by Chortle-d [10]. Overall, the solutions of Chortle-d used 50.4% more 5-LUTs and had 4.8% larger network depth; the solutions of DAG-Map used 8.6% more 5-LUTs and had 2.4% larger network depth. Flow-Map always results in the mapping solution of the smallest depth, and in most cases uses less number of 5-LUTs.

We also compared Flow-Map with MIS-pga(delay) in Table 2. The results of MIS-pga(delay) are cited from [11] (since we are unable to run their program directly). We obtain the results of Flow-Map by first synthesizing the original benchmarks using a standard MIS optimization script (used by Chortle-crf [3] and DAG-Map [9]) for technology-independent optimization, then applying the Flow-Map algorithm for technology mapping. Since MIS-pga(delay) combines logic synthesis and technology mapping, in several cases it produced mapping

Technology Mapping for 5-LUT FPGAs (I)						
	Chortle-d		DAG-Map		Flow-Map	
	LUTs	depth	LUTs	depth	LUTs	depth
<i>5xp1</i>	26	3	24	3	25	3
<i>9sym</i>	63	5	61	5	61	5
<i>9symml</i>	59	5	58	5	58	5
<i>C499</i>	382	6	207	5	154	5
<i>C880</i>	329	8	243	8	232	8
<i>alu2</i>	227	9	169	8	162	8
<i>alu4</i>	500	10	305	10	268	10
<i>apex6</i>	308	4	266	4	257	4
<i>apex7</i>	108	4	91	4	89	4
<i>count</i>	91	4	81	4	76	3
<i>des</i>	2086	6	1433	6	1308	5
<i>duke2</i>	241	4	192	4	187	4
<i>misex1</i>	19	2	15	2	15	2
<i>rd84</i>	61	4	43	4	43	4
<i>rot</i>	326	6	292	6	268	6
<i>vg2</i>	55	4	46	4	45	4
<i>z4ml</i>	25	3	17	3	13	3
<b>total</b>	4906	87	3543	85	3261	83
<b>cmprsn</b>	+50.4%	+4.8%	+8.6%	+2.4%	1	1

Table 1. Comparison with Chortle-d and DAG-Map.

solutions of smaller depth than those of Flow-Map. However, on average MIS-pga(delay) still used 9.8% more 5-LUTs and had 7.1% larger depth.

Our experiments were carried out on a Sun SPARC IPC workstation (14.8 MIPS). For each benchmark example, the Flow-Map package took only less than a minute of CPU time (a few seconds in most cases) to generate a mapping solution. Therefore, it is much faster than Boolean optimization based algorithms in general.

## 6. Future Work

Currently, we are extending the Flow-Map algorithm to handle more complex delay models (such as the *nominal delay model* [22], which considers both the level and the number of fanouts of each node in delay computation). We are also studying the trade-off between delay and area in technology mapping for FPGA designs.

## Acknowledgments

We thank J. Rose, R. Francis and R. Murgai for their assistance in our comparative study. This research is partially supported by the NSF grant MIP-9110511,

Technology Mapping for 5LUT FPGAs (II)				
	MIS-pga (delay)		Flow-Map	
	LUTs	depth	LUTs	depth
<i>5xp1</i>	21	2	22	3
<i>9sym</i>	7	3	60	5
<i>9symml</i>	7	3	55	5
<i>C499</i>	199	8	68	4
<i>C880</i>	259	9	124	8
<i>alu2</i>	122	6	155	9
<i>alu4</i>	259	11	253	9
<i>apex6</i>	274	5	238	5
<i>apex7</i>	95	4	79	4
<i>count</i>	81	4	31	5
<i>des</i>	1397	11	1310	5
<i>duke2</i>	164	6	174	4
<i>misex1</i>	17	2	16	2
<i>rd84</i>	13	3	46	4
<i>rot</i>	322	7	234	7
<i>vg2</i>	39	4	29	3
<i>z4ml</i>	10	2	5	2
<b>total</b>	3182	90	2899	84
<b>comparison</b>	+9.8%	+7.1%	1	1

Table 2. Comparison with MIS-pga (delay optimization) algorithm.

the State of California MICRO program NO. 92-030, and a grant from Xilinx Inc..

## References

- [1] Hill, D., "A CAD System for the Design of Field Programmable Gate Arrays," Proc. ACM/IEEE Design Automation Conference, pp. 187192, 1991.
- [2] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose (1989).
- [3] Francis, R. J., J. Rose, and Z. Vranesic, "Chortlecrf: Fast Technology Mapping for Lookup TableBased FPGAs," Proceedings 28th ACM/IEEE Design Automation Conference, pp. 613619, 1991.
- [4] Karplus, K., "Xmap: A Technology Mapper for Tablelookup FieldProgrammable Gate Arrays," Proc. 28th ACM/IEEE Design Automation Conference, pp. 240243, 1991.
- [5] Murgai, R., et al, "Logic Synthesis Algorithms for Programmable Gate Arrays," Proc. 27th ACM/IEEE Design Automation Conf., pp. 620625, 1990.
- [6] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures , " Proc. Int'l Conf. ComputerAided Design, pp. 564567, Nov., 1991.
- [7] Sawkar, P. and D. Thomas, "Technology Mapping for TableLookUp Based Field Programmable Gate Arrays," ACM/SIGDA Workshop on Field Programmable Gate Arrays, pp. 8388, Feb. 1992.

- [8] Woo, N.S., "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," Proc. 28th ACM/IEEE Design Automation Conference, pp. 248251, 1991.
- [9] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAGMap: Graphbased FPGA Technology Mapping for Delay Optimization," IEEE Design and Test of Computers, Sep. 1992.
- [10] Francis, R. J., J. Rose, and Z. Vranesic, "Technology Mapping of Lookup TableBased FPGAs for Performance," Proc. Int'l Conf. ComputerAided Design, pp. 568571, Nov., 1991.
- [11] Murgai, R., N. Shenoy, R. K. Brayton, and A. SangiovanniVincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," Proc. Int'l Conf. ComputerAided Design, pp. 572575, Nov., 1991.
- [12] Bhat, N. and D. Hill, "Routable Technology Mapping for FPGAs," First Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays, pp. 143148, Feb. 1992.
- [13] Schlag, M., J. Kong, and P. K. Chan, "RoutabilityDriven Technology Mapping for Lookup TableBased FPGAs," Proc. 1992 IEEE International Conference on Computer Design, Oct. 1992.
- [14] Detjens, E., G. Gannot, R. Rudell, A. SangiovanniVincentelli, and A. Wang, "Technology Mapping in MIS," Proc. IEEE Int'l Conf. on ComputerAided Design, pp. 116119, 1987.
- [15] Keutzer, K., "DAGON: Technology Binding and Local Optimization by DAG Matching," Proc. 24th ACM/IEEE Design Automation Conference, pp. 341347, 1987.
- [16] Francis, R. J., J. Rose, and Z. Vranesic, "Technology Mapping for Delay Optimization of Lookup TableBased FPGAs," MCNC Logic Synthesis Workshop, 1991.
- [17] Roth, J. P. and R. M. Karp, "Minimization Over Boolean Graphs," IBM Journal of Research and Development, pp. 227238, April 1962.
- [18] Hoover, H. J., M. M. Klawe, and N. J. Pippenger, "Bounding Fanout in Logic Networks," Journal of Association for Computing Machinery, Vol. 31, pp. 1318, Jan. 1984.
- [19] Wang, A., "Algorithms for Multilevel Logic Optimization," U.C.Berkeley Memorandum No. UCB/ERL M89/50, April 1989.
- [20] Cong, J. and Y. Ding, "An Optimal Technology Mapping Algorithm fo Delay Optimization in LookupTable Based FPGA Designs," in UCLA Computer Science Department Technical Report CSD920022, (May 1992).
- [21] Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, N.J. (1962).
- [22] Schlag, M., P. Chan, and J. Kong, "Empirical Evaluation of Multilevel Logic Minimization Tools for a Field Programmable Gate Array Technology," Proc. 1st Int'l Workshop on Field Programmable Logic and Applications, Sept. 1991.

# LOGIC DECOMPOSITION DURING TECHNOLOGY MAPPING

Eric Lehman, Yosinori Watanabe, Joel Grodstein, Heather Harkness

*Digital Equipment Corporation*

*77 Reed Road, Hudson, MA 01749, USA*

## Abstract

A problem in technology mapping is that quality of the final implementation depends significantly on the initially provided circuit structure. To resolve this problem, conventional techniques iteratively but separately apply technology independent transformations and technology mapping.

In this paper, we propose a procedure which performs logic decomposition and technology mapping simultaneously. We show that the procedure effectively explores all possible algebraic decompositions. It finds an optimal tree implementation over all the circuit structures examined, while the run time is typically logarithmic in the number of decompositions.

## 1. Introduction

Technology mapping is usually applied on a particular structure (decomposition) of logic expressions. Therefore, quality of the final implementation could severely depend upon the structure initially provided to the technology mapper.

The most common approach for generating the initial structure of logic expressions for technology mapping consists of two phases [2]. In phase (1), logic expressions are optimized in a technology independent manner. The result is represented by a graph, which is sometimes called a *boolean network* [2]. Then in phase (2), the resulting expressions are converted into a special type of boolean network, where nodes represent particular functions. (In this paper, we assume that the resulting network is represented by two-input ANDs (AND2s) and inverters. We call such a network AND2/INV network.) Technology mapping is the third and last phase of synthesis, where the AND2/INV network is *mapped* onto a set of library gates.

Synthesis techniques based on these three phases have an obvious drawback: optimizations applied in each of the phases are disconnected. Although decisions made in the first and the second phases critically affect the final results, it is not clear in these phases how the resulting logic expressions are actually implemented. On the other hand, in technology mapping, real data on constraints and library gates are available, but the degree of freedom for the final implementation is limited by the decisions made in the preceding phases. This drawback is fatal when one needs to handle tight and complicated constraints.

A variety of techniques have been proposed in the literature to resolve this problem [1, 6, 15, 5, 11, 16, 18, 13], most of which iteratively apply the three

phases above. Even though these techniques yield some improvements, the effect is limited because the disconnection between phases (1) (2) and technology mapping is not fully resolved. Data from the initial mapping are quickly invalidated by operations of phases (1) and (2), and cease to be an effective guide. Moreover, without information about library gates, one can foresee little about the final implementation, and thus transformations are made somewhat blindly.

The central problem is that technology independent transformations and technology mapping do not cooperate. This is the problem addressed in this paper. Among technology independent transformations, our focus is on algebraic logic decomposition [3], a key operation for changing circuit structures. We asked ourselves if it is possible to apply algebraic logic decomposition and technology mapping simultaneously. The answer is yes, and we present a procedure which is as effective as applying technology mapping over all possible algebraic decompositions exhaustively. The proposed procedure compactly encodes a *set of decompositions (structures)* into a single graph, and applies a technology mapping algorithm on it. In the meanwhile, it dynamically modifies the set of decompositions encoded in the graph by introducing new decompositions while deleting others based on the actual cost function used in technology mapping. It is guaranteed that the procedure finds an optimal solution among all the decompositions examined.

The graph for representing a set of decompositions is called a *mapping graph*. It is obtained as the union of AND2/INV networks, where nodes of the networks are merged as much as possible. Each AND2/INV network corresponds to a single decomposition, which we call an AND2/INV decomposition. State-of-the-art technology mapping techniques can be naturally extended for mapping graphs, retaining linear run time and optimal results<sup>1</sup>.

A set of AND2/INV decompositions are generated through iterative local transformations defined on a mapping graph. We define three such transformations: associative transformation, distributive transformation, and inverter transformation. The associative transformation is based on the associative law, i.e. two AND2s representing  $a(bc)$  are transformed into two AND2s for  $(ab)c$  and vice versa. Similarly, the distributive transformation is based on the distributed law, and transforms  $ab + ac$  to  $a(b + c)$ . The inverter transformation adds or removes two consecutive inverters between two consecutive nodes in a mapping graph. These transformations are the vehicle to introduce new decompositions.

In this paper, we examine two sets of AND2/INV decompositions. The first is the closure of the set of AND2/INV decompositions of a given boolean network  $\eta$  under the associative and inverter transformations, which we denote by  $\Lambda_\eta$ . We show that phase (2) of the conventional logic synthesis approach, where a boolean network is converted to an AND2/INV decomposition, is completely subsumed by  $\Lambda_\eta$ , i.e. every AND2/INV decomposition of  $\eta$  is contained in  $\Lambda_\eta$ . Therefore if  $\Lambda_\eta$  is encoded in a mapping graph and technology mapping is

applied, an optimal result over all possible AND2/INV decompositions of  $\eta$  is obtained. This technology mapping procedure is referred to as the  $\Lambda$ -mapping procedure.

The second set of AND2/INV decompositions we examine is even larger. It is the closure of  $\Delta_\eta$  under the distributive transformation, which we denote by  $\Delta_\eta$ . We show that  $\Delta_\eta$  subsumes phase (2) as well as a key part of phase (1), i.e. algebraic logic decomposition. In other words, no matter how algebraic logic decomposition is applied on a given boolean network  $\eta$  in phase (1), and no matter how the result is converted to an AND2/INV decomposition in phase (2), the resulting decomposition is contained in  $\Delta_\eta$ . Therefore, the technology mapper finds an optimal result over all possible AND2/INV decompositions given through all possible algebraic decompositions. We call this procedure  $\Delta$ -mapping.  $\Delta$ -mapping dynamically applies the distributive transformation during technology mapping, so that new decompositions are introduced on the fly. It also deletes some decompositions, but does so only when it is known that at least equally good implementations can be obtained without them, and thus quality of the final result is not affected. Therefore, it controls the size of the set of encoded decompositions, and effectively explores the entire  $\Delta_\eta$ .

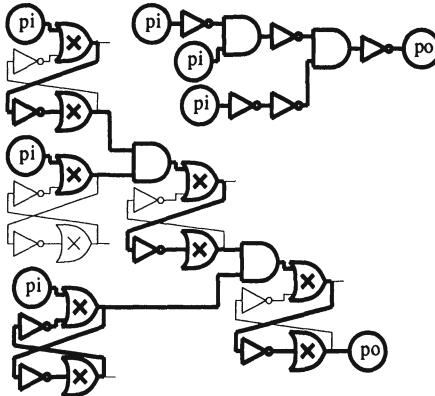
Both the  $\Lambda$  and  $\Delta$  mapping procedures have been implemented. Extensions were made to handle sequential circuits, so that logic decompositions across latches and retiming possibilities are also taken into account. They are being used for commercial design projects, where  $\Lambda$ -mapping is first used, and then  $\Delta$ -mapping is applied on timing-critical regions to further speed up the final implementations. We show that both procedures are feasible in terms of complexity for most practical examples, and demonstrate their effectiveness on benchmark examples.

## 2. Preliminaries

We adopt the notion of *boolean network* and associated terminology from [2]. A variable  $y_n$  is associated with each primary input or internal node  $n$ . A sum-of-products expression  $f_n$  is associated with each internal node  $n$ .

An internal node  $n$  in a boolean network can be replaced by a logically equivalent collection of two-input AND nodes (AND2s) and inverters. Working from the sum-of-products  $f_n$ , each product term is represented by a tree of AND2s. The sum is represented by a tree of AND2s with an inverter at each leaf and at the root. The result is called an *AND2/INV decomposition* of  $n$ . An AND2/INV decomposition of a boolean network is formed by replacing each internal node by an AND2/INV decomposition of that node.

A boolean network in which every internal node is either an AND2 or inverter is called an *AND2/INV network*. An AND2/INV decomposition of a boolean network is always an AND2/INV network. Since our technology mapping pro-



*Figure 1.* The lower left diagram is a mapping graph and the upper right diagram is an AND2/INV network encoded within. The path of the depth-first search which generated the AND/INV network is highlighted on the left.

cedures are tree-based, whether internal nodes in an AND2/INV network are shared or duplicated is irrelevant in this paper. Therefore, it is assumed that all AND2/INV networks are converted by duplication to a canonical form in which every internal node has exactly one fanout.

A *library* is a set of gates, each accompanied by area, delay, and loading characteristics and by a single-output boolean network representing its logic function. A boolean network in which every internal node is associated with a logically equivalent library gate is called a *mapped network*.

### 3. Mapping on a Set of Decompositions

In this section we introduce two basic tools and use them to build a mapping procedure. The first subsection defines the *mapping graph* data structure, which encodes a set of AND2/INV networks. Next, we present the *graph-mapping* algorithm which effectively applies tree-mapping [9] to every AND2/INV network encoded in a mapping graph. Graph-mapping can be applied to any mapping graph. In the third subsection, we use these two tools to build  $\Lambda$ -*mapping*, a two-step procedure consisting of one particular way of generating a mapping graph from a boolean network followed by a call to graph-mapping.

#### 3.1 Mapping Graph Data Structure

Conceptually, a *mapping graph* is an AND2/INV network with four modifications. An example is given in the lower-left portion of Figure 1.

First, a new type of node is introduced. A *choice* node  $b$  is drawn as an OR gate marked with an X. A choice node has an associated variable  $y_b$ , but no associated sum-of-products. All fanins of a choice node must be logically

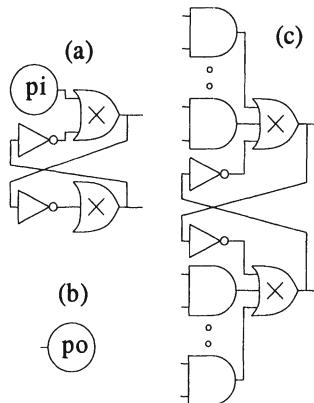


Figure 2. A ugate is a subgraph of a mapping graph. There are three types: (a) primary input (b) primary output (c) AND. For primary input and AND ugates, the top half of each is the *positive half* and the bottom half is the *negative half*. Likewise, the top output is the *positive output* and the bottom is the *negative output*. The set of AND2s in the positive half of ugate  $u$  is denoted  $H_u^1$  and the set of AND2s in the negative half is denoted  $H_u^0$ . Every AND ugate contains at least one AND2, but otherwise there may be any number in each half.

equivalent. Intuitively, the technology mapper will first view the choice node as a wire connecting the first fanin of the choice node to its output, then as a wire connecting the second fanin to the output, etc. Therefore, several AND2/INV representations of the same boolean function can be supplied to the mapper by making the outputs of all the representations into fanins of a common choice node.

Second, a mapping graph may contain directed cycles. A directed cycle specifies that a function  $f$  may be expressed in terms of  $g$  (creating a directed path from a node computing  $g$  to a node computing  $f$ ) and also vice versa (completing the directed cycle). The technology mapper will consider both implementations. A typical example of a directed cycle, consisting of two inverters and two choice nodes, appears several places in Figure 1. This cycle structure encodes the notion of an arbitrary length inverter chain. Cycles are permitted for theoretical and implementation convenience.

Third, a mapping graph can be partitioned into disjoint subgraphs, such that each subgraph is a *ugate*. This term is defined in Figure 2. This restriction imposes a degree of regularity on the structure of a mapping graph, without sacrificing any expressive power. This regularity greatly simplifies implementation. A good mental picture of a mapping graph is a directed graph containing interconnected, but isolated clumps of nodes, where each clump is a ugate.

Fourth, a mapping graph must be *reduced*: there can not exist two choice nodes with logically equivalent outputs. This restriction can be met by merging ugates. If ugates  $u$  and  $v$  have logically equivalent choice nodes, then copies of

the positive and negative AND2s of  $u$  are added to the equivalent halves of  $v$ . If two AND2s of  $v$  now have identical inputs, one is discarded. All fanouts of the positive and negative outputs of  $u$  are connected to the equivalent outputs of  $v$ . The nodes of  $u$  are deleted. This restriction not only shrinks the graph, but also provides greater freedom to the mapper by collecting together all known ways of expressing a function with AND2s and inverters. Intuitively, wherever one such expression was required, all are now available.

An AND2/INV network encoded by a mapping graph is defined as follows. Consider a depth-first search rooted at a primary output of the mapping graph with primary inputs at the leaves. At an AND2 node, the search proceeds to both fanins. At a choice node, the search proceeds to only one fanin. The nodes and arcs traversed in the search form a tree, assuming nodes and arcs visited more than once are duplicated. Replace each choice node in the tree by a wire connecting the output of the choice node to the single fanin explored by the search. See Figure 1. Each complete AND2/INV network encoded in a mapping graph consists of one such tree for each primary output. The set of all complete AND2/INV networks obtainable in this way is the set of AND2/INV networks encoded by a mapping graph.

## 3.2 Graph-Mapping

This section describes the graph-mapping procedure, an extension of tree-mapping [9]. The algorithm transforms a mapping graph  $\mu$  into a mapped network in which primary outputs are driven by disjoint trees of gates with primary inputs at the leaves. In practice heuristic methods for sharing logic between trees are essential, but in the present theoretical discussion we assume no sharing is used.

**3.2.1 Preliminaries.** This section reviews three notions used in the graph-mapping algorithm: matching, mapping, and cost.

For a given choice node and library gate, *matching* identifies all single-output subgraphs of  $\mu$  rooted at the choice node which are logically equivalent to the library gate. Since numerous methods for matching are available [7], we only formalize the problem.

An AND2/INV decomposition of the boolean network for a library gate is called a *pattern*. A *match*  $\phi$  of a pattern at an AND2 or inverter node  $a$  in mapping graph  $\mu$  is a bijective function such that: (1)  $\phi$  projects each internal node in the pattern to an internal node in  $\mu$ . (2)  $\phi$  projects the fanin of the pattern's primary output to  $a$ . (3) Nodes  $b$  and  $\phi(b)$  are either both AND2s or else both inverters. (4) For nodes  $b, c$  in the pattern there is a directed arc from  $b$  to  $c$  if and only if there is a directed path from  $\phi(b)$  to  $\phi(c)$  crossing only choice nodes. (Intuitively, an arc in the pattern may reach across a choice node in the mapping graph.) Every match at internal node  $a$  is also considered a match at

the choice node which is the fanout of  $a$ . Let  $C$  be the image of  $\phi$ . The nodes in  $C$  and choice nodes between nodes in  $C$  are said to be *covered* by the match  $\phi$ . A fanin of a node in  $C$  which is not covered by the match is a *fanin* of the match. From this definition, every fanin of a match is a choice node.

A *mapping* at a choice node  $c$  in  $\mu$  is a recursively-defined set of matches. Each match corresponds to a library gate and so a mapping uniquely defines a tree of library gates with primary input nodes at the leaves. Generally, a mapping consists of a match  $\phi$  at  $c$  and the union of a mapping at each fanin of  $\phi$ . In the special case when  $c$  is the fanout a primary input node, either the mapping is the empty set or else the recursion continues as before. The latter possibility addresses the case of a primary input driving a chain of inverters and buffers.

Each such mapping has a *cost*. This cost may measure the total area of the tree of gates, the arrival time at the root of the tree, etc. Generally, any notion of cost used in tree-mapping [4, 17] is acceptable<sup>2</sup>. However, to guarantee that graph-mapping terminates, the library and cost notion must be designed so that an optimal implementation exists; more precisely, there can exist no infinite sequence of logically equivalent trees of gates with increasing size and non-increasing cost. At each choice node  $c$  in  $\mu$ , the graph-mapping procedure maintains a value  $cost(c)$ . This is the cost of the least-cost mapping at  $c$  yet found.

**3.2.2 The Procedure.** This section describes graph-mapping as a four-step procedure. The first two steps are initializations. The third and fourth steps are analogous to the forward and backward passes of tree-mapping, but with a generalized definition of “match”.

First, for each choice node  $c$  with a fanin which is a primary input,  $cost(c)$  is assigned an externally-supplied initial value. For example, if cost measures the arrival time at the root of a tree of gates, then this initial value is the input arrival time. For every other choice node  $c$  in  $\mu$ ,  $cost(c)$  is set to infinite.

Second, each ugate  $u$  is assigned a number,  $label(u)$ , such that the following three conditions hold. If there is only a directed path from a node in  $u$  to a node in  $v$ , then  $label(u) < label(v)$ . If there is also a directed path from a node in  $v$  to a node in  $u$ , then  $label(u) = label(v)$ . If there is no directed path between a node in  $u$  and a node in  $v$ , then  $label(u) \neq label(v)$ . This requires a straightforward graph algorithm which assigns increasing labels from inputs to outputs and the same label to all ugates in a cycle.

Third, a forward pass is made through  $\mu$ . Ugates are visited in order of increasing label and a function **Match** is called on each visited ugates. This function considers mappings at each choice node  $c$  in the ugates and updates  $cost(c)$  whenever a new, lower-cost mapping is found. **CostFunc** is analogous to the cost function used in tree-mapping. If there is set of ugates all with the same label, then **Match** is iteratively called on every ugates in the set until there is a

```

procedure Match( $u$ )
do
  for each choice node  $c$  in  $u$ 
    for each match  $\phi$  of a library gate  $g$  at  $c$ 
      let  $f_1 \dots f_k$  be the fanins of match  $\phi$ 
      let  $cost(c) = \text{MIN}(cost(f_1), \dots, cost(f_k))$ 
      CostFunc( $g, cost(f_1), \dots, cost(f_k)$ )
  while the cost at some node changed during this iteration

```

*Figure 3.* Match function

complete iteration during which no *cost* value changes at any choice node in any ugates in the cycle.

Fourth, a second pass is made through  $\mu$  by working backward recursively from each primary output. The purpose is to construct a mapped network reflecting the *cost* values computed in step three. The method is analogous to that used in tree-mapping.

**3.2.3 Effectiveness.** The graph-mapping algorithm is as effective as applying tree-mapping to every encoded AND2/INV network, but generally far more efficient.

**Theorem 1** Suppose graph-mapping is applied to a mapping graph  $\mu$ , and tree-mapping is applied to an AND2/INV network  $\theta$  encoded in  $\mu$ . For each primary output, the implementation produced by graph-mapping has cost less than or equal to the implementation produced by tree-mapping with the same CostFunc.

The algorithm usually runs in linear time in the number of nodes in the mapping graph (steps one, two, and three) plus the size of the mapped network (step four). The **while** loop in Match typically runs a small number of times. Loops in step three are uncommon in practice.

The efficiency of graph-mapping compared to exhaustive application of tree-mapping is therefore closely related to the compactness with which a mapping graph encodes a set of AND2/INV networks. In fact, the mapping graph representation is often extremely effective. For example, there are  $2.22 \times 10^{93}$  distinct AND2/INV decompositions of the MCNC91 cht benchmark. However, this entire set can be encoded in a mapping graph with 400 ugates containing a total of 599 AND2 nodes using the procedure described in the next section.

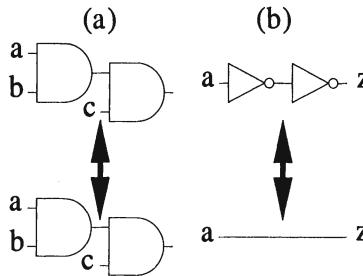


Figure 4. Two transformations of an AND2/INV network: (a) associative transformation (b) inverter transformation.

### 3.3 $\Lambda$ -Mapping

For a boolean network  $\eta$ , the set  $\Lambda_\eta$  is defined as the closure of the set of AND2/INV decompositions of  $\eta$  under the *associative* and *inverter* transformations. These transformations are depicted in Figure 4. This section defines  $\Lambda$ -mapping, a mapping procedure which explores the entire space  $\Lambda_\eta$ . The following three-part procedure constructs a mapping graph  $\mu$  encoding  $\Lambda_\eta$ .  $\Lambda$ -mapping is completed by applying graph-mapping to  $\mu$ .

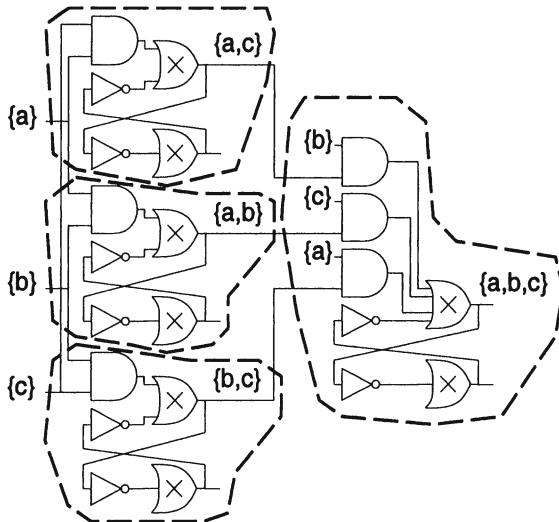
First,  $\eta$  is used to produce a new boolean network  $\tilde{\eta}$ . Each node  $n$  of  $\eta$  generates a set of multi-input AND's and inverters in  $\tilde{\eta}$  representing the sum-of-products  $f_n$ . Each AND and inverter is a single node in  $\tilde{\eta}$ . All pairs of inverters in series are deleted from  $\tilde{\eta}$ . If one AND drives another in  $\tilde{\eta}$ , the two are combined to form a single AND with more inputs.

Second, each node of  $\tilde{\eta}$  is translated to a set of nodes in  $\mu$ . The translation works from inputs to outputs. A particular ugate output, denoted  $root(n)$ , is associated with each node  $n$  in the network  $\tilde{\eta}$  after it is processed. Each primary input  $n$  in  $\tilde{\eta}$  generates a primary input ugate  $u$  in  $\mu$  where  $root(n)$  is the positive output of  $u$ . An inverter  $n$  with fanin  $m$  does not generate a ugate. However, if  $root(m)$  is an output of the ugate  $u$ , then  $root(n)$  is the other output of ugate  $u$ . A primary output  $n$  with fanin  $m$  generates a primary output ugate driven by  $root(m)$ . The most complex case is a multi-input AND node  $n$ . For each subset of two or more inputs, a ugate is created in the positive half which contains an AND2 for each partition of the subset into two non-empty, disjoint parts.  $root(n)$  is the positive output of the ugate associated with the full set of inputs. See Figure 5 for a three-input example.

Finally,  $\mu$  is reduced as described in Section 3.1.

**Theorem 2** *For a general boolean network  $\eta$ , every element of  $\Lambda_\eta$  is encoded in the mapping graph  $\mu$  generated by the above construction.*

As a corollary, the result of  $\Lambda$ -mapping is as good or better than obtained by applying tree-mapping to every possible AND2/INV decomposition of the



*Figure 5.* A portion of a mapping-graph which encodes all AND2/INV decompositions of  $y = a \oplus b \oplus c$  is shown. Ugates are circled and the associated subset of inputs is indicated. The ugates producing signals  $a$ ,  $b$ , and  $c$  are not shown.

original network. All decisions taken blindly in phase (2) of the conventional synthesis approach, AND2/INV decomposition, are now taken optimally during technology mapping.

$\Delta$ -mapping often explores a space larger than  $\Lambda_\eta$ . This is because extra AND2/INV decompositions may be generated during the reduction operation in step three of the mapping graph construction.

In practice, it is usually feasible to encode  $\Lambda_\eta$  in a mapping graph provided no AND in the intermediate network  $\tilde{\eta}$  has more than about 10 inputs. This means that  $\Delta$ -mapping can be applied to most practical examples. Of course, if there is a larger AND in  $\tilde{\eta}$ , it is still possible to encode numerous decompositions of the node.

## 4. Dynamic Logic Decomposition

In this section, we extend  $\Delta$ -mapping, so that the set of AND2/INV networks is dynamically modified during technology mapping. This process is done by dynamically performing logic decomposition on a mapping graph.

### 4.1 $\Delta$ -Mapping

We first define a new transformation on an AND2/INV network, the vehicle for dynamic logic decomposition. The transformation is based on the *distributive law*, and is referred to as the *distributive transformation*. The distributive transformation replaces  $xy + xz$  by  $x(y + z)$ , which is illustrated in Fig-

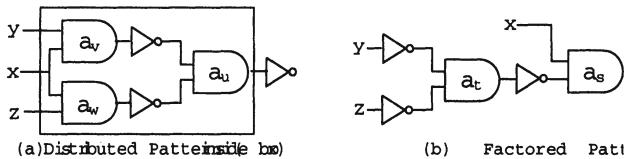


Figure 6. Distributive Transformation

ure 6, where Figure 6-(a) corresponds to  $xy + xz$  and Figure 6-(b) corresponds to  $x(y + z)$ . We call the structure shown in the rectangle of Figure 6-(a) the *distributed pattern*, or *D-pattern* for short. Similarly, the structure shown in Figure 6-(b) is referred to as the *factored pattern*, or *F-pattern* for short. Note that the functionality of a D-pattern is the complement of that of the corresponding F-pattern.

The idea of dynamic decomposition is simple: while traversing a mapping graph during technology mapping, we identify D-patterns and add the corresponding F-patterns to the mapping graph. We call this technology mapping procedure  $\Delta$ -mapping. We illustrate the basis of the procedure in this section, and refine it in Sections 4.3 and 4.4.

The  $\Delta$ -mapping procedure is an iteration of the graph-mapping procedure. Once a mapping graph is constructed for a given boolean network, we apply graph-mapping while finding D-patterns and adding F-patterns. Once graph-mapping is done, we check if there is a solution which meets user-specified constraints. If so, we terminate  $\Delta$ -mapping by generating the result. Otherwise, we apply the graph-mapping procedure again, while identifying more D-patterns, since new D-patterns might have been introduced when F-patterns were added in the previous iteration. This process is iterated until either a desired solution is found or no new D-pattern is found.

The core procedure is `Decomp_and_match`, illustrated in Figure 7. This procedure is called in graph-mapping as a substitute for `Match`, the function defined in Section 3.2 for finding matches at a ugate. `Decomp_and_match` is identical with `Match` if the ugate  $u$  being processed is not of type AND. If the type of  $u$  is AND, `Decomp_and_match` identifies D-patterns and adds F-patterns, before calling `Match`. More precisely, for each polarity  $p \in \{0, 1\}$ , we look at each AND2  $a_u$  of  $H_u^p$ , the set of AND2s of  $u$  for the polarity  $p$ . We check if a D-pattern matches at  $a_u$ . If this is the case, we construct the corresponding F-pattern shown in Figure 6-(b), and add it to the mapping graph. Since the functionality of the resulting F-pattern is the complement of that of the D-pattern we identified, the root AND2 of the F-pattern, denoted by  $a_s$  in Figure 6-(b), is added to  $H_u^{\bar{p}}$ , the set of AND2s of  $u$  whose polarity is the opposite of  $p$ . Recall the structure of an AND ugate shown in Figure 2-(c).

```

procedure Decomp_and_match( $u$ )
  if  $u$  is AND ug ate
    for each polarity  $p \in \{0, 1\}$ 
      for each  $a_u \in H_u^p$ 
        if D-pattern is found at  $a_u$ 
          add F-pattern to the mapping graph
          /*  $t$  is a ug ate for the second
          AND2  $a_t$  of F-pattern. */
          Match( $t$ )
  Match( $u$ )

```

*Figure 7.* Dynamic Logic Decomposition (basic flow)

To add the other AND2 of the F-pattern, denoted by  $a_t$  in Figure 6-(b), we first create a new AND ug ate, which consists of  $a_t$  in the positive half. The negative output of the ug ate fans out to  $a_s$ . The resulting mapping graph is then reduced. We invoke Match for the ug ate containing  $a_t$ .

Once this process has been applied for both polarities, we find matches for  $u$  and update the cost by calling Match. In the rest of the paper, we often say that dynamic decomposition is applied for a ug ate  $u$  or an AND2  $a$ , by which we mean that Decomp\_and\_match, or its subprocedure for a single AND2, is applied for  $u$  or  $a$  respectively.

## 4.2 The Space Explored by $\Delta$ -Mapping

*Algebraic decomposition* is a three-step operation defined on a node  $n$  of a boolean network: (1) find an algebraic divisor  $g$  of  $f_n$ , (2) create or find a node  $m$  such that  $f_m = g$ , and (3) replace  $f_n$  by  $r + \sum_{q \in h} qy_m$ , where  $h$  is a quotient obtained by algebraically dividing  $f_n$  by  $g$ ,  $q$  is a product term of  $h$ , and  $r$  is the corresponding remainder. A definition of algebraic division is found in [3].

For a given boolean network  $\eta$ , let  $\Delta_\eta$  denote the closure of  $\Lambda_\eta$  under the distributive transformation. Then the following claim holds [10].

**Theorem 3** *Given a boolean network  $\eta$ , let  $\tilde{\eta}$  be a boolean network obtained from  $\eta$  by successively applying algebraic decomposition. Then every AND2/INV decomposition of  $\tilde{\eta}$  is contained in  $\Delta_\eta$ .*

This theorem claims that no matter how algebraic decomposition is applied on  $\eta$ , every AND2/INV decomposition of the resulting boolean network is contained in  $\Delta_\eta$ . Now, let  $D_\eta$  be the set of AND2/INV networks captured by applying the  $\Delta$ -mapping procedure on  $\eta$ . Specifically, suppose we apply dynamic decomposition maximally on ugates of a mapping graph  $\mu$  constructed for  $\eta$  as

described in Section 3.3, i.e. whenever a D-pattern is found, the corresponding F-pattern is added to  $\mu$ , and this process is repeated until nothing new is found.  $D_\eta$  is given by the set of AND2/INV networks encoded in the resulting mapping graph. Since  $\Delta$ -mapping subsumes  $\Lambda_\eta$ ,  $D_\eta$  subsumes  $\Delta_\eta$ . Therefore, Theorem 3 implies that by applying  $\Delta$ -mapping, one can effectively explore the entire set of AND2/INV networks obtained from  $\eta$  through algebraic decomposition and AND2/INV decomposition. By Theorem 1, an optimal result over all such decompositions is obtained. We emphasize the point that  $\Delta$ -mapping explores the space directly on a mapping graph during technology mapping, and there is no need to go back and forth between the technology independent phase and the technology dependent phase.

One may generalize the algebraic decomposition so that after the third step, the expression of the node  $n$  is further divided by the complement of the divisor  $g$ . Such a generalized algebraic decomposition is also subsumed by  $\Delta$ -mapping. Let  $\tilde{\eta}$  denote the boolean network obtained by the generalized algebraic decomposition with a divisor  $g$ . In an AND2/INV decomposition of  $\tilde{\eta}$ , a complemented expression  $\bar{g}$  is represented as an inverter driven by an AND2/INV decomposition for  $g$ . Since a mapping graph is reduced, i.e. there are no two distinct choice nodes that are logically equivalent, an AND2/INV network for  $\bar{g}$  is also encoded in the mapping graph as an inverter driven by the choice node whose transitive fanin represents AND2/INV networks for  $g$ . Hence, the set of AND2/INV decompositions of  $\tilde{\eta}$  is contained in  $D_\eta$ . However, it is known that the set is not necessarily contained in  $\Delta_\eta$  in general, and thus  $D_\eta$  can be strictly larger than  $\Delta_\eta$ .

### 4.3 Factor-Free Ugates

In practice, the  $\Delta$ -mapping procedure is used to speed up a timing critical region of a mapped network. Since the region is usually small and to maximize the effectiveness of dynamic decomposition, we initially collapse the region to a sum-of-products expression. Thus, the procedure is typically applied on a boolean network with a single node. In this and next sections, we modify  $\Delta$ -mapping in order to improve efficiency of the procedure without sacrificing the quality of results for this typical case.

$\Delta$ -mapping iteratively applies the graph-mapping procedure with dynamic decomposition. It is inefficient if the distributive transformation is applied many times on a ugate at which no new D-pattern can be found. In this section, we discuss a condition under which such an inefficient computation can be avoided.

Intuitively, dynamic decomposition can be skipped for an AND2  $a$  if all D-patterns have been identified in the transitive fanin of  $a$ . An AND2 of a ugate is said to be *factor-free* if there is no *untransformed* D-pattern in its transitive fanin, where a D-pattern is said to be *transformed* if the corresponding F-pattern

has been added to the mapping graph, and *untransformed* otherwise. A ugate is said to be *factor-free* if all of its AND2s are factor-free.

The  $\Delta$ -mapping procedure identifies this property locally. It marks an AND2  $a$  as FF (factor-free) if dynamic decomposition has been applied for  $a$  and both of its fanin ugates are marked as FF. A ugate is marked as FF if all of its AND2s are marked as FF. Initially, we mark ugates for primary inputs as FF. When `Dynamic_decomp` is invoked, we skip the procedure for an AND2 if it is marked as FF.

#### 4.4 Dynamic Decomposition with Deletion

So far,  $\Delta$ -mapping only introduces new AND2/INV networks, and never deletes existing ones. However, some of them might not lead to good results, and thus it is not worth having them around. The  $\Delta$ -mapping procedure can be modified to detect such a situation, to some extent, and delete those AND2/INV networks. This section illustrates a basic idea, while details are found in [10].

The basic step of the deletion process is to delete an AND2 from the ugate it belongs to. Intuitively, it is worth having an AND2  $a$  if there is a possibility that  $a$  is included in a mapping identified at some node of the mapping graph at which no mapping with at least equally good cost can be found without  $a$ . To identify this condition, we modify  $\Delta$ -mapping as follows.

First, when `Match` is applied on a ugate  $u$ , we compute a subset  $A'$  of the set  $A$  of AND2s of  $u$  such that for each choice node  $c$  in the transitive fanout of  $u$ ,  $cost(c)$  remains same even if only  $A'$  is used for finding matches at  $u$ . This is done by using a notion called *partial matches* [7]. AND2s which are not in  $A'$  are candidates for deletion. The problem of finding  $A'$  with the minimum cardinality is known to be NP-hard [8], and thus we find one using a heuristic.

Consider an AND2  $a$  of  $u$  which is not in the set  $A'$ . We cannot immediately conclude that  $a$  can be deleted. In fact, there are two cases where  $a$  should be retained. One case is that a new match could be found at  $a$  in the future. The other case is that a untransformed D-pattern containing  $a$  could be found in the future. If neither happens,  $a$  can be deleted without affecting the quality of the result. We detect the first case above by using the flag FF (factor-free) defined in the previous section. Since the first case does not arise as long as  $a$  is factor-free, we check if  $a$  is marked as FF. The second case above is detected by using another flag called DF (distribution-free). An AND2  $a$  of a ugate  $u$  is said to be *distribution-free* if for each AND2  $\hat{a}$  which references  $u$  as a fanin,  $\hat{a}$  is distribution-free and no untransformed D-pattern matches at  $\hat{a}$ . If  $a$  is distribution-free,  $a$  will not be contained in a D-pattern when dynamic decomposition is applied the next time. Since this is true for all the fanouts of  $a$ , no new D-pattern will be introduced in its transitive fanout in the rest of the procedure. In the modified  $\Delta$ -mapping procedure, we traverse a mapping graph

between consecutive iterations of the graph-mapping procedure, and mark an AND2  $a$  of a ugate  $u$  as DF if for each fanout  $\hat{a}$  of  $u$ ,  $\hat{a}$  is marked as DF and dynamic decomposition was applied on  $\hat{a}$  in the previous iteration. We delete  $a$  from  $u$  if  $a$  is not in  $A'$  and it is marked as FF and DF. If  $a$  is deleted, the fanins of  $a$  might have no fanout. We delete the fanins as well if this is the case.

It is claimed that the modified  $\Delta$ -mapping does not affect the quality of results in practice [10].

**Theorem 4** *Suppose that a boolean network  $\eta$  consists of a single node and the associated expression is prime and irredundant. Then the cost of a mapped network for  $\eta$  obtained by the  $\Delta$ -mapping procedure with the modifications stated in Sections 4.3 and 4.4 is no worse than that given by the original  $\Delta$ -mapping.*

## 5. Experimental Results

These procedures were implemented in a synthesis system called SynFul. SynFul is developed on top of SIS [12], and inherits technology independent transformations from SIS. Operations related to technology mapping have been newly implemented. The implementation is highly optimized, with extensions for handling sequential circuits and generating non-tree implementations [10].

We conducted experiments on MCNC91 benchmark examples of combinational circuits, and compared the results to SIS-1.2. The library used is the one modeled for a commercial design project in progress at the authors' affiliation. We specified delay characteristics so that both systems computed the cost in exactly the same way to find the fastest possible implementation.

For SIS-1.2, we used a sequence of operations aimed for timing-oriented synthesis [13, 17]. A procedure for restructuring a mapped network [13] was also applied. We consulted SIS developers for detailed usage of these procedures to derive the best performance of the system [14].

In SynFul, we first applied the  $\Lambda$ -mapping procedure where, for each example, four different sets of technology independent transformations were first applied on the initial boolean network. AND2/INV decompositions for each of the resulting boolean networks were encoded into a single mapping graph  $\mu$ . The rationale is that a variety of decompositions can be encoded in a mapping graph in this way. The graph-mapping procedure was then applied on  $\mu$ . Once a mapped network was obtained, we further applied the  $\Delta$ -mapping procedure on a timing-critical region. Such a region was identified using a heuristic suggested in [13].

The results are shown in Table 1. For **SIS-1.2** and  **$\Lambda$ -mapping**, **Delay** is the arrival time of the most critical primary output. The most critical output in the network generated by  **$\Lambda$ -mapping** was re-synthesized by  $\Delta$ -mapping, and the resulting arrival time is shown under **Delay of  $\Delta$ -mapping**. The delay unit is set to the minimum delay from an input pin to the output pin over all the gates

Name	SIS-1.2		SynFul					
			Δ-mapping			Δ-mapping		
	Delay	Area	Delay	Area	Time	Delay	Area	Time
cc	12.0	167.7	9.5	192.0	16.5	8.0	193.3	3.3
cm82a	13.2	86.0	10.5	58.3	8.9	7.0	78.7	8.8
pcle	15.0	266.7	12.0	156.7	35.7	10.5	176.7	11.5
pml	13.5	135.0	9.8	135.0	47.1	9.2	131.3	14.0
x2	17.7	151.0	9.8	109.7	58.0	9.8	109.7	24.1
cm163a	15.5	130.7	12.0	111.3	16.5	9.7	121.0	39.9
comp	32.5	401.3	26.0	502.7	193.4	21.2	454.3	43.5
cu	16.0	145.0	11.8	147.3	27.7	9.8	149.0	50.3
i1	12.7	183.0	11.5	132.3	24.2	9.8	111.0	62.3
cm151a	15.0	77.0	10.5	66.3	14.5	9.0	88.3	103.7
cht	9.2	459.7	9.2	320.7	10.7	8.2	323.7	2.8
ttt2	21.5	530.7	15.3	576.0	112.4	12.7	606.7	134.4
cm162a	12.7	164.7	12.0	111.3	21.4	9.7	122.7	154.8
term1	25.8	612.3	19.5	658.3	314.9	15.7	602.0	266.9

Table 1. Experimental Results (Timing-Driven Technology Mapping)

specified in the library. **Area** shows the area of the entire network obtained by each procedure, where the area unit is the smallest area among the library gates. **Time** is the CPU time in seconds measured on VAXstation 6000.

The results show that SynFul finds faster implementations than SIS-1.2 over all the examples tried. The results are remarkably better for some examples, such as x2, ttt2, and comp. Even though Δ-mapping already outperforms SIS-1.2, further improvements are usually possible by using the Δ-mapping procedure. Note that Δ-mapping effectively tries all the algebraic decompositions using the real cost function in technology mapping. Therefore, we consider the procedure especially effective when the arrival times of input signals vary and a decomposition for the fastest implementation cannot be easily identified by technology independent transformations. The CPU time is reasonable.

## 6. Conclusion

Conventional approaches distinguish three phases in logic synthesis: (1) technology independent processing, (2) AND2/INV decomposition, and (3) technology mapping. Although decisions made in the first two phases critically affect results, accurate delay, area, and loading information is not available until the third phase. Therefore, these critical decisions must be taken almost arbitrarily, producing inferior results. Our approach allows a key part of phase (1) and all of phase (2) to be combined with mapping.

The cornerstones of our approach are the mapping graph data structure and the graph-mapping algorithm. Built atop these pieces, the Δ-mapping proce-

dure combines the AND2/INV decomposition phase of synthesis into mapping. Applying  $\Lambda$ -mapping to a boolean network produces a result as good as can be obtained by applying tree-mapping to every AND2/INV decomposition. The  $\Delta$ -mapping procedure goes further by performing algebraic decomposition directly on a mapping graph during technology mapping. Applying  $\Delta$ -mapping to a boolean network is as effective as applying tree-mapping to every AND2/INV decomposition of every algebraic decomposition of the network. Our experiments suggest that  $\Lambda$ -mapping and  $\Delta$ -mapping improve results substantially over the conventional approach.

A theoretically attractive procedure, which we have not implemented, would apply both the distributive and associative transformations during mapping. In theory, such a procedure would be superior even to  $\Delta$ -mapping.

## Notes

1. As with ordinary techniques, the generalized technology mapping is also tree-based, and thus optimality is claimed on tree implementations. A heuristic is applied to generate non-tree implementations.
2. The present discussion assumes the set of possible costs is totally ordered; like tree-mapping, however, graph-mapping extends to partially ordered costs.

## References

- [1] C. Berman, J. Carter, and A. Day. The Fanout Problem: from theory to practice. In *Conference on Advanced Research in VLSI*, 1989.
- [2] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, Vol. 78(No. 2), 1990.
- [3] R. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. In *ISCAS*, 1982.
- [4] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Technology Mapping in MIS. In *ICCAD*, 1987.
- [5] J. Fishburn. A Depth-Decreasing Heuristic for Combinational Logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between. In *DAC*, 1990.
- [6] J. Fishburn. LATTIS: An Iterative Speedup Heuristic for Mapped Logic. In *DAC*, 1992.
- [7] A. Hoffman and M. O'Donnell. Pattern Matching in Trees. *Journal ACM*, 29(1), 1982.
- [8] R. Karp. Reducibility among Combinatorial Problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, 1972.
- [9] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *DAC*, 1987.
- [10] L. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic Decomposition during Technology Mapping. *submitted to IEEE Trans. CAD*, 1995.
- [11] P. McGeer, R. Brayton, and A. Sangiovanni-Vincentelli. Performance Enhancement through the Generalized Bypass Transform. In *ICCAD*, 1991.
- [12] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *ICCD*, 1992.
- [13] K. Singh. *Performance Optimization of Digital Circuits*. PhD thesis, U.C. Berkeley, 1992.

- [14] K. Singh. private communication, 1995.
- [15] K. Singh and A. Sangiovanni-Vincentelli. A Heuristic Algorithm for the Fanout Problem . In *DAC*, 1990.
- [16] K. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In *ICCAD*, 1988.
- [17] H. Touati. *Performance-Oriented Technology Mapping*. PhD thesis, U.C. Berkeley, 1990.
- [18] K. Yoshikawa, H. Ichiryu, H. Tanishita, S. Suzuki, N. Nomizu, and A. Kondo. Timing Optimization on Mapped Circuits. In *DAC*, 1991.

## Part IV

# ANALOG AND DIGITAL CIRCUIT DESIGN

Highlights in Analog and Digital Circuit Design and Synthesis at ICCAD <i>Ramesh Harjani, Philippe Magarshack, Gerard Mas and Rob A. Rutenbar</i>	269
An Interactive Device Characterization and Model Development System <i>(ICCAD 1984)</i> <i>Ebrahim Khalily, Peter H. Decher and Darrell A. Teegarden</i>	285
TILOS: A Posynomial Programming Approach to Transistor Sizing <i>(ICCAD 1985)</i> <i>J. P. Fishburn and A. E. Dunlop</i>	295
SPECS2: An Integrated Circuit Timing Simulator <i>(ICCAD 1987)</i> <i>Chandu Visweswariah and Ronald A. Rohrer</i>	303
Automatic Synthesis of Operational Amplifiers based on Analytic Circuit Models <i>(ICCAD 1987)</i> <i>Han Young Koh, Carlo H. Séquin and Paul R. Gray</i>	313
Analog Circuit Synthesis for Performance in OASYS <i>(ICCAD 1988)</i> <i>Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley</i>	325
Extraction of Gate-Level Models from Transistor Circuits by Four-Valued Symbolic Analysis <i>(ICCAD 1991)</i> <i>Randal E. Bryant</i>	337
Optimization of Custom MOS Circuits by Transistor Sizing <i>(ICCAD 1996)</i> <i>Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill and Chandu Visweswariah</i>	347

# HIGHLIGHTS IN ANALOG AND DIGITAL CIRCUIT DESIGN AND SYNTHESIS AT ICCAD

Ramesh Harjani

*University of Minnesota*

*Minneapolis, Minnesota 55455*

Philippe Magarshack and Gerard Mas

*ST Microelectronics*

*Crolles, France 38926*

Rob A. Rutenbar

*Carnegie Mellon University*

*Pittsburgh, Pennsylvania 15213*

## Abstract

When ICCAD began in 1983, we had no robust tools for device modelling, analog circuit synthesis, electrical timing simulation, transistor-to-logic abstraction, or large-scale custom circuit tuning. Today, all these techniques are in common industrial usage, most commercially available. The seven papers in this section fundamentally transformed the way in which we model, manipulate, and solve these difficult tasks today. They are linked by a common thread of “deep circuits innovation”.

## 1. Introduction

At first glance, the papers in this section might appear to be rather, well *eclectic*, spanning as they do a range of circuit-related topics from modelling to synthesis to verification to extraction. However, it would be a profound disservice to the papers selected in this section to impute that they lack some deep connection. In this *circuits* area, the core contributions are all about creating the essential, novel problem abstraction that transforms a difficult electrically oriented task into something that now supports efficient synthesis, or accurate modelling, or robust optimization, or deep verification. The papers in this section all share this defining trait: they introduce a deep, novel model of the problem that allows us to make new progress on design automation. Before these abstractions, efforts existed in each of these areas—but those efforts lacked rigor, robustness, scope. After these papers, the way in which we viewed these difficult tasks was fundamentally transformed.

This section describes seven papers chosen from ICCAD that are linked by this “deep circuits innovation” metric of excellence. They are, in chronological order:

- 1 *TECAP2: An Interactive Device Characterization and Model Development System* by Ebrahim Khalily, Peter H. Decher and Darrell A. Teegardin appeared in ICCAD 1984 [1]. Today, it is a bedrock principle of good design that good circuits only result from fabrication technologies with well-characterized devices and correspondingly accurate device models. Today, a deep infrastructure of model extraction, model fitting , and model validation tools exists. In 1984, this was a radical idea. TECAP2 was among the very first efforts to show how one might build design automation tools to help circuit designers with this fundamental task.
- 2 *TILOS: A Posynomial Programming Approach to Transistor Sizing* by Jack Fishburn and Al Dunlop appeared in ICCAD 1985 [2]. Today, everyone who works on circuit optimization knows what *posynomial* means, and understands that one can create efficient, large-scale nonlinear programming problems that can model how MOS device sizing affects delay. This understanding, however, all dates to this single paper with its elegant and powerful geometric programming model of the MOS sizing problem, and its accompanying efficient solution technique. It is not a stretch to argue that TILOS revolutionized the way we treat large digital netlists, paving the way for today’s large-scale convex-solver device sizing tools.
- 3 *Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models* by Han Young Koh, Carlo H. Sequin and Paul R. Gray appeared in ICCAD 1987 [3]. 1987 was a banner year in the (relatively short) history of analog synthesis. Several important papers appeared that defined the first serious attempts at synthesis for analog. The tool described in this paper, OPASYN, was one of these pioneering efforts, showing a linked set of abstractions for topology selection, sizing, and first-cut layout for op-amps.
- 4 *SPECS2: An Integrated Circuit Timing Simulator* by Chandramouli Visweswariah and Ronald A. Rohrer appeared in ICCAD 1987 [4]. By the end of the 1980s, classical SPICE simulation engines were showing their age, and were increasingly incapable of scaling to the very large designs appearing in this decade. Various piecewise linear or discrete models had appeared to attack this problem, but they lacked any rigorous foundation and were plagued with accuracy problems. SPECS2 introduced one of the very first numerically solid formulations for fast timing simulation of large digital circuits.

- 5 *Analog Circuit Synthesis for Performance in OASYS* by Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley appeared in ICCAD 1988 [5]. Analog circuit designers are nothing if not skeptical creatures. Just as the very first generation of analog synthesis papers appeared in 1987, the first volley of questions came forth from potential users, asking “well, can we do real high-performance designs with these tools?” The tool described in this paper, OASYS, was among the first to answer this question with a solid “yes”.
- 6 *Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis* by Randal E. Bryant appeared in ICCAD 1991 [6]. It sounds ever so simple: I have an MOS netlist representing a digital design, I would like to extract an equivalent gate-level description so that I might simulate this design more quickly. Oh, and by the way, don’t bother me about any nasty electrical issues like bidirectional transistors or charge sharing or multiple signal strengths. *Just do it.* The TRANALYZE tool introduced in this paper was the first to be able to do this extremely important form of extraction in a robust way. Prior efforts all failed on one difficult circuit or another. TRANALYZE was the first algorithm to handle this problem in a robust fashion, with an elegant, rigorous theoretical foundation.
- 7 *Optimization of Custom MOS Circuits by Transistor Sizing* by Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, and Chandu Visweswariah. ICCAD 1996 [7]. Sometimes, there is simply no substitute for the *best* design. In applications such as microprocessors, individual circuit blocks are extensively-and laboriously-tuned for optimal power and delay. There is no substitute here for the accuracy of a real circuit simulator to judge the quality of the final solution. This paper introduced techniques to couple state-of-the-art gradient optimizers (based on powerful trust-region methods) with state-of-the-art circuit simulators, for the purpose of efficient, automated local improvement of maximum-performance circuit blocks. The resulting tool, JiffyTune, was only the first of a series of powerful optimizers built and used with great success to optimize custom circuit tuning.

In the remainder of the paper, we describe each paper in a bit more detail, and connect them briefly to other important work in their area.

## 2. Fitting Device Models

The first paper in this section, *TECAP2: An Interactive Device Characterization and Model Development System* by Ebrahim Khalily, Peter H. Decher and Darrell A. Teegarden appeared in ICCAD 1984 [1]. The semiconductor industry

is moving to nanometer device geometries involving physical phenomena that are hard to model. For many circuits now operating at high frequencies using DC device models, *albeit* accurate DC models, are no longer sufficient. Simultaneously, accurate models and process statistics are critical to the convergence of circuit simulation. Furthermore, time-to-market pressures on the overall development cycle place device models on a critical path and push modelling engineers to sometimes provide models prior to performing any measurements on silicon [8, 9]. The actual development of device models is a critical subject and an excellent reference for MOS device modelling is [10].

More than 15 years ago, the TCAP2 tool generated a new technology that has prepared the industry to face our existing challenges. The innovation of this paper stands in the fact that this modelling tool is device independent and within a single environment combines all functions necessary to model a device wrapped up around a user interface.

- *Device independent:* TCAP2 can use built-in models, yet providing users with sufficient flexibility in order for them to create their own. This flexibility was a major step forward considering that device modelling tools used to be closed and proprietary.
- *Integrated:* TCAP2 has the ability to characterize a device, extract a model, and then to graphically analyze and compare the simulated result with the data measured on different geometries.

The productivity gain pioneered by the Hewlett Packard's team, has formed the basis to tackle the current 90 nanometer process challenges. Many current models use more than 150 parameters. For the past 10 years, the full process database has grown from 50KB to 50MB and the number of experimental devices to be characterized have increased from 5 to more than 30. The overall modelling technology process inner loop requires an average work phase of six weeks. In order to adapt to process technologies variation, modelling engineers need the flexibility to modify and extend model parameters beyond those offered by standard models. To control these variations, the state-of-the-art tools provide device designers and process engineers with accurate models and statistical analysis capabilities helping them to determine nominal performance as well as corner cases.

Modern device modelling software such as UTMOST from Silvaco [11], or the TCAP2's next generation IC-CAP (Integrated Circuit - Characterization and Analysis Program) from Agilent [12] demonstrates more capabilities as well as better performance. Data acquisition, instrument control, parameter extraction, graphical analysis, simulation, optimization, powerful characterization, and statistic analysis are integrated into a single software environment.

TECAP2 was the foundation of a new generation of modelling software that has allowed what was previously a highly skilled hand crafted process into a high quality production software.

### 3. Optimal Transistor Sizing for Digital Circuits

The second paper in this section, *TILOS: A Posynomial Programming Approach to Transistor Sizing* by Jack Fishburn and Al Dunlop appeared in ICCAD 1985 [2]. The seventh paper in this section, *Optimization of custom MOS circuits by Transistor Sizing* by Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, and Chandu Visweswariah appeared in ICCAD 1996 [7]. Both of these papers deal with finding digital device sizes to optimize a number of circuit characteristics including timing, power, area etc. However, they used two different approaches. TILOS uses a equation-based approach while Jiffy-Tune uses a simulation-based approach. In the equation-based approach a set of equations model the circuit's behavior. These equations can either be derived automatically or hand-crafted. In the simulation-based approach the circuit's behavior is obtained through numerical simulation.

Complexity and time-to-market pressures urge design teams to use high-level languages and synthesis tools in order to incorporate more and more functionality on the same silicon to reach System-On-Chip complexities. This traditional design flow improves the design entry productivity. However, a final solution can only be found provided that some critical functions are optimized in terms of size, performance and power. In order to meet their specifications, dedicated custom circuit design tools were required. New process technologies accelerate the integration, which increase the size and the complexity of these critical functions. Custom blocks consist of hundreds of transistors and the associated efforts needed to implement an optimal solution is non negligible.

The TILOS [2] paper included in this commemorative selection was the pioneering research that changed all that. Even though previous attempts had been made to optimize transistor sizes to meeting timing requirements [13, 14, 15, 16, 17, 18, 19], this paper provided a technique to cast the problem into a convex posynomial form that ensures that we can find the global minima, i.e., a globally optimal solution. Convex programs have the property that any local optimum is certain to be globally optimal which is not the case for non-convex forms [20]. Consequently, non-convex problems face major hurdles with gradient-based schemes such as getting trapped in local minima, combinatorial (NP-complete complexity) and non-linearity with multiple minima in feasible solution spaces [21].

Nearly 15 years after the TILOS publication, the posynomial programming approach to transistor sizing remains the cornerstone of the equation-based optimization approach. Even though standard Geometric Program form is non-

convex, it can be remodelled into a convex form by a simple change of variables: Each  $x_i$  is transformed to its natural logarithm by replacing it with  $\exp(z_i)$ . As a result, the posynomial form allows for efficient geometric programming techniques for circuit sizing and optimization, thus widening the influence of the method. Should circuit characteristics not be castable into a posynomial function, equations can be approximated to some posynomial format and this is a one-time manual effort.

As today's transistors sizing optimization algorithms cannot focus on delay alone independent of power consumption. The power consumption is currently one of the most significant challenges designers have to face. To tackle this problem, system level techniques such as low power modes and gated clock techniques are widely used. However, the situation is desperate enough to take drastic measures like special processes and very low voltages. In the future, custom low power circuits should play a major role in order to address the problem relating to the power versus the performance trade-offs optimization [22, 23, 24].

The TILOS work has provided the VLSI industry with a solid base for transistor sizing to meet timing and other requirements. This single paper showed that it was possible to cast even very large problems using an elegant and powerful device sizing model using geometric programming techniques. It is not a stretch to argue that TILOS revolutionized the way we treat large digital netlists, paving the way for today's large-scale convex-solver device sizing tools.

The seventh paper in this section, *Optimization of custom MOS circuits by transistor sizing* by Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, and Chandu Visweswariah appeared in ICCAD 1996 [7]. A well optimized circuit provides excellent leverage against the competition as long as it does not affect the time-to-market. Therefore, the industry needs to develop more tools in this field in order to help designers to improve their productivity. By combining technologies of a fast circuit simulator SPECS and a nonlinear optimizer LANCELOT, JiffyTune is the bedrock of this new technology.

The JiffyTune system architecture integrates all functions necessary to automatically design and optimize hardware blocks within a single software environment. Even though the JiffyTune interface still requires an experienced user, it hides the custom-designed circuit optimization complexities and offers an interesting future evolution for the tool.

The JiffyTune technology tackles 3 major problems encountered during custom circuit design:

- *Productivity*: As per Table 2 in [7], “COLD” run (started from an untuned circuit) as compared to manually tuned circuits is very similar. Those results are very encouraging and illustrate the capacity of the tool, i.e., no *a priori* hand tuning is required.

- *Flexibility*: For various reasons, specifications and technology processes change during the design phase affecting circuit optimization. On the one hand, it could induce custom designers to start their manual work from scratch, while, on the other hand, JiffyTune users can minimize the cost by rerunning the transistor size optimization after the tool's parameters and input specifications have been updated.
- *Accessibility*: The tool's power along with the user interface allows designers to work with limited experience and rapidly obtain an excellent result.

The quality and the productivity gain demonstrated by Jiffytune, pioneered a flexible and efficient custom-design circuit methodology that can be targeted to multiple foundries and processes.

#### 4. Design and Synthesis of Analog Circuits

The third paper in this section of the commemorative selection is *Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models* by Han Young Koh, Carlo H. Sequin and Paul R. Gray from the University of California, Berkeley. It was published in ICCAD in 1987 [3]. This paper provides the first description of the *OPASYN* op-amp synthesis tool. The fifth paper in our analog and digital circuit design commemorative selection is *Analog Circuit Synthesis for Performance in OASYS* by Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley from Carnegie Mellon University. It was published in ICCAD in 1988 [5]. The high performance design capabilities and their impact on analog circuit synthesis in the *OASYS* analog circuit synthesis system was first mentioned in this paper.

By the mid to late eighties, CMOS feature sizes had shrunk to below  $3\mu\text{m}$  and large digital chips were routinely synthesized from high level hardware description languages. As feature sizes continued to shrink, it was becoming amply evident that analog portions of mixed-signal systems on a chip (SoC) were taking significantly more time to design even if they involved fewer devices and occupied a smaller percentage of the chip. In fact, anecdotal examples from ATT Bell Labs provided many examples of mixed-signal SoCs where less than 10% of the mixed-signal chip that was analog took more than 90% of the design time [25]. The digital design process was significantly more formalized and followed a strict hierarchy which made it possible to ‘automatically’ synthesize such large circuits. Digital circuits are routinely designed hierarchically as HDL synthesis, gate-level synthesis and back-end physical design. Additionally, timing closure is attempted at various steps in the process.

However, at that time, and to a large extent even at the current time, the analog portions of such SoCs are largely still designed by hand where a designer first

generates a circuit schematic with device sizes which is then followed by a number of SPICE and ‘tweak’ steps. With the result that the analog portions of SoCs tend to be the limiting factor in the design process of such systems. Time to market pressures and the desire for working first silicon has motivated the research into analog circuit synthesis systems. However, unlike digital circuits the strong interaction between device and process characteristics have not allowed for an easy decoupling of various design steps. The relative maturity of digital CAD techniques and relative independence of digital design and fabrication process parameters suggests that whenever possible analog circuits should be replaced by their digital counterparts. However, given that we have to interface with the analog external world, SoCs will always contain some analog and mixed-signal circuits continuing to motivate the need for CAD for analog circuits.

The overall analog circuit design flow can roughly be broken up into three major steps: a) topology selection or schematic synthesis, b) device sizing and c) layout or physical design. Research has focused on all three aspects of the design flow. The set of papers in this section of this commemorative selection focuses on the first two steps of the design flow. However, significant progress has also been made in the physical design aspects of analog circuits [26, 27, 28, 29, 30].

The various approaches that have been used to tackle the analog circuit synthesis problem can roughly be classified into two groups.

- **Knowledge-based methods:** Early in the analog CAD research process it was realized that analog circuit design was different from the digital design process. In fact, analog and RF design has often been referred to as a ‘black art’ where only a handful of expert designers are successful at generating high performance circuits. So, some of the early analog circuit synthesis research has focused on roughly mirroring the ‘expert design flow’. Analog designers use simplified equations for known circuit blocks and follow a simple design plan. Some of the earliest successful research using this approach include [31, 32, 5, 33, 34, 35, 36, 37, 38]. These design methods have been applied to a variety of analog and RF circuits and generally provide results that are comparable to hand-crafted designs. Unlike a number of numerical optimization techniques discussed next, knowledge-based techniques have been used for both topology selection and for device sizing [32]. However, the primary weakness of this approach lies in the coding of the design knowledge. Symbolic analysis techniques have later been developed in an attempt to automatically generate the analytic equations [39]. Among these, the *OASYS* system [5] included in this section is one of the earliest pieces of research in this area. The *OASYS* system uses a strict hierarchy from system, circuit, block and component level. Measurements from fabricated circuits using designs generated by the *OASYS* system have been shown to match predicted values and have shown performance comparable to hand-crafted

designs [32]. The ICCAD paper [5] showed that it was possible to synthesize high performance analog designs.

- *Numerical optimization-based methods:* A number of traditional numerical techniques, such as steepest descent and minmax formulations, have been used to tackle analog circuit design. Most of them have focussed on the device sizing problem including the first paper in this section of the commemorative selection [3, 40, 41, 42, 43, 44, 45]. *OPASYN* [3] uses simplified analytical models to model circuit performance. The program selects an op-amp topology and generates device sizes and bias currents to meet a given set of design specifications [3]. The use of analytical models to represent behavior, including large signal parameters such as slew-rate, ensure that the optimization process is extremely fast. However, the OPASYN system, as with other traditional numerical technique based approaches can get ‘stuck’ in local optima and does not always provide a solution even if there is one. To circumvent this problem variations of classical numerical techniques have been developed, including branch and bound [46], simulated annealing [47, 48] and geometric programming [49]. Interestingly, the branch and bound technique used in [46] is used for topology selection rather than for device sizing. Simulated annealing is another popular technique that in principle avoids local minima by initially accepting solutions that aren’t necessarily better at each program step. However, simulated annealing does *not guarantee* a globally optimal solution nor can it guarantee that there is no solution when one is not found. The geometric programming method developed in [49] casts the analog device sizing problem using posynomial functions that are convex and can be solved quickly and globally. The primary problem that remains is to now cast the design problem into posynomial form.

Analog circuit design continues to be an active research topic suggesting a problem that is significantly less tractable compared to digital synthesis. However, after almost twenty years of active research it is extremely gratifying to see the successful establishment of startup companies that are focused on providing analog circuit synthesis tools [50, 51].

## 5. Timing Simulation

The fourth paper in this section *SPECS2: An Integrated Circuit Timing Simulator* by Chandramouli Visweswarah and Ronald A. Rohrer was published at ICCAD in 1987, and presented a method for fast timing simulation and modelling [4]. This period saw great research thrusts in the direction of developing fast simulators that could provide large speedups over SPICE while maintaining high accuracy. Like prior efforts such as SPECS [52] and MOTIS [53], SPECS2 also used discretized device models, but was able to overcome spuri-

ous algorithmic oscillations seen in earlier methods. The Simulation Program with Integrated Circuits Emphasis (SPICE) has been the industry standard for more than 25 years. Although transistor level circuit simulators are very accurate, they are nevertheless very slow. The ever-increasing number of modern circuits transistors is making the simulation of very large circuits impossible due to the inherent complexity of the circuit models.

The publication [4] introduces a different approach to the simulation problem emphasizing that the simulation effectiveness be not only focused on its accuracy, but also on both its feasibility and its productivity. Less accurate than SPICE, +/- 5 percent, the Simulation Program for Electric Circuits and Systems (SPECS) is about 70 times faster. Nevertheless, the fast simulation, per se, enables the simulation of bigger circuits. Yet, the SPECS simulator offers more than simulation acceleration. It is also a different way of thinking and a new approach showing that the results achieved with the simulation accuracy can be traded off with those of the simulation speed, especially in regard to MOS circuits.

More technologies have derived from this precursor work. By combining switch-level with event driven technologies, simulators such as the IRSIM from Berkeley have been found to be 1,000 times faster than SPICE demonstrating a reasonable accuracy in terms of delay and power estimation for the MOS circuits.

More than 10 years after SPEC2, CAD companies have launched a new generation of simulators maintaining the spirit of SPEC2. Cadence introduced their Affirma tool, Accelerated Transistor-level Simulator (ATS), in June 1999, Mentor Graphics introduced Mach TA in June 2000. The current market leaders include Nassda with HSIM and Synopsys, which announced Nanosim in May 2001.

Netlists consisting of up to several dozen millions of transistors can be managed and are able to run more than 1,000 times faster than Spice with an accuracy trade-off of just a few percent. These simulators provide the low-power designers with the information they need in order for them to optimize the performance of their designs. Mixed-signal netlists and memories are supported, delivering a high-speed and a high-capacity verification for complex chip designs. Lookup table and model-in techniques have improved the third generation of transistor level simulators.

The SPECS2 work has had a significant impact on the state of simulation, and was part of a wide range of efforts during and after that time, including [54] and [55]. Its remarkable combination and speed and accuracy has resulted in its use in the inner loop of circuit optimizers, such as in the development of fast circuit optimization tools, such as in [7] and [56]. The practical impact of this work is visible in its widespread use in industry, particularly at IBM. In 1987, Visweswarah and Rohrer led the way of this new simulators generation by

settling as the precursors of a different approach [57, 58, 59, 60, 61, 62, 63, 64].

## 6. Extracting Gate-Level Models

The sixth paper in this section, *Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis* appeared in ICCAD 1991 [6]. Every year System-On-Chip has become increasingly complex while the associated number of transistors has grown exponentially. Consequently, traditional transistor simulators were not able to cope with the increased complexity and reached the limit of their capability. Although electrical simulators have dramatically improved the simulation time, they did not catch up with the level of complexity, thus remaining limited. In order to be able to simulate a complete system-on-chip at transistor level, the industry was anticipating the development of a new technology. This publication appeared as the cornerstone of new activities that have changed the full functional verification flow [65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75].

Today, the transistor abstraction technique is based upon two different methods: pattern matching and the switch-level symbolic analysis as described in the publication. One of the best advantages that pattern matching has to offer stands in the fact that both digital and analogue designs can be extracted. However, the efficiency only relates to the quality of the associated library. The switch-level symbolic analysis is very effective as the binary decision diagrams are applied to represent and manipulate the four-value functions. It can handle any combinational digital designs in a way that is totally design flow and design style independent, which is one of the strongest assets of the technique.

Based on the technology described in the paper, the “laybool” tools for transistor abstraction was developed in 1994 [71]. The tool has combined the switch-level analysis with other techniques in order to be able to deal with more complex circuits. After the first switch-level analysis was carried out, new algorithms were implemented handling flip-flops, latches, arithmetic operators, RAM, dynamic logic, precharged logic etc.

The transistor abstraction technology is now integrated within the classic functional verification flow promoting the RTL sign-off. Designers first extract a gate-level description from the transistor netlist. Combined with the equivalence checking technology, the abstracted netlist is then matched up with the RTL model. This method fully certifies the layout view equivalence with the RTL level model. The same abstracted gate-level netlist is ported on a hardware accelerator or emulator to test the reset mechanism and makes the extraction of complex chips such as a complete 64-bit processor possible.

Time-to-market is continuing to put pressure on the development team. IP reuse is one of the solutions the design community has found to improve their

productivity. A top-level model simulated early in the design cycle is a key factor to reduce the development time. Despite corporations efforts to normalize IP interfaces, only blocks developed recently abide by these strict rules, thus increasing the design efforts but facilitating the reuse. IP blocks developed several years ago are part of the companies asset even though they sometimes only offer a layout view. The use of the transistor abstraction technique enables a gate-level netlist to be created as well as all of the legacy blocks to be integrated within the new design flow.

The laytool tool is currently commercialized as Lynx-LB by Avant!. Since then, other companies such as Verplex “transformal-LTX” and Avertec “Yagle” (derived from a PARIS/LIP6 work) [75], based on the switch-level analysis, have developed similar tools. TNI-Valiosys have launched the “TLL” combining pattern matching and switch-level analysis techniques. Notwithstanding the fact that transistor abstraction tools are integrated in most modern tool flows, they still are not a push button technology, as they may need to be adapted to different design styles. Still, Randal E. Bryant with his paper [6] is a pioneer in a technology that has significantly impacted the microelectronic industry.

## 7. Conclusions and Acknowledgements

When ICCAD began in 1983, we had no robust tools for device modelling, analog synthesis, electrical timing simulation, transistor-to-logic abstraction, or large-scale custom circuit tuning. Today, all these techniques are in common industrial usage, most commercially available. The seven papers in this section fundamentally transformed the way in which we model, manipulate, and solve these difficult tasks today. They are all linked by a common thread of “deep circuits innovation”.

The authors wish to acknowledge the valuable discussions and input from various friends and colleagues. In particular, the help provided by Prof. Sachin Sapatnekar of the University of Minnesota.

## References

- [1] Ebrahim Khalily, Peter H. Decher and Darrell A. Teegarden, “TECAP2: An Interactive Device Characterization and Model Development System”, in *International Conference on Computer-Aided Design*, pp. 149–151, 1984.
- [2] Jack Fishburn and Al Dunlop, “TILOS: A Posynomial Programming Approach to Transistor Sizing”, in *International Conference on Computer-Aided Design*, pp. 326–328, 1985.
- [3] H. Y. Koh, C H. Sequin and P. R. Gray, “Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models”, in *International Conference on Computer-Aided Design*, pp. 502–505, 1987.
- [4] Chandramouli Visweswariah and Ronald A. Rohrer, “SPECS2: An Integrated Circuit Timing Simulator”, in *IEEE International Conference on Computer-Aided Design*, pp. 94–97, 1987.

- [5] Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley, "Analog Circuit Synthesis for Performance in OASYS", in *International Conference on Computer-Aided Design*, pp. 492–495, 1988.
- [6] Randal E. Bryant, "Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis", in *International Conference on Computer-Aided Design*, pp. 350–353, 1991.
- [7] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill and C. Viswesvariah, "Optimization of custom MOS Circuits by transistor sizing", in *IEEE International Conference on Computer-Aided Design*, pp. 174–180, 1996.
- [8] A.J Scholten, R.v Langevelde, H.M.J. Boots, D.B.M Klaasen, G.Gouget and A.Juge, "BSIM4 and MOS Model 11 Benchmark for MOSFET capacitances", in *CMC meeting*, March 2001.
- [9] G.Gouget, "Measurement conditions for Model Evaluations in 0.13um Technology (HC-MOS9 GPLL, J208SVX, wafer 12)", in *DM02.83*, May 2002.
- [10] Yannis Tsividis, *Operation and Modeling of the MOS Transistor*, McGraw-Hill, New York, 2nd Edition, 1998.
- [11] [www.silvaco.com](http://www.silvaco.com).
- [12] [www.agilent.com](http://www.agilent.com).
- [13] A.E. Ruehli, P. K. Wolf and G. Goertzel, "Power and Timing Optimization of Large Digital Systems", in *Proceedings of the IEEE International Symposium on Circuits And Systems*, pp. 402–405, 1976.
- [14] A.E. Ruehli, P. K. Wolf and G. Goertzel, "Analytical Power/Timing Optimization Technique for Digital System", in *Proceedings of Design Automation Conference*, pp. 142–146, 1977.
- [15] L.A. Glasser and L.P.J Hoyte, "Delay and Power Optimization in VLSI Circuits", in *Proceedings of the IEEE Design Automation Conference*, pp. 529–535, 1984.
- [16] K.S. Hedlund, "Models and Algorithms for Transistor Sizing in MOS Circuits", in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 12–14, 1984.
- [17] W.H. Kao, "ARIES, a Workstation Based Schematic Driven System for Circuit Design", in *Proceedings of the IEEE Design Automation Conference*, pp. 301–307, 1984.
- [18] M. Matson, "Optimization of Digital MOS VLSI Circuits", in *Proceedings Chapel Hill Conference on VLSI*, pp. 109–126, May 1985.
- [19] W.H. Kao, "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits", in *Proceedings of the IEEE Design Automation Conference*, pp. 781–784, 1985.
- [20] Y. Nesterov and A. Nemirovsky, "Interior- point polynomial methods in convex programming", *Studies in Applied Mathematics. SIAM*, vol. 13, 1994.
- [21] W. Orchard-Hays, *Advanced Linear Programming Computing Techniques*, McGraw-Hill, 1968.
- [22] J. Shyu, A. Sangiovanni-Vincentelli, J.P. Fishburn and A.E. Dunlop, "Optimization-Based Transistor Sizing", *IEEE Journal of Solid-State Circuits*, vol. 23, n. 2, pp. 400–409, April 1988.
- [23] K.J. Singh, A.R. Wang, R.K. Brayton and A. Sangiovanni-Vincentelli, "Timing Optimization of Combinational Logic", in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 282–285, 1988.
- [24] D Marple, "Transistor Size Optimization in the Tailor Layout System", in *Proceedings of the IEEE Design Automation Conference*, pp. 43–48, 1989.
- [25] L. Richard Carley and Rob A. Rutenbar, "How to Automate Analog IC Design", *IEEE Spectrum*, vol. 25, pp. 26–30, August 1988.

- [26] Hormoz Yaghutiel, A Sangiovanni-Vincentelli and Paul R. Gray, "A Methodology for Automated Layout of Switched-Capacitor Filters", in *International Conference on Computer-Aided Design*, pp. 444–447, 1986.
- [27] David J. Garrod, Rob A. Rutenbar and L. Richard Carley, "Automatic Layout of Custom Analog Cells in ANAGRAM", in *IEEE International Conference on Computer Aided Design*, pp. 544–547, 1988.
- [28] J. Rijmenants *et al*, "ILAC: An Automated Layout Tools for Analog CMOS Circuits", *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 417–425, April 1989.
- [29] Rob A. Rutenbar John M. Cohn, David J. Garrod and L. Richard Carley, "KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing", *IEEE Journal of Solid-State Circuits*, vol. SC-26, n. 3, pp. 330–342, March 1991.
- [30] Edoardo Charbon, Enrico Malavasi, Davide Pandini and Alberto Sangiovanni-Vincentelli, "Simultaneous Placement and Module Optimization of Analog IC's", in *ACM/IEEE Design Automation Conference*, pp. 31–35, 1994.
- [31] Marc G. DeGrauw and Willy M. C. Sansen, "A Synthesis Program for Operational Amplifiers", in *IEEE Internat. Symposium on Circuits and Systems*, pp. 18–19, 1984.
- [32] Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley, "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis", in *ACM/IEEE Design Automation Conference*, 1987.
- [33] M. Hashizume, H. Y. Kawai, K. Nii and T. Tamesada, "Design Automation System for Analog Circuits based on Fuzzy Logic", in *IEEE Custom Integrated Circuits Conference*, pp. 4.6.1–4.6.4, 1989.
- [34] M. G. R. DeGrauw, "Towards a Analog System Design Envioronment", *IEEE Journal of Solid-State Circuits*, vol. 24, June 1989.
- [35] Fatey El-Turky and Edward E. Blades, "An Artificial Intelligence Approach to Analog Circuit Design", *ICCAD*, vol. 8, n. 6, June 1989.
- [36] Z. Ning, T. Moutahaan and H. Wallinga, "SEAS: A Simulated Evolution Approach for Analog Circuit Synthesis", in *IEEE Custom Integrated Circuits Conference*, pp. 5.2.1–5.2.4, 1991.
- [37] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS op amp Synthesis by means of a Genetic Algorith", in *ACM/IEEE Design Automation Conference*, pp. 433–438, 1995.
- [38] S.K. Gupta and M.M. Hasan, "KANSYS: A CAD Tool for Analog Circuit Synthesis", in *International Conference on VLSI Design*, pp. 333–334, 1996.
- [39] Georges Gielen and Rob A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits", *Proceedings of the IEEE*, vol. 88, n. 12, pp. 1825–1854, December 2000.
- [40] H. Onodera, H. Kanbara and K. Tamaru, "Operational Amplifier Compilation with Performance Optimization", *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 466–473, April 1990.
- [41] Gani Jusuf, Paul R. Gray and A. Sangiovanni-Vincentelli, "CADICS - Cyclic Analog-to-Digital Converter Synthesis", in *IEEE International Conference on Computer-Aided Design*, November 1990.
- [42] P.C. Maulik, L. Richard Carley and D.J. Allstot, "High-Performance Analog Module Generation using Nonlinear Optimization", in *IEEE International ASIC Conference*, pp. T13.5.1–T13.5.2, 1991.
- [43] H. Chang *et al*, "A Top-Down Constraint-Driven Design Methodology for Analog Integrated Circuits", in *IEEE Custom Integrated Circuit Conference*, pp. 8.4.1–8.4.6, 1992.

- [44] P.C. Maulik, M.J. Flynn, L. Richard Carley and D.J. Allstot, "Rapid Redesign of Analog Standard Cells using Constrained Optimization Techniques", in *IEEE Custom Integrated Circuit Conference*, pp. 8.1.1–8.1.3, 1992.
- [45] J.P. Harvey, M.I. Elmasry and B. Leung, "STAIC: An Interactive Framework for Synthesizing CMOS and BiCMOS Analog Circuits", *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 1402–1417, November 1992.
- [46] P.C. Maulik, L. Richard Carley and Rob A. Rutenbar, "Integer Programming based Topology Selection of Cell-Level Analog Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 401–412, April 1995.
- [47] G.G.E. Gielen, H.C.C. Walscharts and W.M.C. Sansen, "Analog Circuit Design Optimization based on Symbolic Simulation and Simulated Annealing", *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 707–713, June 1990.
- [48] Emil S. Ochotta, Rob A. Rutenbar and L. Richard Carley, "Synthesis of High-Performance Analog Circuits in ASTRX/OBLX", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 273–293, March 1996.
- [49] Maria del Mar Hershenson, Stephen P. Boyd and Thomas H. Lee, "Optimal Design of a CMOS Op-Amp via Geometric Programming", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, n. 1, pp. 1–21, January 2001.
- [50] "www.neolinear.com".
- [51] "www.barcelonadesign.com".
- [52] A. J. de Geus, "SPECS: Simulation program for electronic circuits and systems", in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 534–537, May 1984.
- [53] B. R. Chawla, H. K. Gummel and P. Kozak, "MOTIS: An MOS timing simulator", *IEEE Transactions on Circuits and Systems*, vol. CAS-22, n. 12, pp. 901–910, December 1975.
- [54] E. L. Acuna, J. P. Dervenis, A. J. Pagones and R. A. Saleh, "iSPICE3: a new simulator for mixed analog/digital circuits", in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 13.1/1 –13.1/4, May 1989.
- [55] A. Devgan and R. A. Rohrer, "Adaptively controlled explicit simulation", *IEEE Transactions on Computer Aided Design*, vol. 13, n. 6, pp. 746–762, June 1994.
- [56] A. R. Conn, I. M. Elfadel, W. W. Molzen Jr., P. R. O'Brien, P. N. Strenski, C. Visweswarah and C. B. Whan, "Gradient-based optimization of custom circuits using a static-timing formulation", in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 452–459, June 1999.
- [57] Randal E. Bryant, "An algorithm for MOS logic simulation", in *Lambda, the Magazine of VLSI Design*, pp. 46–53, 1980.
- [58] C. Y. Chu, *Improved Models for Switch-Level Simulation*, PhD thesis, Stanford University, October 1988.
- [59] R. A. Rohrer, "Circuit partitioning simplified", *IEEE Transactions on Circuits and Systems*, pp. 2–5, January 1988.
- [60] A. Salz and M. A. Horowitz, "IRSIM: An incremental MOS switch-level simulator", in *In Proceedings of the 26th Design Automation Conference*, pp. 173–1785, June 1989.
- [61] Chandramouli Visweswarah, Peter Feldmann and Ronald A. Rohrer, "Incorporation of inductors in piecewise approximate circuit simulation", in *In International Conference on Computer-Aided Design*, pp. 162–165, November 1990.
- [62] L. T. Pillage and R. A. Rohrer, "Asymptotic Waveform Evaluation for Timing Analysis", *IEEE Transactions on Computer-Aided Design*, pp. 352–366, April 1990.
- [63] C.L. Ratzlaff, N. Gopal and L.T. Pillage, "RICE: Rapid Interconnect Circuit Evaluator", in *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 352–356, June 1991.

- [64] Alexander D. Stein, Tuyen V. Nguyen, Binay J. George and Ronald A. Rohrer, "ADAPTS: A digital transient simulation strategy for integrated circuits", in *In 28th ACM/IEEE Design Automation Conference*, pp. 26–31, June 1991.
- [65] M. Laurentin, A. Greiner and R. Marbot. Desb, "A functional abstractor for CMOS VLSI circuits", in *IEEE European Design Automation Conference*, 1992.
- [66] S. Jain, R.E. Bryant and A. Jain, "Automatic clock abstraction from sequential circuits", in *2nd ACM/IEEE Design Automation Conference*, June 1995.
- [67] N. Halbwachs and F. Maraninchi, "On the symbolic analysis of combinational loops in circuits and synchronous programs", in *Euromicro*, September 1995.
- [68] T. Kam and P.A. Subrahmanyam, "Comparing layout with HDL models: A formal verification technique", *IEEE transactions on Computer Aided Design*, 1995.
- [69] Michael Keating and Pierre Bricaud, *Reuse Methodology Manual for SoC Designs*, Kluwer Academic Publishers, 1998.
- [70] C. Mc Donald, *Workshop on symbolic methods and applications to circuit design*, October 1998.
- [71] T. Goust, M. Bartley, G. Barett and F. Rocheteau, "Formal methods accelerate circuit-level verification", *Electronics Journal*, August 1998.
- [72] R.E. Bryant, "Graph based algorithms for Boolean function manipulation", *IEEE Transactions on Computers*, 1986.
- [73] R.E. Bryant, "Extraction of gate level models from transistor circuits by four-valued symbolic analysis", in *International Conference on Computer-Aided Design*, pp. 350–353, 1991.
- [74] R.E. Bryant, "Boolean Analysis of MOS circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 634–649, July 1992.
- [75] Bazargan Sabet Pirouz and Greiner Alain, "YAGLE: a Second generation Functional Abstractor for CMOS VLSI Circuits Lester Anthony", in *10th International Conference on Microelectronics*, pp. 265–268, December 1998.

# **AN INTERACTIVE DEVICE CHARACTERIZATION AND MODEL DEVELOPMENT SYSTEM**

Ebrahim Khalily<sup>1</sup>, Peter H. Decher<sup>2</sup> and Darrell A. Teegarden<sup>3</sup>

*Engineering Productivity Division, Hewlett Packard*

*19420 Homestead Road*

*Cupertino, CA 95014*

## **Abstract**

A computer aided design system for integrated circuits is not complete without accurate circuit simulation capabilities. Accurate simulation is not possible without accurate models and model parameters. TECAP2 (Transistor Electrical Characterization and Analysis Program) is a program that greatly simplifies the development of models, the extraction of model parameters from measured data, and the comparison of the effectiveness of various models.

## **1. Introduction**

The integrated circuit designer relies on circuit simulation to verify circuit performance. Circuit simulation accuracy is directly affected by transistor modeling accuracy. Transistor models are made up of mathematical equations that contain constants that are fixed for a given fabrication process; these constants are called model parameters. The actual values of the model parameters, as well as the model equations, determine the extent to which a model can accurately predict device electrical characteristics over a range of device geometries. TECAP2 provides an environment for developing accurate models and for extracting model parameters values from measured data. The TECAP2 system evolved from the original TECAP [1], which was developed at Stanford University in the late 1970's.

## **2. TECAP2 SYSTEM**

### **2.1 User Interaction**

TECAP2 was designed with the objective of providing ease of use for the occasional user, as well providing advanced features and capabilities for the expert user. A simple menu structure has been implemented that avoids the problem of getting lost in menu hierarchies. The TECAP2 system commands

---

Authors are currently <sup>1</sup>an independent consultant, with <sup>2</sup>Teseda Corporation and with <sup>3</sup>Mentor Graphics Corporation.

are organized into several subsets, as shown in Figure 1. The major topics of

TECAP2 Main Menu		Subset
D)	Device data	S) S1, S7
C)	Connections	S1) Simulate
U)	Use setup	S2) Int.act.sim.
M)	Measure	
E)	Extract	S4) Select model
S)	Simulate	S5) Enter parameters
O)	Output control	S7) Plot data + axis
P)	Plot control	S8) Plot data only
F)	Filer	S9) Print data
B)	Build Setup	S10) Plot 2nd output
I)	Input sequence	S11) Plot 3rd output
Q)	use seQuence	
A)	Auxiliary	S12) Save data

Figure 1. TECAP2 command menu.

interest are displayed in the main menu and subtopics are displayed in the subset menu. The subset menu changes as the user steps through the main menu topics by typing the letter of the command.

The user prepares the system for a task (such as measurement, simulation, or extraction) by selecting the appropriate command. If information is needed to perform the task (such as start, stop, and increment values), the user is supplied with a table or diagram to edit. Every table or diagram has a set of default values or configurations. The table entry method allows the user to have random access to the specifications of the task. Typical menu driven systems require the user to serially dredge their way through a series of menus, even when only one item is to be changed. The complete set of configuration information, for the entire system, can be saved or retrieved with a single command. This makes it possible for any user to customize the system by configuring it for a specific purpose and saving the configuration on a disc file; any other user can then reload the configuration and thus also have a customized turnkey system.

The TECAP2 system can be operated without extensive knowledge of the computer hardware, the computer operating system, or the instruments. The system automatically adapts to handle whatever instruments are connected to the system (i.e., the user specifies capacitance measurements in the same way regardless of what specific model of capacitance meter is on line).

The system outputs can be directed to external output devices, such as printers, plotters, external monitors, and external graphics displays. The directing of the outputs is controlled with the same menu commands as the other system commands.

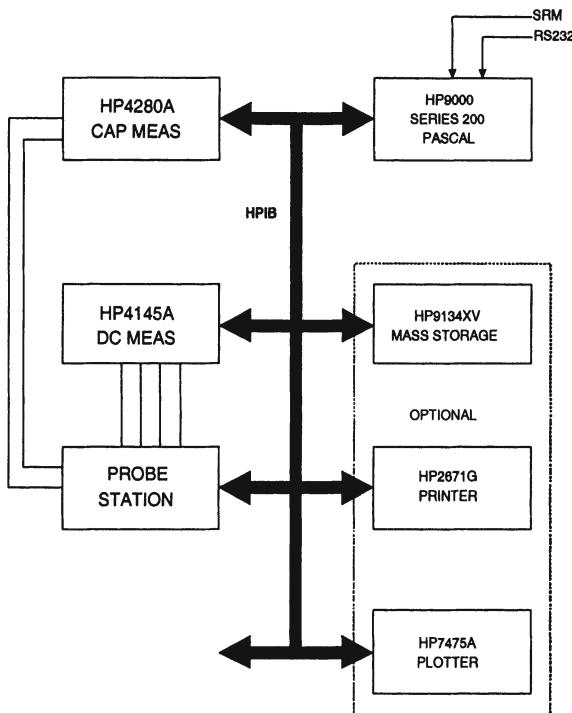


Figure 2. TECAP2 Hardware block diagram.

## 2.2 System Hardware

The TECAP2 system combines the computing power of the HP9000 series 200 PASCAL workstation with the measurement capabilities of a variety of advanced instruments. The TECAP2 measurement hardware can be configured either as a benchtop system using the HP4145A Semiconductor Parameter Analyzer for DC measurements and the HP4280A Capacitance Meter for CV characterization (Figure 2), or as a more sophisticated system using the HP4062A Semiconductor Parametric Test System hardware. The HP4062A system features the addition of an automatic connection matrix and a wafer prober. In addition, TECAP2 supports the Shared Resource Manager (a shared disc network) as well as an RS-232 data link to provide a communications interface to other systems.

## 2.3 System Capabilities

**Measurement.** A wide variety of measurements can be performed on a given device. Constant and swept stimuli may be applied to any terminal of the device under test for DC characterization. The resulting currents and voltages

at any of the device ports may be monitored. AC characterization is available in the form of capacitance versus voltage measurements.

**Simulation.** Although similar to the SPICE circuit simulator in function, the TECAP2 simulator is restricted to a one-transistor simulator, designed to simulate the electrical characteristics of a four-terminal device model for comparison with corresponding measured characteristics. The simulator is device model independent and works equally well for MOS and Bipolar type device models. The model structure used in TECAP2 is the same as that used by HP Modular SPICE [2]. This model structure greatly simplifies the task of implementing new models by consolidating all of the model routines. The TECAP2 simulator will produce single transistor results that are identical to those obtained using a circuit simulator, providing the same model has been implemented in both TECAP2 and the circuit simulator.

The simulator takes the same input instructions as the measurement part of the program. It is therefore easy to generate simulated device characteristics which can be graphically compared to the corresponding measured data. An interactive simulation mode allows the user to vary parameters, one at a time, and graphically display the resulting simulation in real time. This feature allows the user to quickly isolate the effect of an individual parameter on the simulated device characteristics.

**Extraction.** The TECAP2 parameter extraction capability is model independent, allowing use of the same system to model many different types of devices. TECAP2 supports a three-step parameter extraction methodology to extract device model parameters. This approach generates an accurate set of model parameters in a minimum amount of time, while still providing model independence.

The first step in the parameter extraction methodology is to identify the regions of device operation to be modeled. Measured data can then be collected for each of these regions of operation. The model parameters can then be grouped into subsets that most directly affect the device characteristics in each of the selected regions of operation. The pertinent regions of operation and parameter groupings may be selected automatically for the supported models. However, the user has the flexibility to change these if desired.

In the second step, initial estimates for the parameter values are calculated directly from the measured data. This step ensures that the final step will produce a physically meaningful set of parameters in the shortest time. The initial estimates can be calculated automatically for the supported models. However, the user has the flexibility to select the initial values as desired.

Finally, the TECAP2 optimizer is called to determine the parameter values that minimize the difference between the measured and simulated device char-

acteristics. The TECAP2 optimizer uses a nonlinear least squares fit algorithm (Levenberg-Marquardt algorithm [3]) in conjunction with user specifiable constraints for each parameter.

The models supported by TECAP2 are shown in Figure 3. Although these

Model Name	Description
HP-MOS	HP internal MOSFET model in HPSPICE [3]
UCB-MOS (UCB1, 2, 3)	3 levels of MOS model in Berkeley's SPICE [4]
CSIM	New analytical/empirical model in SPICE [5]
CLASSICAL-MOS	A simple first order MOSFET model
HPSPICE-BJT	Gummel-Poon Bipolar model in HPSPICE/SPICE
HPSPICE-DIODE	The Diode model in HPSPICE/Berkeley's SPICE
PN-CAP	P-N junction capacitance model
MOS-CAP	MOSFET gate structure capacitance model

Figure 3. Available device models in TECAP2.

models include the most widely used models at present, new models will continue to be developed. Consequently, TECAP2 is designed to provide the user with the capability to implement new models and extraction modules.

## 2.4 User Enhancements

New user-defined commands which customize the system for special functions can be implemented via a user-linkable module. These commands will appear in a TECAP2 command menu and may be used with the control structure like any other TECAP2 command. User defined models are also implemented through this module. A new model, when implemented, will automatically have the same simulation capability as TECAP2 supported models. Parameter extraction using the optimizer and the interactive simulation is also immediately available for new models. A fully automated extraction capability can be easily developed by creating a custom extraction module for the new model.

## 3. TECAP2 APPLICATIONS

The TECAP2 system was used to extract parameter values for five different models, in order to compare model performance. N-channel test devices ranging in masked channel length and width from 10.4 microns to 1.4 microns were characterized. The channel length reduction, LD, (due to processing effects) was about 0.35 micron and the channel width reduction WD was 0.55 micron resulting in an effective minimum geometry device of 0.3 by 0.7 micron. The objective of the parameter extraction was to obtain a single set of parameter for each model that would give the best possible fit over a range of different device geometries. The pertinent parameters are listed by functional category in Figure

4. Note that each model may also support other optional parameters that were not used here.

Model	HP-MOS	UCB1	UCB2	UCB3	CSIM
Basic Parameters	UO VT0 NSUB	UO VT0 GAMMA PHI	UO VT0 NSUB	UO VT0 NSUB	UN VFB PHI K1 BETA {1}
Mobility Reduction	VNORM ETRA		UEXP	THETA	UO UO_B
Sub-threshold			NFS	NFS	
Channel Width Effects	WD VWFF WFF	WD {2}	WD {2} DELTA	WD {2} DELTA	WD {3}
Channel Length Effects	LD VDFF LFF	LD	LD XJ	LD XJ	LD {3}
External Resistances	RS RD	RS RD	RS RD	RS RD	
Velocity Saturation	ECRIT		VMAX	VMAX	U1 {4}
Channel Length Modulation	DESAT	LAMBDA	NEFF	ETTA KAPPA	K2 ETA {5}
Scale with L Scale with W Sub-threshold Velocity Sat	Good Good No Good	No No No No	Good Add WD Good OK	Good Add WD Good Good	Good Good No Good
Speed	88 mSec	17 mS	180 mS	71 mS	29 mS
RMS Error Max Error	2.5% 7.3%	8% 17%	2.0% 6.2%	5.2% 6.8%	1.8% 5.1%

{1} Represented by five parameters.

{2} Added to the TECP2 implementation.

{3} 34 parameters model width/length effects for CSIM.

{4} Represented by 3 parameters.

{5} Represented by 3 parameters.

Figure 4. Functional grouping of model parameters.

### 3.1 Parameter Extraction Process

The model parameters were extracted using a total of four measurements (six for CSIM) on three different size devices. Each measurement typically consists

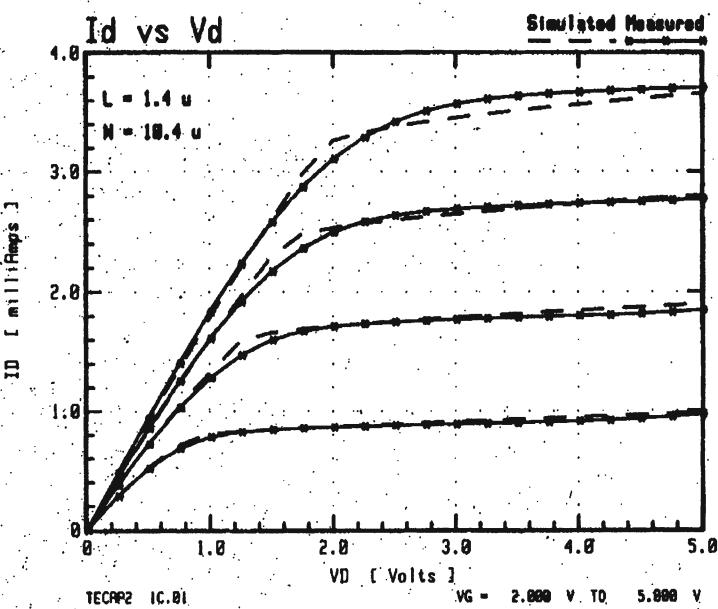


Figure 5. Comparison of the measured and simulated data after extraction of all parameters.

of 20 to 100 data points. An average measurement and extraction of a complete set of model parameters takes less than five minutes.

The "Basic" parameters and the "Mobility Reduction" parameters were first extracted using Id versus Vg data measured in the linear region on a large device (10.4/5.4). The "Sub-threshold" parameter (if any) was then extracted from the low current region of the same measurement. The "Channel width" and "Channel length" parameters were extracted using Id versus Vg data on narrow and short devices respectively. The "Velocity Saturation" and "Channel Length Modulation" parameters were then extracted using an Id versus Vd measurement on the shortest channel length device. Figure 5 shows the Measured and simulated data after final extraction of the HP- MOS model parameters.

The CSIM model requires a slight modification of the above extraction procedure, since scaling of device characteristics with geometry is achieved empirically. For each model parameter shown in Figure 4, two additional parameters are used to describe the variation of parameter value with geometry. These geometry dependent parameters are extracted from Id/Vg and Id/Vd measurements for three different geometry devices.

### 3.2 Geometry scaling

The extraction process described here extracts a complete set of model parameters from three different size devices. These parameters are then used to

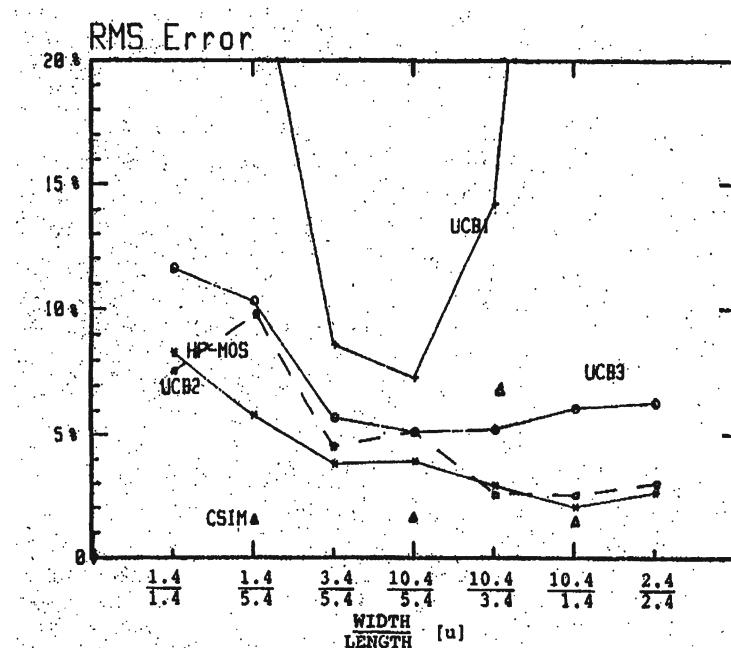


Figure 6. RMS error between measured and simulated data on different geometries.

simulate other geometries. The simulation results can then be compared to the measured data in order to study how each model scales with geometry. Figure 6 shows the RMS error between the simulated and measured data on different channel lengths and widths. The HP-MOS, UCB2 and UCB3 model show a reasonable accuracy in all geometries except when the channel width is 1.4 micron (effective width of 0.3 micron). The UCB1 model does not perform well, even for a large device. The CSIM model provides excellent accuracy near the geometries used to extract the parameter values. Larger deviations between measured and simulated results are encountered, however, for different geometries. This suggests that some parameters do not scale well using a  $1/L$  and  $1/W$  geometry dependence.

#### 4. CONCLUSIONS

A thorough comparison of five different models shows that the HP-MOS, UCB2, UCB3 and CSIM models scale down reasonably well to less than one micron. More fundamental work however is needed to push these models to less than half a micron. Implementation of a new model (CSIM) was accomplished in less than a week, demonstrating the expandability and flexibility of the TECAP2 system. All of the measurements, extractions, and comparisons

presented above were performed in a few hours, showing the effectiveness of TECAP2 as a modeling and characterization workstation.

## Acknowledgments

The authors wish to thank R. Dowell, Y. Yuan, M. Law, I. Klein, B. Loh, A. Tamer and R. Dutton for their help in developing the TECAP2 system. Also the authors wish to thank Kit Cham for providing the test wafers.

## References

- [1] Khalily, E. (1979). "TECAP An Automated Characterization System;" SEL 79-004, Stanford University.
- [2] Yuan, Y. et al. (1983). "Modular SPICE: A Modular Circuit Simulation Program," ICCAD.
- [3] More, J. (1977). "The Levenberg-Marquardt Algorithm: Implementation and Theory;" Numerical Analysis: Seventh Biannual Conference at the University of Dundee, Scotland, Springer Verlog, N.Y.
- [4] "HPSPICE MOSFET MODEL," DA35OD7, HP Internal Report, Engineering Productivity Division, April 1982.
- [5] Vladimirescu, A., and S. Liu, (1980). "The Simulation of MOS Circuits Using SPICE2," UCB/ERL M80/7, University of California at Berkeley,
- [6] Sheu, B. J. et al. (1984). "Compact Short-channel IGFET Model (CSIM)," UCB/ERL M84/20, University of California at Berkeley.

# TILOS: A POSYNOMIAL PROGRAMMING APPROACH TO TRANSISTOR SIZING

J. P. Fishburn<sup>1</sup> and A. E. Dunlop<sup>2</sup>

*AT&T Bell Laboratories*

*Murray Hill, New Jersey 07974*

## Abstract

A new transistor sizing algorithm, which couples synchronous timing analysis with convex optimization techniques, is presented. Let  $A$  be the sum of transistor sizes,  $T$  the longest delay through the circuit, and  $K$  a positive constant. Using a distributed RC model, each of the following three programs is shown to be convex: 1) Minimize  $A$  subject to  $T < K$ . 2) Minimize  $T$  subject to  $A < K$ . 3) Minimize  $AT^K$ . The convex equations describing  $T$  are a particular class of functions called posynomials. Convex programs have many pleasant properties, and chief among these is the fact that *any point found to be locally optimal is certain to be globally optimal*. TILOS (TImed LOgic Synthesizer) is a program that sizes transistors in CMOS circuits. Preliminary results of TILOS's transistor sizing algorithm are presented.

## 1. Introduction

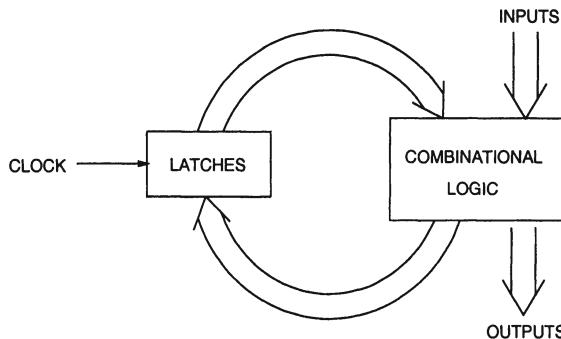
Given a synchronous MOS circuit of the form shown in Figure 1 with  $N$  transistors of sizes (channel widths)  $x_1, x_2, \dots, x_N$ , the following question is considered: How can the circuit's performance be improved by adjusting the  $x_i$ ? Two figures of merit are of special interest.  $T$  is defined to be the minimum clock period at which the circuit will operate. The other quantity,  $A$ , is simply the sum of transistor sizes.  $A$  is positively correlated with a number of other attributes of the circuit that should be minimized or constrained: These include silicon area, capacitance-discharge power, short-circuit power, and probability of a device failure within a chip.

TILOS (pronounced tee-los) is a program that requires a transistor connectivity file and I/O-delay file, and adjusts transistor sizes and connectivity within logical gates to meet the user's requirements for  $A$  and/or  $T$ . TILOS's output is a transistor connectivity file that can be passed to a layout program such as SC2 [5].

TILOS contains a static timing analyzer which recognizes latches and thus is capable of extracting all relevant timing paths from a circuit of the form shown

---

Authors are currently with <sup>1</sup>Agere Systems and <sup>2</sup>Crossbow Consulting.



*Figure 1.* Memory/combinational-logic model of digital MOS circuits. A path begins at the output of a latch or an input, and ends at the input of a latch or an output.

in Figure 1. This recognition process is similar to that used in other static timing analyzers [1] [10] [6], and will not be further described here.

Several authors ([13] [3] [4] [7] [9]) have reported on optimization techniques for transistor sizing. Contributions of this work over previous approaches include: 1) Using a distributed RC model of delay,  $T$  is proved to be a convex function of the transistor sizes.  $T$  remains convex if wire widths are also considered to be variables. 2) TILOS couples static timing analysis with transistor sizing, relieving the user of the need to specify which paths are to be optimized. Rather, the user specifies desired behavior of I/O signals, including clocks, and TILOS determines what paths are in need of improvement. 3) Transistors in latches as well as combinational gates are sized. 4) TILOS sorts series-connected subnetworks in a complex gate so that the subnetworks with earlier-arriving inputs are closer to the power supply. This heuristic allows transistor sizing a chance to operate: A transistor with an earlier-arriving input can be made larger and still have time to turn on, despite the increased gate capacitance, before other inputs arrive, providing a lower-resistance path to power. In addition, the increased source & drain capacitance of the larger transistor helps, rather than hinders, the output transition.

## 2. Three Formulations of the Transistor Sizing Problem

Let  $K$  be a positive constant, and let  $A$  and  $T$  be as defined above. Consider the following three optimization programs:

- 1 Minimize  $A$  subject to the constraint  $T < K$ .
- 2 Minimize  $T$  subject to the constraint  $A < K$ .
- 3 Minimize  $AT^K$ .

The first formulation can be used when the circuit must fit inside a system with a given clock period K. The second formulation might be used when the circuit is to be made as fast as possible, subject to constraints on silicon area, power, or yield. The third formulation represents a wholistic approach in which both A and T are important, but have relative weights assigned to them.

### 3. MOSFET Model

The MOSFET model that TILOS uses is shown in Figure 2. The gate, source, and drain capacitances are all proportional to the transistor size X, and the source-to-drain resistance is inversely proportional to X.

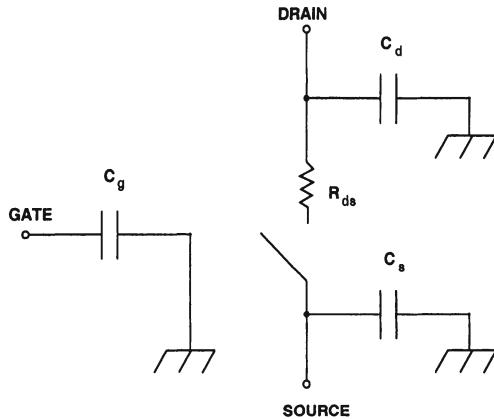


Figure 2. The source, drain, and gate capacitances are proportional to transistor size. The effective resistance is inversely proportional to transistor size.

### 4. RC Delay Model

Figure 3 illustrates the modeling of gate delay by a distributed RC network [11] [8]. In the RC network shown, an upper bound for the discharge time is

$$(R_1 + R_2)C_2 + (R_1 + R_2 + R_3)C_3. \quad (1)$$

This represents a much tighter bound than a lumped R and C model.

With the MOSFET and RC-delay models, the delay through any series configuration of transistors can be expressed in terms of the transistor sizes and routing parasitics. The important point here is the form of (1) when expressed as a function of the transistor sizes  $x_i$ : Each  $R_i$  is proportional to  $1/x_i$ , and each  $C_i$  is some constant (for wire capacitance) plus one term for each transistor whose gate, drain or source is connected to the node. This term is proportional to the

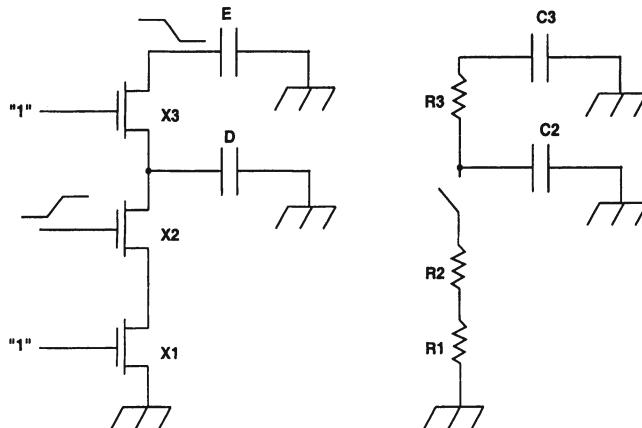


Figure 3. Delay through pulldown network of NAND gate is modeled with RC network.

transistor size. Thus (1) can be rewritten as:

$$(A/x_1 + A/x_2)(Bx_2 + Cx_3 + D) + (A/x_1 + A/x_2 + A/x_3)(Bx_3 + E) \quad (2)$$

where A, B, and C are constant coefficients for resistance, drain and source capacitance, respectively, and D and E are wire capacitances. It is interesting to note that if wire widths, as well as transistor widths, are treated as variables, then the expression for delay remains in the same form as (2).

## 5. Delay Through Complex Gate

For each transistor in a pulldown or pullup network of a complex gate, the greatest resistance path from the drain to the gate output is computed, as well as the greatest resistance path from the source to a supply rail. Thus for each transistor, the network is transformed into an equivalent series configuration, and the calculation of the previous section can be applied.

## 6. Delay Through a Single Circuit Path

Every circuit path has two path delays: one for the case where the input to the path is rising, the other for the falling input. Since a path delay is simply a sum of gate delays as in (2), the general form of a path delay is as follows:

$$\sum_{i,j=1}^N a_{ij} \frac{x_i}{x_j} + \sum_{i=1}^N \frac{b_i}{x_i}, \quad (3)$$

where the  $a_{ij}$  and  $b_i$  are nonnegative constants that are mostly zero. The function which (3) describes is *convex*, which means that any straight line segment in

$N+1$ -dimensional space whose endpoints lie in the graph of the function is itself entirely on or above the graph.

## 7. Delay Through Entire Circuit Is a Convex Function of the $x_i$

The delay  $T$  through a single combinational block is defined as the maximum, over all possible paths through the block, of expressions of the form (3). Since convexity is preserved under sums and maxima,  $T$  is thus a convex function of the variables  $x_i$ . As a consequence, all three formulations considered in section 2 are the minimization of a convex function subject to an upper bound constraint on another convex function. This implies that any point found to locally minimize the objective function also globally minimizes it. The field of *Convex Programming* [12] has been intensively explored over the last several decades, and many techniques are available. In addition, (3) belongs to a special class of functions called *posynomials*, and the more specialized techniques of *Geometric Programming* [2] can be used. Techniques such as simulated annealing or multiple random starts are unnecessary.

## 8. Tailoring Convex Programming Techniques to Transistor Sizing

Since we are interested in applying the sizing algorithm to circuits as large as an entire VLSI chip or perhaps to an entire system, the algorithm must be as efficient as possible, perhaps at the expense of absolute accuracy in finding the optimum. In our experience, the algorithm described below provides an efficient method for converging to the optimum point.

TILOS proceeds as follows: Starting with minimum sizes for all transistors a static timing analysis is performed on the circuit, which assigns two numbers to each electrical node:  $t_l$  (latest time to go low), and  $t_h$  (latest time to go high). From each path output that fails to meet its timing goals for  $t_l$  or  $t_h$ , TILOS walks backward along the failing path. Whenever a node X is visited, TILOS examines in turn each NFET (if X's  $t_l$  is failing) or PFET (if X's  $t_h$  is failing) which could have an affect on the path. In general, this includes both the *critical transistor* (the transistor whose input is on the critical path), and *supporting transistors* (transistors along the highest-resistance-to-power-supply path from the source of the critical transistor). TILOS calculates the *sensitivity* of each such transistor  $i$ , which is the time savings accruing per increment of  $x_i$ . The size of the transistor with the largest sensitivity is increased, and the process is repeated.

Figure 4 illustrates a series configuration in which the critical path extends back along the gate of the top (critical) transistor. The sensitivity calculation for this transistor is derived as follows: Fix all transistor sizes except  $x$ , the size of

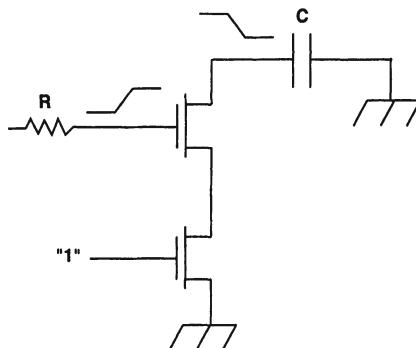


Figure 4. The critical path extends back along the gate of the top transistor.

the critical transistor.  $R$  is the total resistance of an RC chain driving the gate.  $C$  is the total capacitance of an RC chain being driven by the configuration. Then the total delay  $D(x)$  of the critical path is

$$D(x) = K + RC_u x + \frac{R_u C}{x},$$

where  $R_u$  and  $C_u$  are resistance and capacitance of a unit-sized FET.  $K$  is a constant that depends on the resistance of the bottom transistor, capacitance RC chain, and resistance in the driven RC chain. The sensitivity  $D'(x)$  is then

$$D'(x) = RC_u - \frac{R_u C}{x^2}.$$

The sensitivity calculation for supporting transistors is done in a similar way. When  $D'(x)$  is set equal to zero, the resulting value of  $x$ , which minimizes delay, is equal to a constant times

$$\sqrt{C/R}. \quad (4)$$

An interesting consequence of (4) occurs in the special case when all inputs are equally critical. Since the quantity  $C$  includes capacitance of all sources and drains of FETS higher in the series configuration, the transistor sizes that produce minimal delay are smaller near the output, and larger near the power supply. This is similar to the “pyramid shaped” nand gates of Shoji [14].

The sizing process terminates when either the constraints are met, or when the circuit has passed its absolute minimum and is getting slower instead of faster. Since the number of paths through a circuit can be very large in comparison to the size of the circuit itself, the optimization is performed without ever actually keeping track of a sensitivity (Lagrange Multiplier) for each critical path, or even enumerating all paths.

## 9. Some Preliminary Results

TILOS currently consists of 2705 lines of C, Lex, & Yacc source code. Design started in January, and coding in March of 1985. Recently sorting of series subnetworks, recognition of latches, and sizing of transistors within latches were added. The following table summarizes TILOS's performance on 4 sample circuits. The first 3 are 2, 8, and 32-bit adders, and the fourth is a standard-cell finite-state machine with 2 static flip-flops. The table gives the number of transistors, T (ns) and A (microns) before and after sizing, and the number of seconds of TILOS execution on a 68000-based workstation.

Speedup Due to Transistor Sizing						
Name	FETs	Unsized		Sized		Ex. sec.
		T ns	A mic.	T ns	A mic.	
add2	56	14	210	8	278	6
add8	224	42	840	18	986	49
add32	896	154	3360	58	3609	1424
fsm	180	30	675	8	744	97

## 10. Directions for Future Research

The major source of error in the distributed RC timing model is the lack of consideration for slowly rising inputs. Unfortunately, it has not been possible to prove, as it was for the distributed RC model, that delay through the circuit is a convex function of the transistor sizes when the input waveform shape is taken into account. Although there are several static timing analyzers and a transistor sizer [9] that take into account input waveform shape, we hesitate to do so without a convexity proof in hand. If a more accurate model turns out to be non-convex, there is always the danger that the optimizer might become trapped in a local minimum that is not a global minimum, resulting in a more pessimistic solution than the less accurate model.

## 11. Acknowledgments

The authors would like to thank T. G. Szymanski, D. M. Gay, and E. Rosenberg for many helpful discussions.

## References

- [1] V. Agrawal. Synchronous path analysis in MOS circuit simulator. In *Design Automation Conf.*, pages 629–635, 1982.
- [2] J. G. Ecker. Geometric programming: methods, computations and applications. *SIAM Review*, 22(3):338–362, July 1980.
- [3] L. A. Glasser and L. P. J. Hoyte. Delay and power optimization in VLSI circuits. In *Design Automation Conf.*, pages 529–535, 1984.

- [4] K. S. Hedlund. Electrical optimization of PLAs. In *Design Automation Conf.*, pages 681–687, 1985.
- [5] D. D. Hill. SC2: A hybrid automatic layout system. In *Int. Conf. on Computer Aided Design*, pages 172–174, 1985.
- [6] N. Jouppi. Timing analysis for nMOS VLSI. In *Design Automation Conf.*, pages 411–418, 1983.
- [7] C. M. Lee and H. Soukup. An algorithm for CMOS timing and area optimization. *IEEE J. of Solid-State Circuits*, 19:781–787, October 1984.
- [8] T. M. Lin and C. Mead. Signal delay in general RC networks with application to timing simulation of digital integrated circuits. In *Conf. on Advanced Research in VLSI*, pages 93–99, 1984.
- [9] M. Matson. Optimization of digital MOS VLSI circuits. In *Proc. Chapel Hill Conf. on VLSI*, pages 488–491, 1985.
- [10] J. Ousterhout. Switch-level delay models for digital MOS VLSI. In *Design Automation Conf.*, pages 542–548, 1984.
- [11] P. Penfield and J. Rubinstein. Signal delay in RC tree networks. In *Proc. 2nd Caltech VLSI Conference*, pages 269–283, 1981.
- [12] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
- [13] A. E. Ruehli, P. K. Wolff, and G. Goertzel. Analytical power/timing optimization technique for digital system. In *Design Automation Conf.*, pages 142–146, 1977.
- [14] M. Shoji. Electrical design of BELLMAC-32A microprocessor. In *Int. Conf. on Circuits and Systems*, pages 112–115, 1982.

# SPECS2: AN INTEGRATED CIRCUIT TIMING SIMULATOR\*

Chandu Visweswariah<sup>1</sup> and Ronald A. Rohrer<sup>2</sup>

*Carnegie-Mellon University  
Pittsburgh, PA 15213*

## Abstract

SPECS2 is a prototype implementation of a new timing simulation and modeling methodology. SPECS2 (Simulation Program for Electronic Circuits and Systems 2) is a tree/link based, event-driven, timing simulator. A modeling technique, which is predicated on the conservation of charge and energy, is employed to produce table models for device evaluation. The tables may be constructed to model devices at any desired level of detail. Thus, SPECS2 is a variable accuracy simulator. Grossly differing accuracy requirements may be specified for different runs and also mixed over different parts of the same circuit. SPECS2 implements a novel oscillation detection and suppression scheme that prevents algorithmic oscillation, while leaving real circuit results undistorted. SPECS2 takes advantage of the tree/link formulation of the circuit equations to provide a formal and general approach to timing simulation. It encounters no special problems with floating capacitors or transmission gates. Further, SPECS2 provides the framework for a generalized macromodeling and simulation capability.

## 1. Introduction

Timing simulators [1, 2, 3, 4, 5, 6] seek to sacrifice accuracy for efficiency in the simulation of large digital integrated circuits. The following observations can be made about existing timing and circuit simulators.

- The cost-accuracy tradeoff is not continuous and often not in complete control of the user. Most simulators are either quick and dirty or excruciatingly slow and very accurate. There is no simulator that accommodates both simple models for first cut simulation as well as more complex models for further refinement.
- To guarantee reliability (convergence), most simulators require unnecessarily complicated models, often resulting in poor efficiency. Simplistic models often are not allowed. There is usually no flexibility in the variation of modeling complexity.

---

\*This work was supported by the CMU CAD Center Industrial Affiliates and the Semiconductor Research Corporation.

<sup>1,2</sup>Authors are currently with <sup>1</sup>IBM Thomas J. Watson Research Center, Yorktown Heights, NY and <sup>2</sup>Magma Design Automation, Cupertino, CA, and <sup>2</sup>Neolinear, Pittsburgh, PA.

- Many modern simulators are specialized to handle only MOS circuitry and lack the capability to handle circuits of a more general nature.

With a view to overcoming some of the above limitations, a prototype timing simulator called SPECS2 (Simulation Program for Electronic Circuits and Systems 2) has been developed. It is an event-driven, tree/link based, timing simulator that uses table models for device evaluation. The accuracy of the modeling is user-specified and may be varied over different parts of the same circuit and over multiple runs. SPECS2 handles a wide range of models from very simple to very complex. The simulation strategy is not tied to the nature of MOS digital circuitry and may be used for bipolar and general analog circuitry as well. SPECS2 uses a novel oscillation prevention method for circumventing spurious algorithmic oscillation. Variable accuracy modeling is allowed and is based on the conservation of charge and energy.

## 2. Overview of modeling and simulation

### 2.1 Modeling

Most simulators tacitly assume that the polynomial order of the variation of currents and voltages between computed time points is the same. However, for a linear capacitor, the variation of the current must be one polynomial order lower than that of the voltage in order to conserve charge. Bearing this fact in mind, the modeling in SPECS2 is based on the following assumptions.

- All branch voltages in the circuit are approximated to be piecewise linear in time. All branch currents are piecewise constant in time.
- Kirchhoff's current and voltage laws are satisfied at all times, at the extremities and in the interior of all piecewise segments.
- The single table (constant) current  $I$  of a two-terminal device represented by  $i = f(v)$  in the voltage range  $v_i$  to  $v_f$  is given by the formula

$$I = \frac{1}{(v_f - v_i)} \int_{v_i}^{v_f} f(v) dv. \quad (1)$$

The following conclusions may be drawn from the above assumptions.

- Since all branch voltages are piecewise linear and branch currents piecewise constant in time, charge is always conserved in every linear capacitor.
- Since all currents and all voltages in the circuit are of the same polynomial order of variation, respectively, and since KCL and KVL are guaranteed to hold always, Tellegen's theorem guarantees the conservation of energy. These two assumptions are the same as those made in [7].

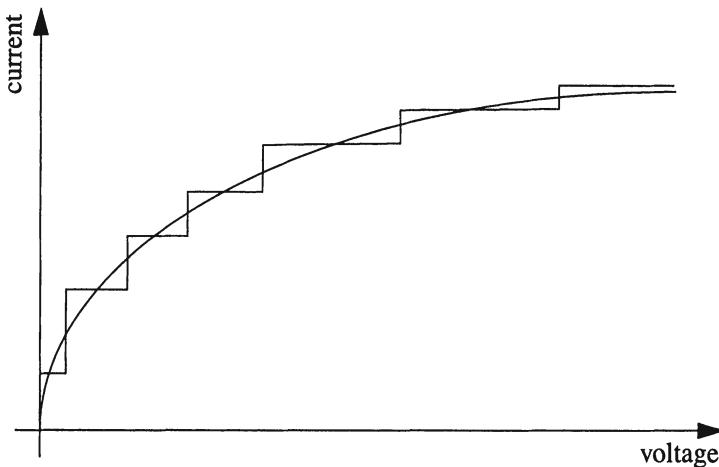


Figure 1. i-v characteristics are represented by a stair step function.

3 For a linear resistor, the branch constitutive relation, Ohm's Law in this case, is violated by the above assumptions. Hence, the device characteristics are forced to be represented by a stair step function in the i-v plane, as shown in Fig. 1. This approximation leads to a timing error incurred in the process of simulation. However, if the resistor is modeled using the formula of (1), the timing error is only second order [8, 9].

## 2.2 Simulation

The timing simulation in SPECS2 is based on a tree/link formulation of the circuit equations. The tree consists only of capacitors and independent voltage sources. The remaining branches form the links and are represented by stepwise models in the i-v plane (multi-terminal elements are represented by  $n$ -dimensional stepwise table models). The existence of such a tree is assumed. However, this is not a limiting assumption.

An “event” is said to occur whenever a (possibly nonlinear) element reaches a corner of its stair step model. Then, a new branch current is found for the element by look-up in its associated table. The current distributes over tree branches as dictated by KCL. Constant capacitor branch voltage slopes are summed algebraically around fundamental loops to predict the next event times of the links in their loops. The tree/link partition determines *a priori* the topological extent of the propagation of an event. Any two links that share a capacitor

in their fundamental loops are said to be in each other's "sphere of influence." Whenever one of them has an event, the other is affected. The effect manifests itself as the rescheduling of events associated with all the links in the "sphere of influence." The entire simulation algorithm consists of this event processing until the simulation interval elapses.

### 3. Oscillation prevention

Discretization in device models causes the device current to be approximated by a value that is almost always a little too high or too low. Near equilibrium, the computed value hovers back and forth around the correct steady state value. The result is a limit cycle, causing spurious algorithmic oscillation. While the irregular waveforms produced may be acceptable, the loss in efficiency due to the repeated and unnecessary scheduling and processing of events is not. Many timing simulators, such as SPECS [4], MOTIS [3] and E-LOGIC [5] face a similar problem. SPECS2 uses a novel and effective method for preventing spurious oscillation. The advantages of this method are that no *a priori* knowledge of the steady state is required (unlike in [3]), real oscillation is not suppressed (which is an obvious disadvantage of a cycle detector) and convergence is guaranteed irrespective of the model coarseness. The oscillation prevention scheme never allows a device to cross its estimated steady-state (or "pseudo steady-state") value. Whenever possible, a device is pushed into a "pseudo steady-state" and it leaves the event queue. Subsequent scheduling of topologically related events could, of course, cause this element to reenter the active event queue.

This oscillation prevention algorithm has been generalized to multi-terminal elements. Thus, a formal and general scheme has been developed to tackle the problem of spurious oscillation. This scheme is one of the important features of SPECS2.

### 4. Computational efficiency

Many means of improving computational efficiency have been implemented in SPECS2.

- **Sparsity:** The cutset and loop linkages are stored as sparse matrices to exploit spatial sparsity.
- **Fast incremental KCL:** When an event occurs, the changes in the currents of the corresponding tree branches may be computed without repeating KCL through the cutset.
- **Fast incremental KVL:** The new  $dv/dt$  around a loop when an event occurs may be found in terms of the old  $dv/dt$ , the change in the link current and the loop matrix entry, rather than repeating KVL on a member-by-member basis.

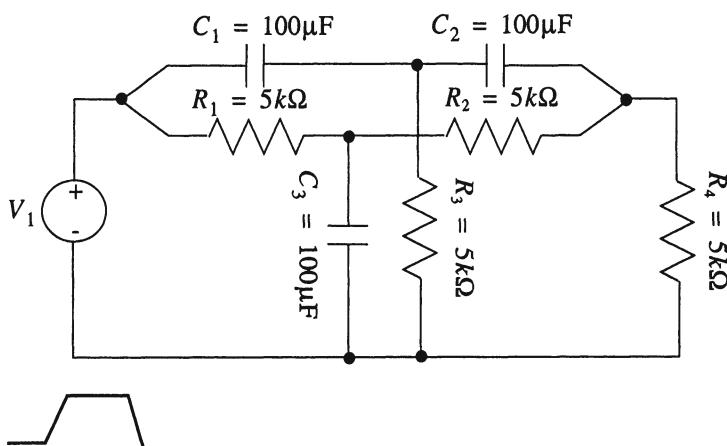


Figure 2. "Twin-T" RC circuit.

- **The event heap:** The inserting, scheduling, rescheduling and deleting of events could be a high overhead rider on the SPECS2 algorithm. To decrease this overhead, the events are stored in a "priority queue" or "heap" [10], that makes the worst case operation time  $O(\log_2 n)$  in the number of events, with the complexity typically being more favorable.

## 5. Results of simulation

SPECS2 runs on a VAX<sup>1</sup> 11/780 or station II running 4.3 BSD UNIX<sup>2</sup>. It has been run on numerous circuits containing resistors, diodes and MOSFETS. The results of the simulation of some circuits are presented in this section. To demonstrate accuracy, the "Twin-T" RC circuit of Fig. 2 (with floating capacitors) has been selected. The results of simulation with 100 mV segments in the resistor models are shown in Fig. 3. SPICE waveforms are in dotted lines for comparison. Fig. 4 shows the delay of a signal through a set of 6 inverters, with SPICE for comparison on the same circuit. Fig. 5 shows an XNOR gate and the results of simulation are shown in Fig. 6. For these small circuits with 5 to 50 segments on each of the device model axes, SPECS2 is about an order of magnitude faster than SPICE. When very accurate models with more than 75 segments on each axis are employed, the run times tend to become comparable to those of SPICE. SPECS2 is still experimental and has not been optimized for speed.

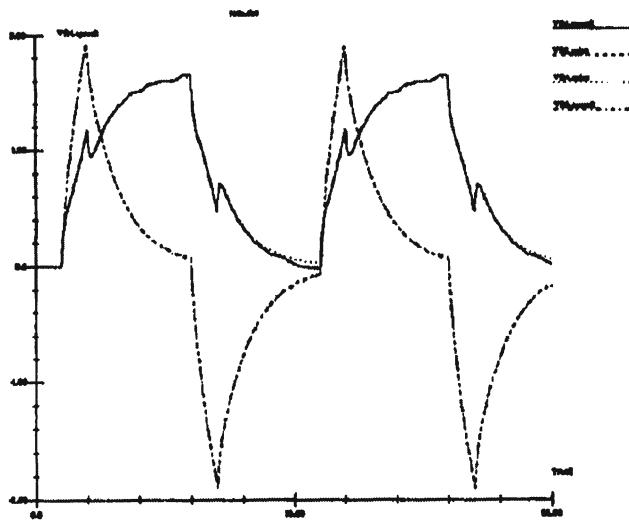


Figure 3. Simulation of "Twin-T" RC circuit.

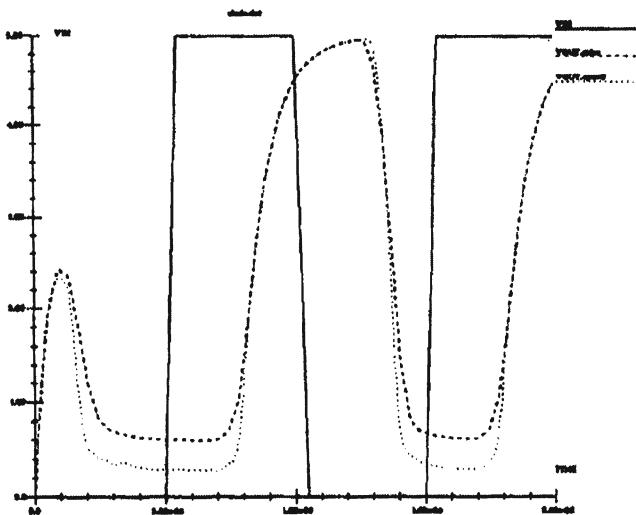


Figure 4. Delay of signal  $vin$  through 6 inverters.

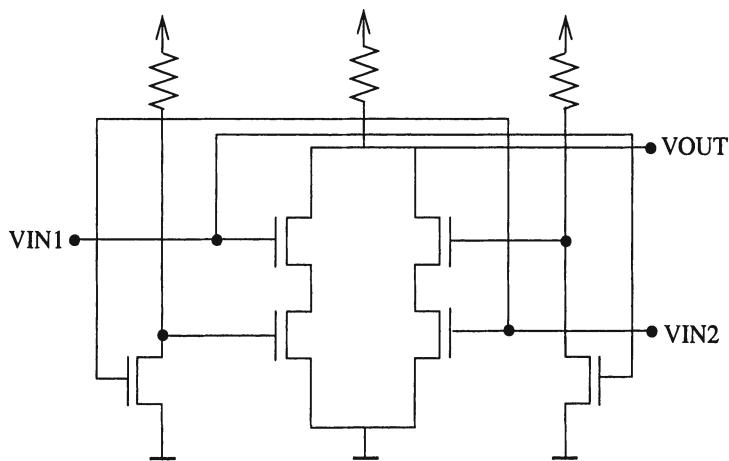


Figure 5. XNOR gate.

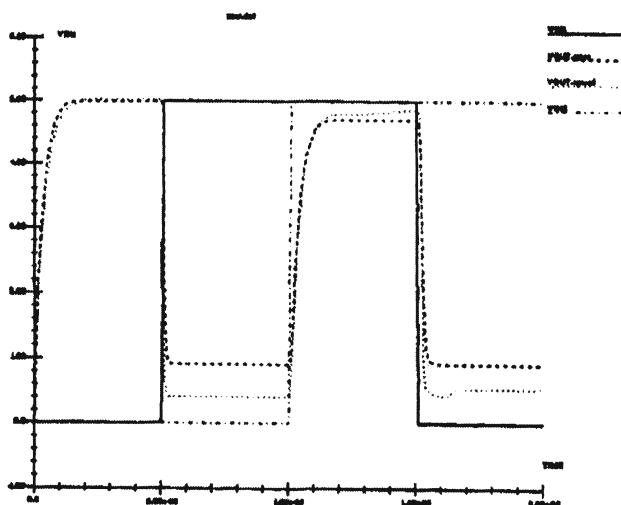


Figure 6. Simulation of XNOR gate.

## 6. Conclusions

A new timing simulator has been presented that is event-driven, uses table models and is based on tree/link circuit equation formulation. Models are built based on the conservation of charge and energy. SPECS2 is a variable accuracy simulator. A novel oscillation prevention method has been implemented that weeds out only spurious oscillation and is guaranteed to provide convergence. A circuit theoretic basis is used in the event processing and oscillation prevention algorithms. Simulation results, even with very coarse device models, are encouraging. Some of the ongoing and future projects involving SPECS2 are described in the next paragraph.

The incorporation of nonlinear capacitors and a larger repertoire of devices such as bipolar transistors is under investigation. Efficient handling of systems with large variations in time constants must be worked out. "Illegal" loops and cutsets must be handled. It is possible to have a hierarchy of device models at different accuracy levels. An adaptive error control scheme would dynamically decide which model would be used, depending on how close the concerned link was to its steady state. A generalized macromodeling capability with user-defined models is another future goal of the SPECS2 simulator.

## 7. Acknowledgments

The authors would like to thank Mr. Kah-Kuen Low for his SPICE input parser, Mr. Rajeev Jayaraman for the sparse storage routines and Ms. Cynthia Della-Torre for building the table models. Fruitful discussions with Mr. Xiaoli Huang, Mr. Larry Pillage and Ms. Xueqing Zhang are acknowledged.

## Notes

1. VAX is a trademark of Digital Equipment Corporation
2. UNIX is a trademark of AT&T Bell Laboratories

## References

- [1] B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS - An MOS timing simulator," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, pp. 901-910, December 1975.
- [2] C.-F. Chen, C.-Y. Lo, H. N. Nham, and P. Subramaniam, "The second generation MOTIS mixed-mode simulator," *Proc. 1984 Design Automation Conference*, pp. 10-16, 1984.
- [3] D. Tsao and C.-F. Chen, "A "fast timing" simulator for digital MOS circuits," *IEEE International Conference on Computer-Aided Design*, pp. 185-187, November 1985.
- [4] A. J. de Geus, "SPECS: simulation program for electronic circuits and systems," *Proc. International Symposium on Circuits and Systems*, pp. 534-537, May 1984.
- [5] Y. H. Kim, J. E. Kleckner, R. A. Saleh, and A. R. Newton, "Electrical-logic simulation," *IEEE International Conference on Computer-Aided Design*, pp. 7-10, November 1984.

- [6] L. M. Vidigal, S. R. Nassif, and S. W. Director, "CINNAMON: coupled integration and nodal analysis of MOS networks," *Proc. 1986 Design Automation Conference*, pp. 179–185, June 1986.
- [7] G. Ruan and J. Vlach, "Current limited switch-level timing simulator for MOS logic networks," *Proc. IEEE International Conference on Computer Design*, pp. 597–601, 1985.
- [8] C. Visweswariah, "SPECS2: a timing simulator," Tech. Rep. CMUCAD-86-24, Carnegie Mellon University, October 1986.
- [9] C. DellaTorre, "Table look-up device modeling for timing simulation," Tech. Rep. CMUCAD-86-25, Carnegie Mellon University, October 1986.
- [10] R. Sedgewick, *Algorithms*. Addison-Wesley Publishing Company, 1983.

# AUTOMATIC SYNTHESIS OF OPERATIONAL AMPLIFIERS BASED ON ANALYTIC CIRCUIT MODELS

Han Young Koh, Carlo H. Séquin and Paul R. Gray

*Department of Electrical and Computer Engineering and Computer Sciences*

*University of California*

*Berkeley, CA*

## Abstract

An automatic synthesis tool for CMOS op amps (OPASYN) has been developed. The program starts from one of a number of op amp circuits and proceeds to optimize various device sizes and bias currents to meet a given set of design specifications. Because it uses analytic circuit models in its inner optimization loop, it can search efficiently through a large part of the possible solution space. The program has a SPICE interface that automatically performs circuit simulations for the candidate solutions to verify the results of the synthesis and optimization procedure. The simulation results are also used to fine-tune the analytic circuit descriptions in the database. OPASYN has been implemented in Franz Lisp and demonstrated for three different basic circuits with a conventional  $3\text{ }\mu\text{m}$  process and a more advanced  $1.5\text{ }\mu\text{m}$  process. Experiments have shown that OPASYN quickly produces practical designs which will meet reasonable design objectives.

## 1. Introduction

In recent years, rapid advances have been made in automating the design of analog and mixed analog/digital circuits [1, 2, 3, 4, 5, 6]. The general approach is to use building blocks which may be stored in the form of parameterized generators or as entries in macrocell or standard cell libraries. While the usage of libraries of predefined building blocks can shorten the design period, it cannot give an optimal design for every application. Furthermore, the library entries become obsolete each time the technology or the design rules are modified. Generators that are operating at the circuit or symbolic level are more flexible and can be useful over a much larger domain.

Among various building blocks used in analog systems, an operational amplifier (op amp) is one of the most widely used circuit components. An efficient design of optimal op amps is thus a corner-stone of a design environment for many applications. Several methods concerning the automated design of op amps have been published [1, 6, 7, 8, 9]. An optimization-based approach is based on algorithmic optimization and circuit analysis techniques. It is applicable to a broad range of analog circuits and produces near optimal solutions. However, this approach is costly in CPU time and has difficulty in properly tuning the system according to the designers' needs. Alternatively, an expert system

can be used to store human designer's knowledge. But the large search space resulting from the many degrees of freedom in the design of op amp circuits makes this approach difficult and inefficient.

This paper introduces a practical intermediate approach to automating op amp synthesis. It is based on analytic circuit models of the op amp circuit topologies covered by the synthesis system. It follows the general approach often taken by human designers, but automates the tedious and computationally extensive aspects of the design process. Based on the applications, a suitable circuit topology is selected, and its device sizes and bias currents are then adjusted in an iterative manner until the circuit optimizes some selected targets. When a reasonable design configuration has been found, it is subject to extensive circuit simulation to verify that indeed all the design specifications are met. This whole process has been automated.

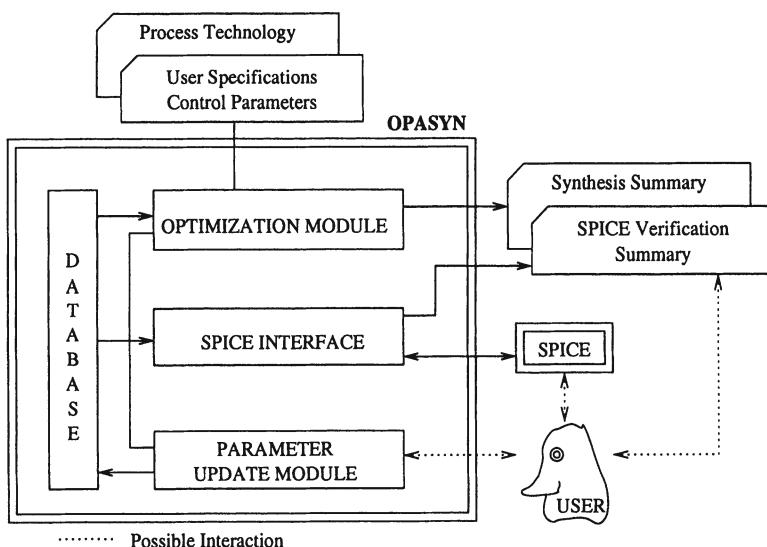


Figure 1. Organization of the OPASYN system.

As shown in Fig. 1 our op amp synthesis system (OPASYN) consists of a topology database, an optimization module, an interface to the circuit simulator, SPICE, and a parameter update module; the latter three are circuit and technology independent. The topology database stores analytic circuit performance models for each of the adopted circuit topologies so that the design parameters can be calculated without the aid of a circuit simulator. The optimization module improves the optimality of the design with an algorithmic search procedure. The SPICE interface automatically simulates the chosen circuit and determines its exact performance parameters, thus verifying the synthesis results. The SPICE simulation summary is also used by the parameter update module to improve the

analytic circuit models in the database. A modular system configuration makes it easy to update OPASYN's database as device technology changes or better circuit models become available.

## 2. Circuit Synthesis Based on Analytic Circuit Models

At the heart of OPASYN is an efficient circuit optimization module. The optimization process relies on analytic circuit models which contain analog circuit designers' expert knowledge about each op amp circuit topology and the dependencies between device sizes, bias currents, and performance characteristics of the overall circuit. Thus the inner loop of the optimization process can run much more efficiently using the explicit dependencies in these models rather than performing a circuit simulation after each optimization step. In addition, this analytic modeling of a given op amp circuit topology produces a simple and smooth solution space so that the optimization procedure can use faster - but less robust - search procedures. In OPASYN, a simple steepest-gradient descent method has been used to explore large portions of the solution space within a short period of time.

Each of the analytic circuit models contains a netlist for the corresponding circuit topology, a declaration of independent design parameters for the circuit, and a reasonable range of values for these parameters. On top of these basic properties, the model stores analytic expressions to compute circuit performances. These expressions were derived by using first-order circuit analysis techniques and topology-specific approximations [13]–[17]. For most dc characteristics these computed approximations are excellent. For highly non-linear specifications such as gain, phase margin, and settling time of the circuit, fitting parameters have been introduced to obtain more accurate predictions of specific performance characteristics. One example of such analytic expressions is

$$a_v = c f_{\text{gain}} \times \frac{g_m 2 g_m 6}{(g_o 2 + g_o 4)(g_o 6 + g_o 7)}$$

where  $a_v$  is the small signal dc gain of the circuit topology in Fig. 2,  $g_m$  is a transconductance,  $g_o$  is an output conductance of a transistor, and  $c f_{\text{gain}}$  is a fitting parameter. All the conductances are dependent on transistor sizes and bias currents. The fitting parameters are being updated as the system acquires more information from repeated synthesis and verification steps.

Based on these equations and the user-defined design targets, a cost function is computed which represents a relative figure-of-merit for any particular combination of design parameter values. In the present version, the cost function

(C) is formed as follows:

$$C = \sum_1^n \left( \exp \left( \pm \frac{w_i (\text{spec}_i - p_i)}{\text{spec}_i} \right) - 1 \right)$$

where  $n$  is the number of circuit performance parameters considered in the program,  $p_i$  is the  $i$ -th performance parameter,  $w_i$  is a relative design priority of  $p_i$  and  $\text{spec}_i$  is the corresponding design specification. The plus sign is chosen when meeting the specification requires the value of  $p_i$  to be greater than that of  $\text{spec}_i$ . The cost function produces a smooth search space where the gradient descent method works effectively. The exponential nature of the cost function prevents the penalty for violating any specification from being compensated by overly satisfying other specifications.

The search for an optimal solution starts with a coarse-grid sampling through the entire parameter space fine enough to yield a starting point in each cost function well. From the more promising sampling points found in this survey, a steepest-gradient descent algorithm searches for the optimal solution in this neighborhood. OPASYN will often return several solutions if many different locally optimal solutions are found that come close enough to the stated design targets.

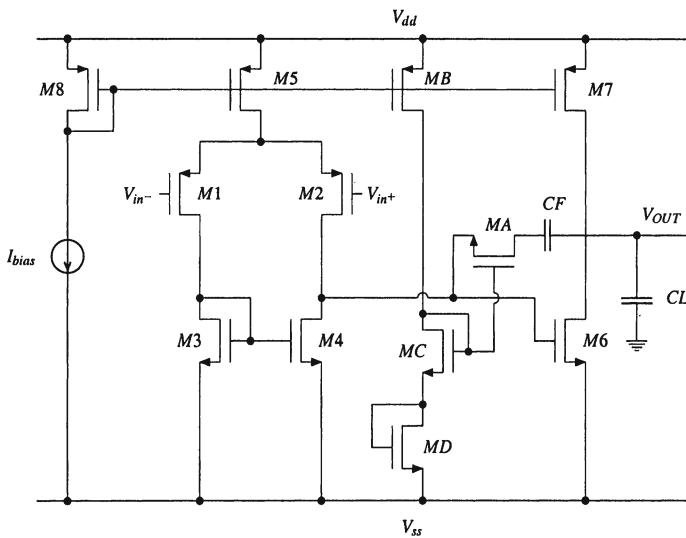


Figure 2. Basic Two Stage OP Amp.

The described optimization algorithm is independent of the specific device technology or circuit topology used. However, when a new circuit topology needs to be introduced into OPASYN, the corresponding analytic circuit models

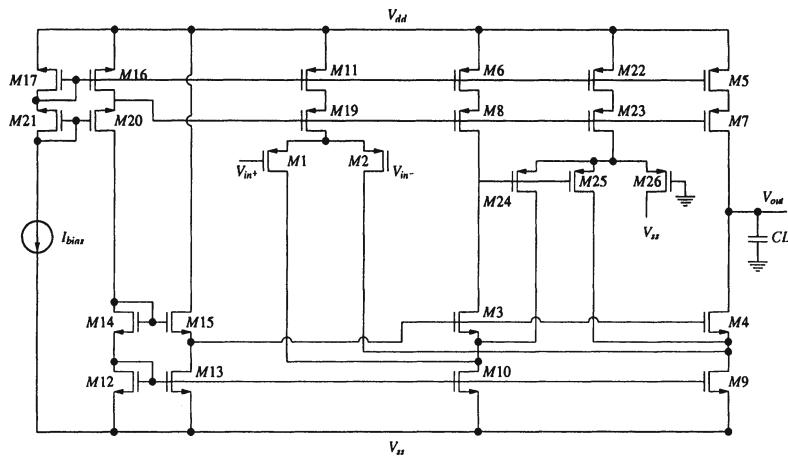


Figure 3. Single Stage Folded Cascode OP Amp.

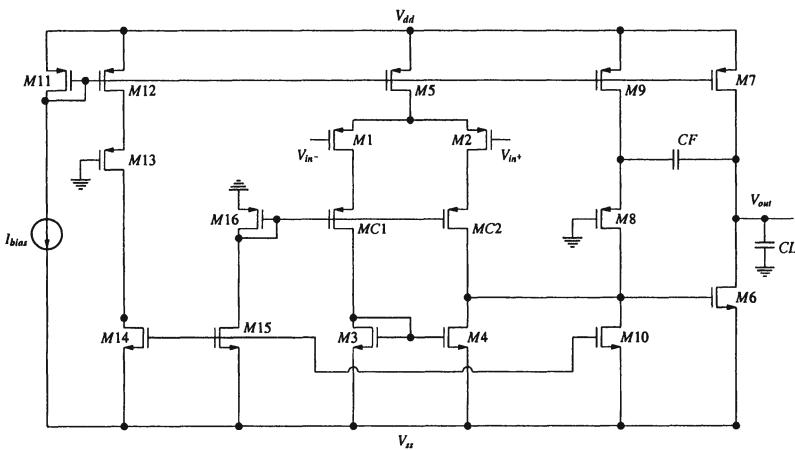


Figure 4. Two Stage OP Amp with Cascoded First Stage.

must first be created. This is a non-trivial task that demands the attention of a good analog circuit designer.

In addition to creating the proper expressions that describe the functional dependencies in a new circuit model, reasonable values for the various fitting parameters must be determined; this requires a substantial number of simulation runs. Any discrepancy between the predictions and the results of the circuit simulations can be used to fine-tune these fitting parameters and to improve the accuracy of the predictions. Thus the more the system is used, the more accurate it gets because of the ‘experience’ gained from previous design tasks.

### 3. Automatic Verification of Synthesis Results

After the device parameters have been determined in the optimization phase, the resulting circuit(s) need to be carefully checked against the given design specifications. This is done with a series of SPICE simulation runs to accurately determine the various performance characteristics. At this stage it is also possible to study the influence of processing variations and device tolerances.

OPASYN's SPICE interface module generates the needed SPICE input files, makes the necessary system calls to execute the various simulations, interprets the SPICE output files to determine the various performance characteristics, and compares the latter against the given design targets. OPASYN then supplies a summary of these simulation results to the user. It also utilizes these results to improve the fitting parameters in the analytic models in the database.

One of the difficulties with SPICE simulation of analog MOS circuits is to achieve dc convergence. Designers often spend a considerable amount of time trying to find the right initial conditions and control parameters to be able to execute a particular simulation successfully. To alleviate this problem and to automate the verification process, the simulation runs are started with suitable initial node voltages. These initial node voltages are obtained from a SPICE simulation of a slightly modified circuit topology which has much better dc convergence properties. We have also investigated the effects of various control parameters in SPICE on dc convergence and used the most promising combination of these parameter values. This method was successful in most of the design examples we have tried. In case OPASYN fails to successfully complete the SPICE verification, it generates the necessary SPICE input files and let the user complete the verification work and run the parameter update module (see Fig. 1).

### 4. Implementation and Results

All of the OPASYN modules shown in Fig. 1 have been implemented in Franz Lisp and are running under an Ultrix X2.0-3A system on a VAX 8800. The topology database contains the analytic circuit models for three of the most frequently used op amp circuit topologies [14, 15] shown in Figures 2-4. It also contains expert analog designers' conventional design rules in the form of various relations that must hold between certain circuit parameters.

The OPASYN technology database currently contains the SPICE device parameters for a conventional MOSIS 3  $\mu\text{m}$  process and the more advanced GE 1.5  $\mu\text{m}$  process. As demonstrated with Tables 1 to 3, OPASYN has found good design configurations for the three examples shown where a wide range of user specifications and optimization priorities are applied. It can also be seen that the predictions from the analytic circuit models are in rather good agreement with the SPICE simulation results. For dc characteristics, excellent agreement is gen-

erally achieved. For ac and transient characteristics such as phase margin, gain, and settling time, there are some differences, but all these deviations are less than 20%. If the user's design objectives are too demanding to be met, the program will provide the user with the best result. In the example shown in Table 2 the designer specified a very fast settling time of 100 nsec. The optimization procedure tried to comply as much as possible and ended up in 160 nsec.

CPU time for the synthesis phase varies with the difficulty of the user specifications and the degree of optimality to be achieved. The range of CPU times observed for the synthesis phase of the interpreted version of OPASYN is from 70 to 280 seconds on a VAX 8800. The SPICE verification phase typically requires about 200 seconds.

## 5. Conclusion

An efficient CMOS op amp synthesis tool (OPASYN) has been developed. It uses analytic circuit models to estimate circuit performance during the search for the optimal solution. Our experiments have shown that this approach greatly reduces the required CPU time while still producing near-optimal results.

OPASYN has been applied successfully to three of the most commonly used op amp circuit topologies using two different process technologies. The synthesis process has been reliable and produced good results.

OPASYN can be easily used by engineers inexperienced in op amp design. It does not normally require any user intervention in the design phase. Users simply specify their design requirements and optimization priorities and select the most desirable result out of several options produced by OPASYN.

## Acknowledgments

The authors would like to thank Dr. Richard D. Baertsch in General Electric Company for evaluating the program and providing valuable feedback on many practical aspects. We also gratefully acknowledge the contributions of many graduate students at U.C. Berkeley.

This work has been supported by DARPA VLSI-N00039-87-C-0182 and SRC-82-11-008.

## References

- [1] Harjani, R., A. Rutenbar, and L. R. Carley (1987). "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis," *24th Design Automation Conf.*, pp. 42-49.
- [2] Allen, P. E. (1986). "A Tutorial - Computer Aided Design of Analog Integrated Circuits," *Proc. of IEEE CICC*, pp. 608-616.
- [3] Pletersek, T., J. Trontelj, and L. Trontelj (1985). "Analog LSI Design with CMOS Standard Cells," *Proc. of IEEE CICC*, pp. 479-483.

- [4] DeGrauwe, M. G., et al. (1987). "An Analog Expert Design System," *IEEE ISSCC Digest of Technical Papers*, pp. 212–214.
- [5] Kelson, G. (1985). "Design Automation Techniques for Analog VLSI," *VLSI Design*, pp. 78–82.
- [6] El-Turky, F. M., and R. A. Nordin (1986). "BLADES: An Expert System for Analog Circuit Design," *Proc. of IEEE ISCAS*, pp. 552–555.
- [7] DeGrauwe, M. G. (1984). "A Synthesis Program for Operational Amplifiers," *IEEE ISSCC Digest of Technical Papers*, pp. 18–19.
- [8] Allen, P. E., and H. Nevarez-Lozano (1983). "Automated Design of MOS OP Amps," *Proc. of IEEE ISCAS*, pp. 1286–1289.
- [9] Bowman, R. J., and D. J. Lane (1986). "A Knowledge-Based System for Analog Integrated Circuit Design," *Proc. of IEEE ICCAD*, pp. 210–212.
- [10] Nye, B., D. Riley, and A. Sangiovanni-Vincentelli (1984). "DELIGHT.SPICE User's Guide," Dept. EECS, University of California, Berkeley.
- [11] Vladimirescu, A., K. Zhang, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli (1981). "SPICE Version 2G - User's Guide," Dept. EECS, University of California, Berkeley.
- [12] Horowitz, E., and Sahni (1978). *Fundamentals of Computer Algorithms*, Computer Sciences Press.
- [13] Gray, P. R., and Meyer, R. G. (1984). *Analysis and Design of Analog Integrated Circuits*, 2nd Edition. New York: Wiley.
- [14] Gray, P. R. (1982). "MOS Operational Amplifier Design- A Tutorial Overview," *IEEE J. Solid-State Circuits*, vol. SC-17, No. 6.
- [15] Ahuja, B. K. (1983). "An Improved Frequency Compensation Technique for CMOS Operational Amplifiers," *IEEE J. Solid-State Circuits*, vol. SC-18, pp. 629–633.
- [16] Chuang, C. T. (1982). "Analysis of the Settling Behavior of an Operational Amplifier," *IEEE J. Solid-State Circuits*, vol. SC-17, pp. 74–80.
- [17] Kamath, B. Y., R. G. Meyer, and P. R. Gray (1974). "Relationship between Frequency Response and Settling Time of Operational Amplifiers," *IEEE J. Solid-State Circuits*, vol. SC-9, pp. 347–352.

abbreviations used:

Vdd	=	positive power supply voltage
Vss	=	negative power supply voltage
CL	=	load capacitor
C <sub>c</sub>	=	compensation capacitor
wu	=	unity gain bandwidth
PSRR <sup>-</sup> @dc	=	Vss rejection ratio at dc
PSRR <sup>-</sup> @lkilo	=	Vss rejection ratio at 1 kHz
TS	=	settling time (1 V step, 0.1 % interval)
V <sub>o,max</sub>	=	maximum output voltage
V <sub>o,min</sub>	=	minimum output voltage
V <sub>ic,max</sub>	=	maximum common mode input voltage
V <sub>ic,min</sub>	=	minimum common mode input voltage
1/f noise	=	input equivalent 1/f noise at 1 kHz

parameter	specification	synthesis	SPICE
Vdd	2.5 V	2.5 V	2.5 V
Vss	-2.5 V	-2.5 V	-2.5 V
CL	10 pF	10 pF	10 pF
gain	10,000	23,610	24,250
power dissipation	1 mW	0.67 mW	0.66 mW
phase margin	60 deg	57.1 deg	56.1 deg
wu	4 MHz	4.7 MHz	4.7 MHz
gain margin	none	none	15.1 dB
CMRR	80 dB	none	92 dB
slew rate	2.5 V / $\mu$ sec	3.9 V/ $\mu$ sec	3.4 V/ $\mu$ sec
PSRR <sup>-</sup> @dc	70 dB	94.8 dB	94.8 dB
PSRR <sup>-</sup> @1kilo	40 dB	58.0 dB	74.3 dB
TS	500 nsec	453 nsec	550 nsec
systematic offset	none	none	0.008 mV
V <sub>o,max</sub>	1.5 V	2.38 V	2.40 V
V <sub>o,min</sub>	-1.5 V	-2.33 V	-2.39 V
V <sub>ic,max</sub>	1 V	1.45 V	1.40 V
V <sub>ic,min</sub>	1 V	-2.33 V	-2.50 V
1/f noise	1E-6 V/ $\sqrt{\text{Hz}}$	2.2E-7	none
total gate area**	40 mil <sup>2</sup>	35.3 mil <sup>2</sup>	none
C <sub>c</sub>	none	4.8 pF	4.8pF

Table 1. Synthesis and verification results for the basic two stage op amp (shown in Fig. 2) with optimization priority given to total gate area. MOSIS 3  $\mu$ m process was used. CPU time for the synthesis was 175 seconds on a VAX 8800.

parameter	specification	synthesis	SPICE
Vdd	2.5 V	2.5 V	2.5 V
Vss	-2.5 V	-2.5 V	-2.5 V
CL	2 pF	2 pF	2 pF
gain	1,500	1,421	1,496
power dissipation	30 mW	2.85 mW	2.72 mW
phase margin	60 deg	38.8 deg	33.0 deg
wu	4 MHz	42.3 MHz	30.0 MHz
gain margin	none	none	15 dB
CMRR	none dB	none	132 dB
slew rate	8 V / $\mu$ sec	22 V/ $\mu$ sec	19 V/ $\mu$ sec
PSRR <sup>-</sup> @dc	40 dB	122 dB	112 dB
PSRR <sup>-</sup> @1kilo	10 dB	122 dB	112 dB
TS**	100 nsec	164 nsec	160 nsec
systematic offset	0.1 mV	none	0.01 mV
V <sub>o,max</sub>	1.5 V	1.86 V	2.10 V
V <sub>o,min</sub>	-1.5 V	-1.88 V	-2.10 V
V <sub>ic,max</sub>	1 V	0.78 V	1.50 V
V <sub>ic,min</sub>	1 V	-2.50 V	-2.50 V
1/f noise	1E-6 V/ $\sqrt{\text{Hz}}$	1.5E-7	none
total gate area	70 mil <sup>2</sup>	75 mil <sup>2</sup>	none
C <sub>c</sub>	none	none	none

Table 2. Synthesis and verification results for the single stage folded cascode op amp (shown in Fig. 3) with optimization priority given to settling time. MOSIS 3  $\mu$ m process was used. CPU time for the synthesis was 278 seconds on a VAX 8800.

parameter	specification	synthesis	SPICE
Vdd	2.5 V	2.5 V	2.5 V
Vss	-2.5 V	-2.5 V	-2.5 V
CL	5 pF	5 pF	5 pF
gain	10,000	15,140	13,890
power dissipation	1 mW	0.85 mW	0.86 mW
phase margin	60 deg	65.3 deg	62.1 deg
wu	4 MHz	7.2 MHz	7.2 MHz
gain margin	none	none	9 dB
CMRR	none dB	none	95 dB
slew rate	2.5 V/ $\mu$ sec	4.7 V/ $\mu$ sec	4.6 V/ $\mu$ sec
PSRR <sup>-</sup> @dc	70 dB	90 dB	90 dB
PSRR <sup>-</sup> @1kilo	40 dB	90 dB	90 dB
TS**	500 nsec	387 nsec	420 nsec
systematic offset	none	none	0.46 mV
V <sub>o,max</sub>	1.5 V	2.37 V	2.39 V
V <sub>o,min</sub>	-1.5 V	-2.32 V	-2.39 V
V <sub>ic,max</sub>	1 V	1.45 V	1.50 V
V <sub>ic,min</sub>	1 V	-0.88 V	-1.00 V
1/f noise	1E-6 V/ $\sqrt{\text{Hz}}$	1.7E-7	none
total gate area**	40 mil <sup>2</sup>	38.7 mil <sup>2</sup>	none
C <sub>c</sub>	none	3.4 pF	3.4 pF

Table 3. Synthesis and verification results for the two stage op amp with cascoded first stage (shown in Fig. 4) with optimization priority given to settling time and total gate area. GE 1.5  $\mu$ m process was used. CPU time for the synthesis was 82 seconds on a VAX 8800.

# ANALOG CIRCUIT SYNTHESIS FOR PERFORMANCE IN OASYS

Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley

*Department of Electrical and Computer Engineering*

*Carnegie Mellon University*

*Pittsburgh, PA 15213*

## Abstract

This paper describes mechanisms needed to meet aggressive performance demands in a hierarchically-structured analog circuit synthesis tool. Experiences with adding a high-speed comparator design style to the OASYS synthesis tool are discussed. It is argued that design iteration — the process of making a heuristic design choice, following it through to possible failure, then diagnosing the failure and modifying the overall plan of attack for the synthesis — is essential to meet stringent performance demands. Examples of high-speed comparators automatically synthesized by OASYS are presented. Designs competitive in quality with manual expert designs, e.g., with response time of 6 ns and input drive of 1 mV, can be synthesized in under 5 seconds on a workstation.<sup>1</sup>

## 1. Introduction

This paper considers how to attack real-world performance issues in automatic synthesis tools for analog circuits. OASYS is a tool for synthesizing sized circuit schematics from process and performance specifications; a companion tool, ANAGRAM transforms these schematics into layouts [1]. OASYS represents circuits as a hierarchy of alternate design styles, and uses a planning mechanism to refine specifications down the hierarchy to primitive devices. Details of the OASYS synthesis framework, in particular this planning mechanism, appear in [2]. Examples of functional, fabricated CMOS op amps designed by OASYS appear in [3]. The purpose of this paper is to describe how synthesis in OASYS *changed* when a high-performance design style was added to the system. We focus on the mechanisms needed to meet aggressive performance demands in a hierarchical automatic synthesis tool.

Many previous attempts at full-custom circuit synthesis have two essential characteristics: a simplified target domain, and a simplified model of design [4]. For example, most recent synthesis systems have used operational amplifiers as a trial domain [3, 5, 6, 7]. Unfortunately, many critical design constraints do not manifest themselves here: op amps are essentially linear circuits, can be analyzed primarily by using small signal models, do not have devices changing regions of operation during normal use, etc. Not all design problems have these characteristics, and it is precisely the need to identify, model and manage sub-

stantial non-linearities, transients, changes in operating region, etc., that is critical in high-performance designs. In addition, many previous synthesis attempts model design as a rather simple process. At one end of the spectrum, tools such as IDAC [5] model synthesis as a simple sequence of equations, known *a priori*, whose solution gives a complete design. When this scheme fails, the input specifications are tightened, even though it may be possible that an *alternate* plan of attack for the design might succeed. At the other end of the spectrum, tools such as OPASYN [7] model design with a complex set of coupled non-linear equations solved with aggressive numerical techniques. This approach can support complex performance requirements, but seems currently limited to those situations where *all* relevant aspects of analog circuit behavior can be captured beforehand in a single set of static equations. Moreover, the CPU time required here is substantially greater, effectively precluding exploration of different designs in the analog design space. In addition, these tools can generate only flat, specified circuit topologies, further limiting opportunities to achieve higher-performance designs by *changing* topologies as necessary.

In contrast, expert human designers explore design problems as a design evolves, change models or topologies, simplify as necessary, and iterate on their designs to converge to workable solutions. This paper argues that the notion of *iteration*, i.e., making a heuristic design choice, following it through to possible failure, then diagnosing the failure and modifying the plan of attack, is central to synthesis of tightly-specified, high-performance designs. This is the synthesis style followed by the planning mechanism in OASYS. We use experience gained from the addition of a high-performance CMOS comparator design style to OASYS to support his argument. The remainder of the paper is organized as follows. Section 2 describes the key concerns behind the design strategy for this comparator, and compares qualitatively some of the design decisions required by the new OASYS comparator style against those required by the earlier OASYS op amp styles. Section 3 presents results, in the form of several OASYS-synthesized comparator schematics, and simulation data. Finally, Section 4 presents concluding remarks.

## 2. Design Strategy

High-performance comparators were chosen as a test domain because, like op amps, they are widely used as building blocks, but *unlike* op amps, they have substantial non-linearities, transients, and so forth. The particular comparator design style we chose is a novel regenerative, latch-based comparator. This design style, as are all OASYS styles, is explicitly hierarchical, and is completed by alternating selection of a design style with refinement of specifications down to the next, more primitive level. Hence, several different device-level topologies can be reached from this single style. In OASYS, a design style is formally

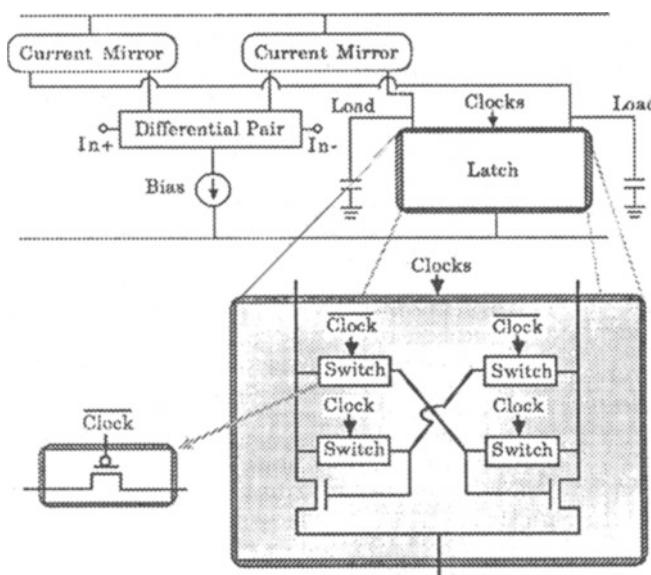


Figure 1. Comparator Design Style.

an interconnection of abstract blocks. Adding a design style to OASYS requires specifying this topology of blocks, and adding a *design plan* for refining overall performance specifications down to specifications for the component sub-blocks [2]. The design style for this comparator appears in Figure 1.

The key difficulty for this high-performance comparator style is the construction of the design plan. Roughly speaking, there are three requirements for producing such a design plan. First, we need detailed knowledge of which effects can be modeled, either analytically or numerically. Second, we need knowledge of where heuristic design decisions are required to advance the design (because, for example, there is insufficient information in the current design state to know with certainty the optimum decision). Finally, we need to identify and manage the *degrees of freedom* in the design problem. These are the explicit performance tradeoffs available in a specific design style; choices here are complicated by the fact that frequently there is no clear analytical model to prefer one choice over another. As design plans are executed to refine specifications, heuristic design decisions made earlier are often found later to be imprecise, requiring that earlier phases of the design be repeated. OASYS incorporates a rule-based failure diagnosis and re-planning mechanism to identify failures, correct the overall plan of attack for the design, and restart the internal design plan at the appropriate point.

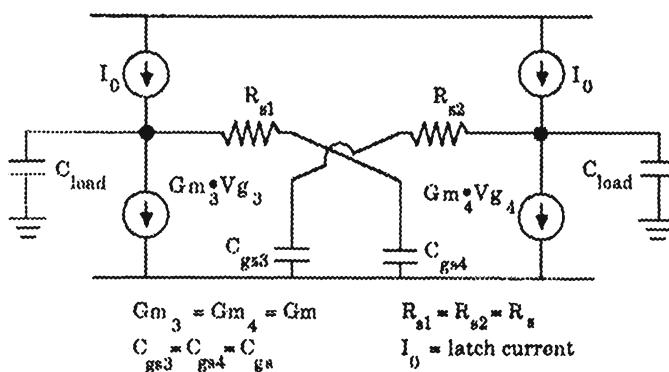


Figure 2. Small-Signal Model for the Latch.

Because of space limitations, we omit a complete, analytical description of all the design steps in the comparator design plan; these appear in [9]. Instead, we focus on four key concerns that must be dealt with in *any* analog design plan:

- 1 Simplifications versus subtle effects
- 2 Compensating for parasitics
- 3 Managing degrees of freedom
- 4 Managing synthesis within a hierarchy

and illustrate some important design decisions in the comparator design plan that cope with these concerns.

## 2.1 Simplifications versus Subtle Effects

Simplifications are essential to suppress unwanted details that obscure general insights about overall circuit behavior. But whereas in low-performance designs these suppressed details really can be ignored, in high-performance designs they must eventually be recovered and considered. For example, it is common to neglect the channel length modulation ( $\lambda$ ) effect while calculating the transconductance ( $g_m$ ) of a MOS device, however, for large signal operation, e.g., large supply voltages, channel length modulation is no longer a negligible effect. With respect to our comparator design problem, an essential simplification involves the time domain response of the latch within the comparator. Figure 2 shows a simplified small-signal model for this latch.

Even though this latch operates primarily as a “large-signal” circuit — because it sees large voltage transients — the time domain response is dominated by the small-signal behavior. If we assume that the switch resistance ( $R_s$ ) is

negligible, as is normally done, then a second order characteristic equation describes the latch, and the time domain response is dominated by a single pole [10]. In our latch, we assume that the switch resistance  $R_s$  is non-zero, and contrary to common belief, this actually reduces the response time of the latch because the two critical capacitors,  $C_{gs}$  and  $C_{load}$  in Figure 2, are decoupled by the switch resistances  $R_s$ . For the same voltage swing at the latch output node, the gate voltages need not vary as much as when  $R_s$  is assumed zero. Unfortunately, a fourth-order characteristic equation now describes this latch, seemingly precluding simple analytical models for synthesis. However, in the case that the circuit is totally symmetrical, it is possible to show [9] that the solution to this fourth-order system has a non-dominant real pole, a complex-conjugate pair in the left-half plane (LHP) and a dominant pole in the right-half plane (RHP). Thus, the time domain response is still dominated by a single pole, but the solution incorporates the non-zero value for  $R_s$ . We employ this simplified model to suggest the “correct” avenue of attack when designing the latch, but return to the higher-order model to verify our design decisions. In addition, though this simplified model assumes that the latch response is dominated by one particular state, i.e., the regions of operation of each of the latch’s constituent devices, note that we must compensate for the fact that these devices actually change state as the latch switches. Thus, we incorporate heuristics to choose the right models dynamically, to meet desired design goals. In all cases, more complete models are used to verify the evolving circuit design after these heuristic choices.

The process of selecting and validating such simplifications is central to the construction of a design plan within OASYS. While “designing” such simplifications, some effects are ignored not because they are difficult, but because they clutter models. These can be handled by switching from simple to complex models as needed during synthesis. More important there are subtle effects that really only appear when performance limits are pushed hard. A good example in our comparator concerns the analog switches within the latch sub-module. Charge that is stored in the channel of the switch transistors when they are conducting is dumped into the local circuit environment of the surrounding latch when the switches turn off. If this effect is neglected, we cannot accurately predict high-speed behavior. A simple model of this environment appears in Figure 3.

Due to the (typical) assumption of minimal channel length for the switch devices, and finite rise and fall times for the clock, we can safely neglect charge injection into the substrate [11]. Due to the symmetrical nature of our latch, we can now assume that all the channel charge, accumulated during the switch’s conducting period, is equally distributed between the source and the drain terminals of the transistor  $Q_S$  in Figure 2 during the turn-off transient.

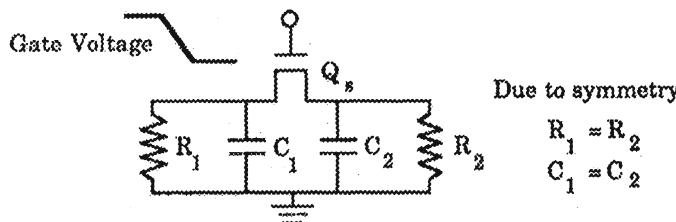


Figure 3. Switch Environment.

## 2.2 Compensating for Parasitics

Some subtle effects, such as the charge dump from the switches, can be predicted adequately. In contrast, device parasitics, though extremely influential during synthesis, cannot be calculated until *after* devices are actually designed. Hence, compensating for parasitics in OASYS design plans almost always involves iteration, i.e., modifying and re-executing a design plan. Parasitics are estimated with heuristics, design proceeds, and if subsequent verification proves the estimate sufficiently wrong, the design process is reiterated. Specifically, whenever a step in an OASYS design plan is able to calculate an actual parasitic, it also compares this value with any previously predicted value. If these values are sufficiently different, a heuristic is invoked to decide how many of the recent steps in the design should be redone, using this newer, more accurate parasitic estimate. In this way, sequences of plan steps can be iterated until the predicted and calculated values converge. Usually, if the sequence of design activities to be iterated is carefully constructed, predicted and calculated values actually do converge. However, sometimes convergence is never reached, because the design is simply infeasible; it is impossible adequately to meet all performance specifications in the current circuit design style, using the existing models and heuristics.

## 2.3 Managing Degrees of Freedom

Another important component in the generation of a design plan for the comparator is identification and management of the degrees of freedom in the design. For example, one critical degree of freedom here is the area/power tradeoff. With respect to our simplified models, constraints exist that limit the extremal points in the spectrum of feasible area, power values, but many values in between are viable, if not necessarily optimal. Heuristics were developed to choose the best "bias" for this tradeoff when design decisions were required, or when failures were encountered during synthesis and the plan of attack needed

modification. In this comparator topology, higher speed is achieved primarily by manipulating bias currents, and to a lesser extent by increasing device sizes. Indeed, many of the devices tend to be designed small to minimize the parasitics they contribute (each of which may slow the comparator's overall response time). As an example of managing this tradeoff, the design plan for the latch sub-module currently employs the simplified first-order model described in Sec. 2.1 to predict an approximate family of feasible (area, power) points meeting the desired response time. (Here, “area” estimates active device area only, i.e., device sizes, not routing area, etc.) The very simple models are attractive here since they can be used quickly to explore this area versus power surface. At this rather coarse level of detail, we bias the tradeoff in favor of minimum area, and for a specific area choice, use higher-order models to compute the required power. Unfortunately, (area, power) choices that look promising with coarse models are not always valid when revisited with more accurate models. Our strategy here is to first try to increase the power gradually, keeping the area constant, until the response time specification is satisfied, or the power is deemed unreasonable large; if the power becomes too large, we increase the area slightly, and again search for a reasonable power value. Note that other strategies are feasible here, but this approach appears to work well.

We refer to the process described above as “absorbing” a degree of freedom: the tool, and not the designer, determines the tradeoff. A goal for a mature, complete design plan is to identify and absorb all relevant degrees of freedom. Because the current OASYS comparator design plan is still immature, not all degrees of freedom are absorbed; some of these are temporarily promoted to the level of specifications for the designer to choose. Currently, a few tradeoffs related to the overall gain, and gain partition among the sub-blocks of the comparator style, e.g., the gain of the input differential amplifier, are specified by the designer. However, our expectation is that with experimentation, adequate heuristics will be determined to manage these decisions internally. Indeed, a similar tradeoff, the gain partition between the stages of the OASYS two-stage op amp is absorbed internally, and its performance has improved as this op amp design plan has matured [9]. It is also interesting to note that other synthesis architectures, notably IDAC [5], routinely avoid *all* such tradeoff decisions, forcing the designer to specify manually many of these internal design tradeoffs, along with ordinary performance specifications.

## 2.4 Managing Synthesis within a Hierarchy

OASYS was the first analog synthesis tool to make extensive use of hierarchy even for basic cells like op amps and comparators; subsequent tools have adopted this approach [12]. This hierarchy helps greatly to simplify the process of design for complex objects. Unfortunately, it also complicates attempts to

push hard on performance limits, because the flow of design information across boundaries of modules within the hierarchy is restricted and stylized. An example in the comparator occurs within the design plan for the latch. The latch needs to know the amount of charge dumped by the analog switches within it, but it actually does not know what topology will be used for these switches. In the OASYS hierarchy, the switches are a separate block, with their own independent design plan. To handle this, the latch computes the maximum allowable charge dump, invokes the design plan for the switches, and analyzes the prediction for charge dump returned by the switches. If the prediction is not within acceptable limits, the separate latch and switch design plans iterate to reach a mutually agreeable value. However, if it becomes apparent that the switches cannot possibly meet their specification, the latch design plan itself is determined to have failed, and the overall comparator design plan will be modified and reinvoked to try to avoid this problem.

Superficially, it may seem that the need to negotiate among sub-blocks and iterate toward convergence that is imposed by a strict hierarchy is more a burden than an advantage. However, we regard this a small price to pay; there are many striking advantages to this hierarchical formulation [2]. Most notably, design knowledge is encapsulated in the form of reusable sub-blocks. There is only one current mirror designer in OASYS, despite the fact that several op amp and comparator topologies require mirrors, and this mirror designer can design in any of several circuit styles. Moreover, design is decomposed into smaller, more easily managed tasks. For example, our comparator is not designed at the level of transistors, but rather, at the level of latches, differential pairs, etc., all specified by the behavior seen at their terminals. Finally, because all sub-block can be realized in several different design styles, a single high-level design style can be refined down to many distinct transistor-level circuit topologies. Thus, a small hierarchy of design plans can reach a very large population of different flat circuit topologies (avoiding the need for a large library of synthesis tools for each flat circuit schematic).

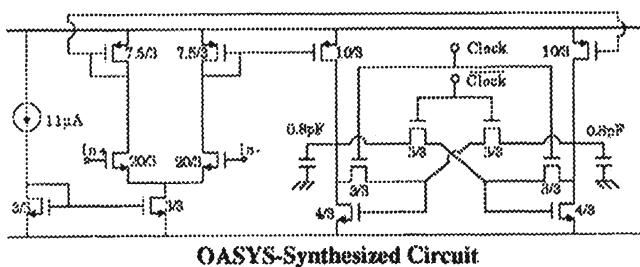
### 3. Results

Prior to this study, OASYS was capable of designing CMOS op amps in OTA and generic two-stage Miller-compensated styles. A wide variety of lower-level circuit styles, essentially used as building blocks, were also supported: current mirrors, differential pairs, level shifters, etc. Full implementation of the comparator style increased the size of OASYS by about 25%, to approximately 10,000 lines of Franz Lisp. This required addition of designers (i.e., design styles and plans) for the comparator itself, as well as for two new lower-level blocks: analog switches and latches.

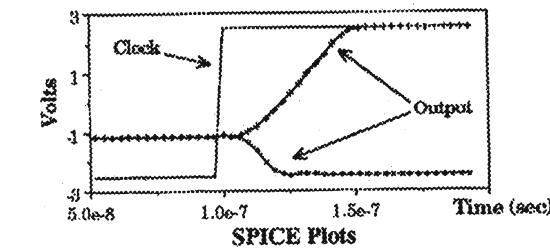
Figure 4 shows the specifications for a modest comparator, the OASYS-synthesized design, and results from a SPICE simulation for verification. This

	Input Specs	Spice Results
Input Drive	$\leq 1\text{mV}$	0.5mV
Response Time	$\leq 100\text{ns}$	65ns
Estimated Power	$\leq 0.5\text{mW}$	0.12mW
Load Capacitance	0.8pF	-

Input Specifications and SPICE Results



OASYS-Synthesized Circuit



SPICE Plots

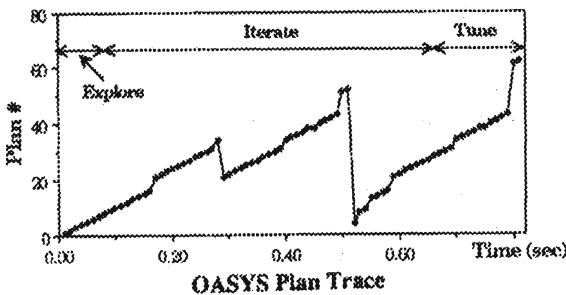


Figure 4. Simple Comparator Synthesis Example.

comparator has a response time of 65 ns, with an input drive of 0.5 mV and estimated power consumption of 0.12 mW. Total design time was about 1 second on a microVAX. All designs were simulated using models from the MOSIS 3  $\mu\text{m}$  bulk CMOS process. Also included here is a *plan trace* for this synthesis.

Individual plan steps within OASYS have been uniquely numbered, and their execution plotted as a function of design time. These traces illustrate succinctly how designs evolve. For example, an "ideal" design plan appears as a monotonically increasing straight line: each plan step is visited exactly once, in the default order specified by each plan at each level of the hierarchy; no steps are repeated. However, in our experience, comparators are never so ideal. The plan trace shown in 4 suggests this was a fairly simple design problem: one pass through the default design plan, with only two plan corrections (the discontinuities in the trace) sufficed to meet specifications.

The situation changes considerably when high-performance designs are attempted. Figure 5 shows a much more aggressive comparator designed for a response time of 5.75 ns and 1 mV input drive. Power consumption is estimated to be 1.01 mW. Total design time was about 3 seconds on a microVAX. This circuit's performance compares favorably with the speed, resolution, and power tradeoffs of several published manual designs [13, 14, 15, 16, 17]. Notice that the topology has changed from that shown in Figure 4. A more complex topology for the analog switches was chosen to compensate for charge dumping. The plan trace is also significantly different for this design: there is considerable iteration and replanning activity. This synthesis has three different phases. First, obviously wrong attempts are made and rejected. Second, a good plan of attack evolves, but must be iterated extensively; the specific heuristic decisions made in individual plan steps are not quite right, and many alternatives are explored before the design converges. Finally, a complete retuning with respect to more accurate final parasitic estimates is attempted, and the design completes.

Direct comparison with other analog synthesis systems is difficult, since none of which we are aware attack high-speed comparator design explicitly. For example, the ultra-low-power comparators designed by IDAC [5] appear to have response times around 0.5  $\mu$ s at best. The comparator used in the A/D designer AIDE2 [18] is essentially just a fixed standard cell, with roughly a 1  $\mu$ s response time.

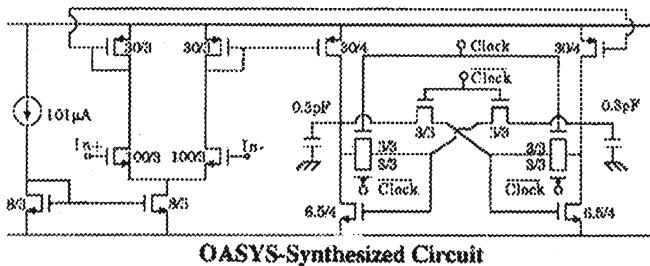
## 4. Conclusions

From experience with adding a high-performance comparator style to OASYS, we observe that high-performance designs require substantial exploration among tradeoffs, i.e., design iteration, to find workable designs. A variety of high-performance comparators have been designed by OASYS and successfully simulated. These results are significant because these designs can be synthesized in under once minute of CPU time, and yet appear to approach performance limits roughly comparable to manually produced expert designs.

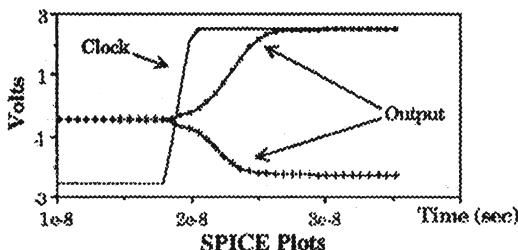
Current work is focused on extensive tuning of the comparator design heuristics (notably, dealing with unabsorbed degrees of freedom), broadening the com-

	Input Specs	Spice Results
Input Drive	$\leq 2\text{mV}$	1mV
Response Time	$\leq 10\text{nS}$	6.75nS
Estimated Power	$\leq 1.5\text{mW}$	1.01mW
Load Capacitance	0.3pF	-

Input Specifications and SPICE Results



OASYS-Synthesized Circuit



SPICE Plots

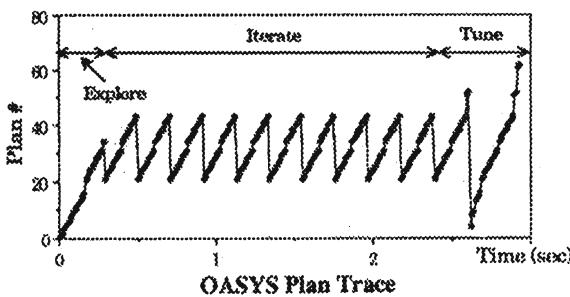


Figure 5. Aggressive Comparator Synthesis Example.

parator's design hierarchy to include more latch and switch sub-block styles, and accommodation of noise and input offset specifications for the comparator.

## Notes

1. This research was supported in part by the Semiconductor Research Corporation, by the National Science Foundation, under grants MIP-8657369 and ENG-8451496, and by a grant from Texas Instruments.

## References

- [1] Garrod, D. J., R. A. Rutenbar, and L. R. Carley (1988). "Automatic Layout of Custom Analog Cells in ANAGRAM," *Proc. IEEE ICCAD*.
- [2] Harjani, R., R. A. Rutenbar, and L. R. Carley (1987). "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis," *Proc. ACM/IEEE Design Automation Conf.*
- [3] Harjani, R., R. A. Rutenbar, and L. R. Carley (1988). "Analog Circuit Synthesis & Exploration in OASYS," *Proc. IEEE ICCT*.
- [4] Carley, L. R., and R. A. Rutenbar (1988). "Analog IC Design," *IEEE Spectrum*, Vol. 25, No. 8.
- [5] Degrauwe, M. G. R., et al. (1987). "JDAC: An Interactive Design Tool for Analog CMOS Circuits," *IEEE JSSC*, Vol. SC-22, No. 6.
- [6] Bowman, R. J., and D. J. Lane (1985). "A Knowledge-Based Approach to Analog IC Design," *IEEE ICCT*.
- [7] Koh, H. Y., C. H. Sequin, and P. R. Gray (1987). "Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models," *Proc. IEEE ICCAD*.
- [8] Sheu, B. J., A. H. Fung, and Y-N. Lai (1988). "A Knowledge-Based Approach to Analog IC Design," *IEEE Trans. Circuits and Systems*, Vol. 35, No. 2.
- [9] Harjani, R. (1988). *OASYS: A Framework for Analog Circuit Synthesis*, PhD dissertation, Elec. & Comp. Engin., Carnegie Mellon, in preparation.
- [10] Gregorian, R., and G. C. Temes (1986). *Analog MOS Integrated Circuits for Signal Processing*, John Wiley & Sons.
- [11] Wegmann, G., E. A. Vittoz, and F. Rahali (1987). "Charge Injection in Analog MOS Switches," *IEEE JSSC*, Vol. SC-22, No. 6.
- [12] Berkcan, E., M. D'Abreu and W. Laughton (1988). "Analog Compilation Based on Successive Decomposition," *Proc. ACM/IEEE Design Automation Conf.*
- [13] Yee, Y. S., L. M. Teman, and L. G. Heller (1978). "A 1 mV MOS Comparator," *IEEE JSSC*, Vol. SC-13, No. 3.
- [14] Allstot, D. J. (1982). "A Precision Variable-Supply CMOS Comparator," *IEEE JSSC*, Vol. SC-17, No. 6.
- [15] Ng, W. T., and C. A. T. Salama (1986). "High-Speed High-Resolution CMOS Voltage Comparator," *Electronics Letters*, Vol. 22, No. 6.
- [16] Koch, R., et al. (1986). "A 12-bit Sigma-Delta Analog-to-Digital Converter with a 15-MHz Clock Rate," *IEEE JSSC*, Vol. SC-21, No. 6.
- [17] McCarroll, B. J., C. G. Sodini, and H.-S. Lee (1988). "A High-Speed CMOS Comparator for Use in an ADC," *IEEE JSSC*, Vol. SC-23, No. 1.
- [18] Barton, P. R. (1986). "A Synthesis Program for CMOS Successive Approximation A/D and D/A Converters," Tech. report T86033, Semiconductor Research Corp., Feb. 1986.

# EXTRACTION OF GATE-LEVEL MODELS FROM TRANSISTOR CIRCUITS BY FOUR- VALUED SYMBOLIC ANALYSIS

Randal E. Bryant  
*Fujitsu Laboratories, Ltd.*  
*Kawasaki, JAPAN*  
*(On leave from*  
*Carnegie Mellon University*  
*Pittsburgh, PA 15213 USA)*

## Abstract

The program TRANALYZE generates a gate-level representation of an MOS transistor circuit. The resulting model contains only four-valued unit and zero delay logic primitives, suitable for evaluation by conventional gate-level simulators and hardware simulation accelerators. TRANALYZE has the same generality and accuracy as switch-level simulation, generating models for a wide range of technologies and design styles, while expressing the detailed effects of bidirectional transistors, stored charge, and multiple signal strengths. It produces models with size comparable to ones generated by hand.

## 1. Introduction

Switch-level simulation has proved an effective means for verifying digital circuits implemented in MOS technology. By directly working from a transistor representation, a switch-level simulator can handle a large range of circuit designs and capture such subtle effects as bidirectional signal flow, dynamically stored charge, and multiple signal sources with different driving impedances. Traditionally, switch-level simulation requires evaluation mechanisms that are not found in conventional gate-level simulators. To utilize the features and modeling libraries of existing simulators, and for execution on gate-level hardware simulation accelerators, we would like to overcome this incompatibility.

By automatically generating gate-level models from transistor circuits, we can provide a simulation methodology that combines switch-level generality and accuracy with gate-level compatibility and performance. In taking this approach, we should take care to satisfy several design constraints. First, we must not give up the generality and accuracy of switch-level simulation. Second, the generated models should be suitable for evaluation by any gate-level simulator. Third, the generated models should be as compact as possible.

## 1.1 Previous Work

A variety of programs have been developed that fall into the general category of gate-level model extractors. On close examination, however, we see that all of them fall short in at least one of the design goals listed above.

Most model extractors operate by repeatedly applying transformation rules that eliminate or reduce some of the transistor logic and replace it by logic gates [3, 4]. These rules typically include transformations such as: series/parallel reduction of pullup and pulldown networks; recognition of static logic gates; identification of complementary control signals; merging transistor pairs in transmission gates; elimination of “uninteresting” nodes [1]; transistor direction assignment [2]; and recognition of special structures such as multiplexors and busses. Often, these programs encounter cases where none of the transformation rules apply, but transistors remain in the circuit model. The user must then either supply hand-generated models or add new transformation rules to the model extractor.

An alternative to rule-based approaches is to generate models using methods derived from switch-level simulation. This has the advantage of starting from a firm foundation in terms of generality and accuracy. The challenge becomes to satisfy the remaining goals of compatibility and conciseness. Symbolic switch-level analysis, as exemplified by the ANAMOS program [7] (the preprocessor for the COSMOS simulator [5]), can extract a logic description from the transistor network. ANAMOS generates models consisting of Boolean DAGs: networks of 2-valued operations describing the computation for one unit time step of the simulator. These DAGs operate on encoded signal values—each 3-valued node state is encoded as a pair of binary values.

In earlier work at CMU, we developed a program HLGCC to convert the Boolean DAGs generated by ANAMOS into logic gates [9]. HLGCC reduces the model size by mapping the two-valued operations of the Boolean DAGs into 3-valued logic primitives, and merging multiple operations into three-input logic gates. These optimizations, however, had only limited success. As an example, for a CMOS transmission gate multiplexor, ANAMOS generates a DAG containing 17 Boolean operations, which HLGCC maps into 10 three-input gates. Ideally, we should be able to generate a single gate model for this circuit. Several reasons can be identified for this shortcoming. First, by operating on encoded signal values, ANAMOS loses much of the original context about the node state values. Second, by analyzing each channel-connected component independently, ANAMOS cannot exploit correlations between signals, e.g., complementary signal pairs. Finally, ANAMOS generates a very conservative model, assuming that all transistors have unit delay, and that the effects of X signals should be modeled very strictly. Thus, the generated models must include many terms for evaluating sneak paths and unit delay glitches.

## 1.2 New approach

As with ANAMOS, TRANALYZE performs a symbolic, switch-level analysis to extract a logic representation of the circuit behavior. Instead of using a binary encoding of signal values, however, it performs the analysis in an algebra with values 0, 1, X, and Z. The fourth value Z is similar to the “high impedance” value used by many gate-level simulators for simulating tri-state drivers and busses. The analysis algorithm has the same general form as that used in ANAMOS [7]—each channel-connected component is analyzed by setting up systems of equations, which are solved by a symbolic form of Gaussian elimination [6] to yield the logic representation. TRANALYZE differs from its predecessor in the following key respects: it represents and manipulates its logic description as a 4-valued gate-level network rather than as a Boolean DAG; it analyzes the channel-connected components in rank order, so that it can exploit the correlations between the input signals of each component; and the optimizer can optionally generate models with less conservative, and hence simpler handling of X signals.

During the analysis, TRANALYZE applies extensive optimizations of the gate-level model to reduce its size. The combination of symbolic analysis plus logic optimization effectively performs many of the circuit optimizations implemented explicitly by rule-based systems.

## 2. Generated gate-level models

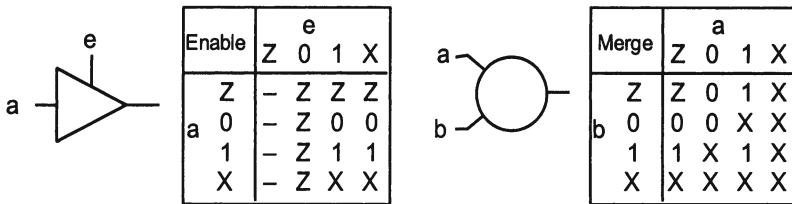
In a switch-level network, each node may have state 0, 1, or X, where X indicates either an unknown value or a potentially nondigital voltage. To represent intermediate terms in the analysis, we introduce a fourth value Z indicating the absence of a signal. In the gate-level model produced, we can guarantee that no circuit node will ever have state Z—at the very least an isolated node will retain its stored charge.

An MOS transistor can serve in two different capacities. First, as in relay contact networks, series-parallel (and other) configurations can be formed that conditionally connect two terminals according to some logic function of the transistor gate nodes. We will refer to this as “And/Or” logic. Second, a transistor can propagate a data value from its source to its drain (or vice-versa) under the control of its gate node. We will refer to this as “Steering” logic. Typical MOS circuits contains both forms of logic. Logic gates are created using And/Or logic as pullup and pulldown networks. Transmission gates, multiplexors, and carry chains are created using steering logic. Our signal algebra includes operators to express both forms of logic.

## 2.1 Primitive gates

We express And/Or logic using gate types AND and INVERT. The OR operation is expressed in terms of these two gate types according to DeMorgan's Laws, aiding the detection of identical and complementary terms. We can guarantee that the Z state will never arise in And/Or logic.

We express Steering logic using gate types ENABLE and MERGE. These operations are defined as follows, where entry '-' indicates a condition that will never arise in the generated gate-level network:



The ENABLE gate conditionally propagates the data on input a according to the control signal on input e. As the table indicates, the control input can never equal Z—its value is generated by And/Or logic. This gate has functionality similar to that of a tri-state buffer, except that it treats control signal value X the same as a 1. TRANALYZE resolves the effects of unknown control signals on tri-state buffers by including additional gates in the model. The MERGE gate combines two 4-valued signals. Its functionality is identical to a “wired-logic” function in tri-state logic. As a final gate type, the DELAY gate implements a unit delay. All other gates have zero delay.

## 2.2 Gate-level logic example

Figure 1 shows a CMOS circuit and the generated gate network. As can be seen, the program successfully recognizes that the pullup and pulldown networks form a NOR gate driving node S. The logic generated for node T selects either the output of the NOR gate, or the stored value on node T (delayed by 1 time unit), depending on control signal C. After technology mapping, TRANALYZE generates a 2 gate model, consisting of a NOR and a multiplexor. In contrast, HLGCC generates a 7 gate model.

## 3. Analysis method

Following the parsing of a transistor netlist file, the analysis proceeds by a series of steps.

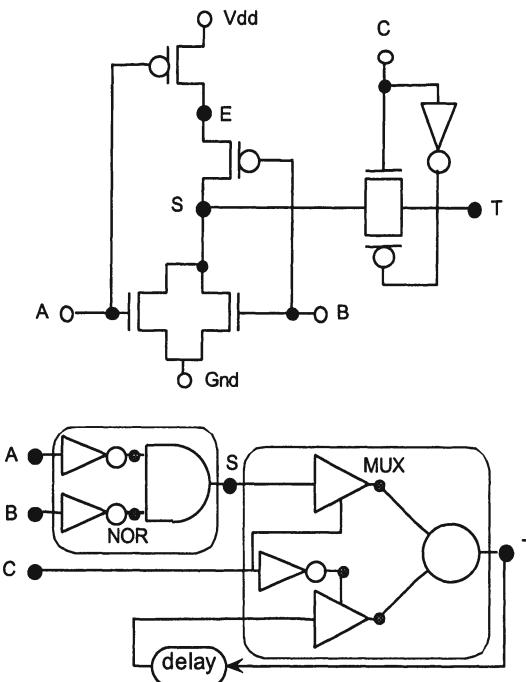


Figure 1. Example Circuit and Generated Model.

### 3.1 Partitioning and ordering

The transistor circuit is first partitioned into “channel-connected components,” each consisting of a set of storage nodes connected by the source-drain terminals of transistors. Each such region is analyzed separately.

TRANALYZE exploits the logical dependencies implied by zero delay transistor logic in its logic optimization. It does this by analyzing the circuit components in *rank order*, so that as a component is analyzed, the gate-level models for the signals controlling zero delay transistors are available to the optimizer. TRANALYZE automatically inserts unit delays on some of the transistors to enable a rank ordering and to avoid generating a gate network with zero delay cycles. It does this by a simple greedy method during the topological sorting of the components.

### 3.2 Symbolic analysis

During the analysis of a channel-connected component, gates are added to the generated network describing the functionality of the component nodes. Each component node is temporarily treated as a primary input fed through a unit delay to represent the initial node charge. Working from the maximum strength level downward, two systems of equations are set up and solved symbolically at

each strength level. The gate outputs representing the final steady state solution are then connected to the primary inputs that were temporarily introduced for the storage nodes.

For strength level  $s$ , the first system of equations, termed the “clear” equations, expresses the conditions under which each node is not the destination of a definite path of strength  $s$ . These equations are expressed in terms of And/Or logic. The second system, termed the “state” equations, expresses the combined effect of all unblocked paths of strength greater than or equal to  $s$  to each node. These equations are expressed in terms of Steering logic, with ENABLE expressing conditional signal propagation and MERGE expressing the effect of multiple signals to a single destination.

### 3.3 Symbolic manipulation

During the setting up and the solving of equations, each signal corresponds to either a primary input or the output of a logic gate. To “compute” the result of applying some operation (AND, MERGE, etc.) to a set of argument signals, the symbolic manipulator either finds an existing gate with the appropriate functionality, or it adds a new gate to the network having the argument signals as inputs. The symbolic manipulator also applies extensive optimizations as it proceeds. Most of the optimizations are similar to those found in ANAMOS, as well as in many optimizing compilers, e.g., constant evaluation, common subexpression detection, etc., except that it performs these optimizations in the 4-valued signal algebra. Although most of these optimizations require only constant time, others involve attempting a simple form of proof by contradiction to show that a candidate gate would always produce an output value equal to one of its inputs.

Unlike ANAMOS where the effects of X signal values are modeled very conservatively, TRANALYZE can generate models with “Boolean” optimization, ignoring the effects of X values in And/Or logic. For most applications, designers are willing to give up detailed modeling of X values in favor of simpler gate-level models. Indeed, they would find the gate model generated by Boolean optimization less prone to false X value propagation.

Additional transformations are implemented by the symbolic manipulator to convert Steering logic into And/Or logic. The first two illustrated in Figure 2 effectively perform series-parallel reduction of the transistor network. The first takes a chain of ENABLE gates, such as arises from the analysis of a series transistor chain, and collapses it into a single ENABLE controlled by an AND. The second takes the MERGE of a set of ENABLE gates having a common data input, such as arises from the analysis of a set of parallel paths, and collapses it into a single ENABLE gate controlled by the OR of the parallel control signals. This OR is implemented as a combination of INVERT and AND to exploit the equivalences implied by DeMorgan’s Laws. The third rule of Figure 2 illustrates

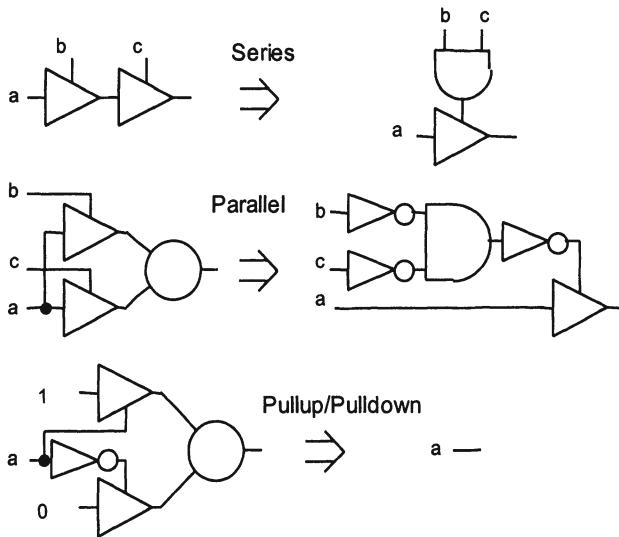


Figure 2. Example Transformation Rules.

Circuit	Trans.	ANAMOS 2 inp.	HLGCC	
			2 inp.	3 inp.
74181 ALU	240	265	193	131
Figure of Merit		<b>1.10</b>	<b>0.80</b>	<b>0.55</b>
Shift64	658	1669		
Figure of Merit		<b>2.54</b>		
DRAM256	1140	8346	8562	4888
Figure of Merit		<b>7.32</b>	<b>7.51</b>	<b>4.29</b>
SLAP	20167	80057	74498	45085
Figure of Merit		<b>3.97</b>	<b>3.69</b>	<b>2.24</b>

Figure 3. Performance of Previous Switch-Level Analysis Tools. Figure of merit is defined as (Gate Count)/(Transistor Count).

the final transformation required to extract static logic gates. The reduction of the pullup and pulldown networks by series-parallel transformations yields a pair of complementary signals. These signals control ENABLE gates to constants 1 and 0, representing power and ground. This final configuration of ENABLE and MERGE gates can then be eliminated.

### 3.4 Network pruning

During the analysis, TRANALYZE generates gate level logic describing the functionality of every storage node in the circuit, including such nodes as the intermediate points in series transistor chains. TRANALYZE prunes the network by a form of mark-sweep garbage collection. Starting with the gates rep-

Circuit	Trans.	Unit/Ternary			Zero/Binary		
		2 inp.	3 inp.	4 inp.	2 inp.	3 inp.	4 inp.
74181 ALU	240	194	120	98	185	122	89
<b>Figure of Merit</b>		<b>0.81</b>	<b>0.50</b>	<b>0.41</b>	<b>0.77</b>	<b>0.51</b>	<b>0.37</b>
Shift64	658	773	454	326	515	323	195
<b>Figure of Merit</b>		<b>1.17</b>	<b>0.69</b>	<b>0.50</b>	<b>0.78</b>	<b>0.49</b>	<b>0.30</b>
DRAM256	1140	5430	3172	2361	5296	3087	2310
<b>Figure of Merit</b>		<b>4.76</b>	<b>2.78</b>	<b>2.07</b>	<b>4.65</b>	<b>2.71</b>	<b>2.03</b>
SLAP	20167	67035			42861		
<b>Figure of Merit</b>		<b>3.32</b>			<b>2.13</b>		

Figure 4. Performance of TRANALYZE. Figure of merit is defined as (Gate Count)/(Transistor Count).

resenting the circuit nodes that the user wishes to observe during simulation, the program traces backward, marking all reachable gates. Any unmarked gate can then be eliminated; its output value can have no bearing on the simulation results. In this manner, the program prunes a large fraction of the nodes from the transistor circuit, eliminating the need for heuristic methods to identify these nodes initially [1]. As an example, connection node E in the pullup chain of the NOR gate of Figure 1 is successfully eliminated by the pruning.

### 3.5 Technology mapping

As the final stage, the primitive gates are merged into a smaller number of more complex gate types. The initial target simulator for TRANALYZE is the hardware simulation machine SP [8]. This machine can model arbitrary 4-valued gates with up to four inputs. Our technology mapper merges trees of gates, using a simple tree matching algorithm [10].

## 4. Experimental results

Figures 3 and 4 show the results of the different analysis methods for several switch-level benchmarks. Results are given for 4 different representations: the Boolean DAG model generated by ANAMOS, the ternary gate model generated by HLGCC, and the 4-valued models generated by TRANALYZE for two extremes of network optimization. Unit/ternary indicates that all transistors have unit delay, and X values are modeled conservatively. The resulting model is functionally equivalent to that produced by ANAMOS. Zero/binary indicates that transistors are assigned unit delay only to break feedback loops, and with Boolean optimization.

As a figure of merit, ratios are given of the number of gates (or DAG nodes) to the number of transistors in the circuit. Low (less than 1.0) values indicate that the program is able to abstract the circuit behavior. High numbers indicate cases

where a complex gate-level model is required to capture the subtleties of switch-level behavior. As this figures indicate, TRANALYZE consistently outperforms both ANAMOS and HLGCC, even when forced to generate models with exactly the same functionality.

The 74181 ALU is a direct mapping of a gate-level ALU into static CMOS gates. Both HLGCC and TRANALYZE successfully recognize all of the gates, except for the XORS. The Shift64 circuit is a 64-bit transmission gate shift register that can either shift or hold its data on each clock cycle. TRANALYZE can reduce each stage to just 3 four-input gates—comparable to a hand generated model. The DRAM circuit is an nMOS 3-transistor dynamic RAM. This represents a difficult case for gate-level model generation, since any gate-level implementation of a RAM is far more complex than what can be implemented using custom transistor logic. The SLAP circuit is a 16-bit CMOS processor designed at CMU. It contains many difficult structures for gate-level model extraction, including a register file implemented as a static RAM, a Manchester carry chain ALU, and a transmission gate shifter network. Even for these more difficult circuits, TRANALYZE is able to generate reasonably concise models.

Thus far, we have successfully simulated all of the benchmark circuits except for SLAP on SP. For the largest circuit simulated (DRAM256), SP operates 80 times faster than COSMOS executing on a SUN-4/110. Even greater speedups can be expected for larger circuits.

## Acknowledgements

Both S. Shimogori and M. Shoji of Fujitsu Laboratories have been instrumental in mapping the output of TRANALYZE onto SP, and in executing the benchmark simulations.

## References

- [1] D. T. Blaauw, P. Banerjee, and J. A. Abraham, “Automatic classification of node types in switch-level descriptions,” *ICCAD*, 1990.
- [2] D. T. Blaauw, D. G. Saab, J. Long, and J. A. Abraham, “Derivation of signal flow for switch-Level simulation,” *EDAC*, 1990, 301-305
- [3] D. T. Blaauw, D. G. Saab, P. Banerjee, and J. A. Abraham, “Functional abstraction of logic gates for switch-level simulation,” *EDAC*, 1991.
- [4] M. Boehner, “LOGEX—An automatic logic extractor from transistor to gate level for CMOS technology,” *25th DAC*, 1988, 517–522.
- [5] R. E. Bryant, *et al*, “COSMOS: a compiled simulator for MOS circuits,” *24th DAC*, 1987, 9–16.
- [6] R. E. Bryant, “Algorithmic aspects of symbolic switch network analysis,” *IEEE Trans. CAD/IC*, 1987, 618–633.
- [7] R. E. Bryant, “Boolean analysis of MOS circuits,” *IEEE Trans. CAD/IC*, 1987, 634–649.
- [8] F. Hirose, K. Takayama, and J. Niitsuma, “An event-driven logic simulation machine of computer systems,” *Proc. 1990 European Simulation Multiconference*, Nuremberg, Germany, June, 1990.

- [9] A. Jain, and R. E. Bryant, "Mapping switch-level simulation onto gate-level hardware accelerators," *28th DAC*, 1991.
- [10] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," *24th DAC*, 1987, 341–347.

# OPTIMIZATION OF CUSTOM MOS CIRCUITS BY TRANSISTOR SIZING

Andrew R. Conn<sup>†</sup>, Paula K. Coulman<sup>‡</sup>, Ruud A. Haring<sup>†</sup>, Gregory L. Morrill<sup>\*</sup> and Chandu Visweswariah<sup>†</sup>

<sup>†</sup>*IBM Thomas J. Watson Research Center, Yorktown Heights, NY*

<sup>‡</sup>*IBM Microelectronics Division, Austin, TX*

<sup>\*</sup>*IBM Microelectronics Division, Burlington, VT*

## Abstract

Optimization of a circuit by transistor sizing is often a slow, tedious and iterative manual process which relies on designer intuition. Circuit simulation is carried out in the inner loop of this tuning procedure. Automating the transistor sizing process is an important step towards being able to rapidly design high-performance, custom circuits. JiffyTune is a new circuit optimization tool that automates the tuning task. Delay, rise/fall time, area and power targets are accommodated. Each (weighted) target can be either a constraint or an objective function. Minimax optimization is supported. Transistors can be ratioed and similar structures grouped to ensure regular layouts. Bounds on transistor widths are supported.

JiffyTune uses LANCELOT, a large-scale nonlinear optimization package with an augmented Lagrangian formulation. Simple bounds are handled explicitly and trust region methods are applied to minimize a composite objective function. In the inner loop of the optimization, the fast circuit simulator SPECS is used to evaluate the circuit. SPECS is unique in its ability to efficiently provide time-domain sensitivities, thereby enabling gradient-based optimization. Both the adjoint and direct methods of sensitivity computation have been implemented in SPECS.

To assist the user, interfaces in the Cadence and SLED design systems have been constructed. These interfaces automate the specification of the optimization task, the running of the optimizer and the back-annotation of the results on to the circuit schematic.

JiffyTune has been used to tune over 100 circuits for a custom, high-performance microprocessor that makes use of dynamic logic circuits. Circuits with over 250 tunable transistors have been successfully optimized. Automatic circuit tuning has been found to facilitate design re-use. The designers' focus shifts from solving the optimization problem to specifying it correctly and completely. This paper describes the algorithms of JiffyTune, the environment in which it is used and presents a case study of the application of JiffyTune to individual circuits of the microprocessor.

## 1. Introduction, motivation and previous work

Designers often spend a lot of time manually sizing their schematics for area, delay and power, particularly in the context of custom designs. The tuning process is iterative, slow, tedious and error-prone, with circuit simulation in the inner loop. The updating of transistor widths from one iteration to the next relies on human intuition. Automating the circuit optimization process is an important

step towards rapidly designing high performance, custom circuits. Automatic circuit tuning has the additional benefit of facilitating design adaptation and re-use. Hence an automatic tuning (and retuning) capability is crucial to the productive design of custom circuits.

There have been many attempts to automate the transistor sizing problem. The first class of methods [1, 2] is based on static timing analysis [3] in which the circuit is assumed to consist of pre-characterized library cells. The delay of each cell is available as an analytic function of the sizes of the transistors in the cell. Total path delay is expressed as a function of the individual transistor widths and optimized. In particular, if the Elmore delay model [4, 5] is used, this overall delay is seen to be a posynomial function (a particular algebraic form) of the transistor widths. By a simple mapping of variables, the objective is converted to a convex function [1], and hence any minimum of the latter is guaranteed to be a global minimum. The advantages of static-timing-based methods include efficiency, ability to handle large designs and freedom from requiring input patterns to carry out the tuning. One of the problems with these methods is that they are not applicable to full-custom circuit designs, since static timing analyzers usually rely on pre-characterized library cells. Second, the accuracy of static timing analysis is limited (to about  $\pm 25\%$  in our experience) making it unsuitable as a basis for tuning high-performance custom circuits. Finally, static timing analysis is prone to the *false-path problem*, so the optimizer may be working hard to tune paths that are either irrelevant or can never be sensitized. Recently, power optimization has been proposed in this general framework [6]. Power is measured by probabilistic methods [7] and then approximated by a posynomial function. Simultaneous tuning of drivers and interconnect has been proposed in [8, 9].

Tuning based on *dynamic simulation* overcomes many of the above limitations of static tuning. The accuracy is as good as the simulator employed, false paths are not a problem and the method is applicable to any custom circuitry that the simulator can analyze. Appropriate input patterns must be provided by the user. These methods [10, 11] typically run SPICE in the inner loop to optimize such circuit performance functions as gain, area, delay and phase margin. However, using SPICE iteratively is computationally expensive and limits the size of the circuit that can be tuned. From an overall design perspective, we see static and dynamic methods complementing each other at different stages of the methodology, depending on the type of design.

In this paper, we present a method for tuning custom MOS circuits that uses dynamic simulation and *gradient-based optimization*. Our ability to compute gradients efficiently is crucial to the success of this approach. JiffyTune is a prototype implementation of our method. An overview of JiffyTune is presented in Section 2. JiffyTune uses SPECS [12, 13], a fast circuit simulator, to evaluate the circuit and provide function and gradient values. SPECS and the

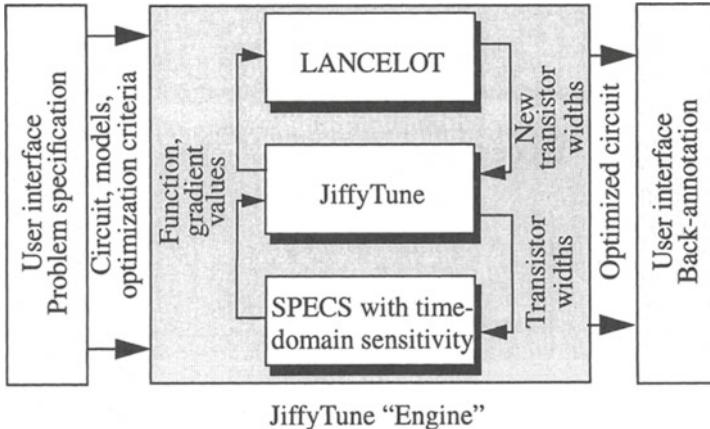


Figure 1. High-level view of JiffyTune.

computation of sensitivities are the topics of Section 3. The optimization engine used in JiffyTune is LANCELOT [14, 15, 16], a large-scale nonlinear programming package that handles simple bounds explicitly and accommodates general constraints with an augmented Lagrangian formulation. LANCELOT has been customized to the circuit tuning problem. The numerical methods involved in the nonlinear optimization are described in Section 4. To make the tuning environment productive and intuitive, interfaces have been built in two different design environments. Section 5 is devoted to the concepts guiding these interfaces and their benefits. JiffyTune has been used on many custom, dynamic-logic circuits of a high-performance microprocessor. A case study of the application of JiffyTune to this chip design, along with benchmarks, is presented in Section 6, followed by a section containing conclusions and future work.

## 2. Overview of JiffyTune

This section provides an overview of the various high-level software components of JiffyTune, as depicted in Figure 1. Subsequent sections contain detailed descriptions of the individual components.

The JiffyTune “engine” solves the following problem. Given a circuit schematic, input signals, a list of tunable transistors with initial widths and a set of circuit performance requirements, determine the optimal assignment of transistor

widths to tunable transistors in order to achieve the requirements. The user interface makes it convenient for the user to specify the problem, and visualize and accept the results of optimization.

## 2.1 JiffyTune

The JiffyTune block in Figure 1 performs the administrative portion of the tuning task. A control file grammar has been defined for the specification of circuit optimization problems. The control file contains the following information.

*Parameters:* This section contains a list of tunable transistors, their initial widths and bounds. Tunable transistors can be ratioed to other tunable transistors. Further, the user interface allows *grouping* of instances of similar structures so that they track each other during tuning. Thus, for example, the cells of an  $n$ -bit wide multiplexer can be grouped to ensure that the cells stay identical through the tuning process and thus lend themselves to a structured, regular layout.

*Measurements and functions:* A measurement is either a crossing-time, power or area measurement. In the absence of layout information, area is modeled by the sum of the tunable transistors' widths. Functions consist of any linear combination of measurements. Thus delays and rise/fall times are typically the difference of two crossing times. Each function has a weight, a target and a relation. A relation of "less than" implies that this function should be less than the target value. Similarly, relations of "greater than" and "equal to" are allowed. Alternately, a function can be "minimized" which means that the optimizer will try to decrease the value of this function as much as possible. Weights can be used to explore various trade-offs in tuning the circuit; they are especially required when functions of different quantities (area, delay, power) are being combined into a composite objective function.

Any number of functions can be grouped into a *minimax function*. A minimax function implies that the largest of some number of functions needs to be minimized or must meet a constraint. For example, the statement of the problem might be to minimize the delay of the worst of three paths through some combinational logic block.

*Controls:* This section provides administrative information like the maximum number of iterations, the layout grid for rounding transistor widths at the end of optimization and the location of the device model files.

JiffyTune reads the control file and internally represents the problem in a format that is understood by LANCELOT. JiffyTune also provides to LANCELOT a callable routine that will accept a set of transistor widths, perform a SPECS simulation, and return function and gradient values in the form required by LANCELOT. Then JiffyTune begins a LANCELOT optimization. At each iteration, JiffyTune keeps track of the best results so far. One of the main func-

tions of the JiffyTune block is to *chain rule and combine* gradients to provide to LANCELOT the gradients of various functions with respect to *independent variables only*. Typically, 25 to 30 iterations are required for convergence. The default maximum number of iterations in JiffyTune is 50. The recently implemented *slack updating method* described in Section 4.2 has led to fewer iterations being required in general.

## 2.2 SPECS

SPECS is a fast circuit simulator that uses simplified device models and event-driven techniques. JiffyTune calls SPECS in the inner loop to evaluate the circuit, and provide function and gradient values. SPECS and its sensitivity computation capabilities are described in Section 3.

## 2.3 LANCELOT

LANCELOT is a large-scale nonlinear optimization package that handles simple bounds and general constraints. JiffyTune provides the problem description and initial transistor sizes to LANCELOT. LANCELOT repeatedly calls SPECS with different transistor size settings, and builds a model of the “performance surface” of the circuit. It uses sophisticated nonlinear programming techniques to minimize the objective function. Details regarding LANCELOT and its application to the circuit tuning problem are provided in Section 4.

In addition to LANCELOT, the Levenberg-Marquardt [17] and Minos [18] optimization packages have been integrated into JiffyTune. The optimization testing environment described in [19] was used to integrate Minos into JiffyTune. The Levenberg-Marquardt method is limited since it only performs unconstrained optimization and is relatively unsophisticated. The Minos integration has been used only for comparisons and “sanity checks.”

## 2.4 The user interface

JiffyTune requires a knowledgeable user to carefully specify the optimization problem, and greedily takes advantage of any unspecified aspects. Further, the engine requires a control file that is difficult to create manually, particularly for large circuits. The user interface helps the user concentrate on the specification of the optimization problem by providing an intuitive interface and eliminating the tedium of dealing with a file-driven tool. It also provides facilities for back-annotation of the results of tuning. Section 5 is devoted to a discussion of the environment in which JiffyTune is used and the description of the interface.

### 3. SPECS and time-domain sensitivity computation

SPECS (Simulation Program for Electronic Circuits and Systems) is a fast circuit simulation program. SPECS is on average two orders of magnitude faster than AS/X, an internal traditional circuit analysis program [20] like SPICE. SPECS uses simplified device models and event-driven techniques to efficiently simulate MOSFET circuits in the time-domain, and has been used in production mode in various integrated circuit designs. JiffyTune uses SPECS to evaluate the circuit being optimized. However, this paper will not describe SPECS in any detail. The reader is referred to [12, 13, 21]. The device modeling assumptions in SPECS restrict its relative timing accuracy to  $\pm 5\%$ , and hence JiffyTune can only tune to within this accuracy limit.

#### 3.1 Sensitivity computation

SPECS uses simplified device models that consist of piecewise constant characteristics in multiple dimensions and grounded, linear capacitances. These simplifications allow efficient, incremental time-domain sensitivity computation [22, 23, 24]. Both the adjoint [25, 21] and direct [26] method have been implemented. In the direct method, branch constitutive relations (device characteristics) are *directly differentiated* with respect to the sensitivity parameter of interest. The circuit reflecting these differentiated equations, called the *sensitivity circuit*, has the same topology as the original circuit. Since SPECS uses piecewise constant device models, the sensitivity circuit consists of *disconnected capacitances* for large sub-intervals of time, with occasional *impulses of currents* flowing between these capacitances at times corresponding to events in the nominal simulation. Thus the solution of the sensitivity circuit is extremely efficient. In the direct method, the sensitivities of *all functions* with respect to *one parameter* are computed with a single solution of the sensitivity circuit.

In the adjoint method, elements are replaced by *adjoint equivalents* based on Tellegen's theorem [25, 21]. Again, the circuit is very simple and lends itself to efficient solution. In this case, however, *time is run backwards* in the adjoint circuit, and the waveforms of the adjoint circuit are *convolved* with those of the original circuit to obtain the required sensitivities. The gradients of *one function* with respect to *all parameters* are computed in a single solution of the adjoint circuit. Hence, when there are sufficiently more parameters than functions to justify the overhead of convolution, the adjoint method is advantageous.

Once the approximation in the simplified device models is accepted, the computation of gradients is exact. After the sensitivity circuit is solved in either method, gradients are chain-ruled and combined to obtain the sensitivity of each function with respect to *all the ramifications of variation of the tunable transistors' widths*. When the width of a transistor varies, its source and drain diffu-

sion capacitance and all the intrinsic MOSFET parasitic capacitances change. Each of these is submitted as an internal sensitivity parameter and then all the gradients are post processed and combined appropriately. The flavor of these computations is captured by the following simplified equation.

$$\begin{aligned} \frac{df}{dW} = & \frac{\partial f}{\partial W_{eff}} \frac{dW_{eff}}{dW} + \frac{\partial f}{\partial CD_{total}} \frac{dCD_{total}}{dW} \\ & + \frac{\partial f}{\partial CS_{total}} \frac{dCS_{total}}{dW} + \frac{\partial f}{\partial CG_{total}} \frac{dCG_{total}}{dW}, \end{aligned} \quad (1)$$

where  $f$  is the sensitivity function of interest,  $W$  is the transistor width (sensitivity parameter),  $W_{eff}$  is the effective width and  $CG_{total}$ ,  $CS_{total}$  and  $CD_{total}$  are the total parasitic capacitance at the gate, source and drain nodes, respectively. (0) is further expanded in terms of the device model parameters.

Voltage crossing sensitivities are expressed in terms of the nominal voltage waveform and transient sensitivity waveform of the appropriate signal, both sampled at the time of the voltage crossing of interest. In the case of the adjoint method, the transient sensitivity waveform is sampled by expressing the required value as a convolution integral and choosing to excite the adjoint circuit by an appropriate current source connected to that node.

### 3.2 Sensitivity benchmarks

The number of time-domain gradients computed during a typical JiffyTune run may be in the millions! Hence gradient computation must be extremely efficient to make this process feasible. A dynamic logic “branch scan” circuit with 144 MOSFETs was chosen to demonstrate the efficiency of gradient computation. The circuit was simulated in SPECS for a simulation interval of 27 ns. The CPU time for simulation was 2.05 s on an IBM Risc/System 6000 model 590. Then the same simulation run was carried out with 36 sensitivity functions (crossing times) and 104 MOS transistor widths as sensitivity parameters. Since there were 64 diffusion and other parasitic capacitances dependent on these 104 transistor widths, the total number of sensitivity parameters was 168. The number of gradients computed in this benchmark was 6,048, since SPECS finds the gradient of *every* sensitivity function with respect to *each* sensitivity parameter (our Jacobian matrix is dense). The number of sensitivities required was unusually large in this example, which was chosen to showcase the efficiency of gradient computation. The run times of SPECS with both the adjoint and direct method on this benchmark circuit are shown in Table 1. From the table, we see that the total run time for a JiffyTune iteration would be 24.94 s (assuming that the direct method were used). For comparison, the AS/X [20] run time on this circuit (with no gradient computation, of course) was 40.11 s. Hence, even on this modest example, JiffyTune can almost complete two iterations with

*gradient computation* in the time it takes AS/X to simulate the nominal circuit once.

Run time in CPU seconds	Adjoint method	Direct method
Total run time	32.38	24.94
Run time for sensitivity computation only	30.32	22.89
Run time per sensitivity circuit solution	0.84	0.14
Run time per sensitivity circuit solution as a fraction of simulation time (2.05 s)	40.78%	6.65%
Run time per gradient computation	5.01e-3	3.78e-3
Run time per gradient evaluation as a fraction of simulation time	0.24%	0.18%

Table 1. Sensitivity computation run time.

As can be seen from the table, the overhead of computing *one gradient* is a fraction of a percent of the original simulation time, which works out to 5 ms or less of CPU time in this example! The overhead of *one sensitivity circuit analysis* is about 7% for the direct method and about 40% for the adjoint method. Note that the number of runs in the adjoint method is equal to the number of functions, while it is equal to the number of sensitivity parameters in the direct method. The higher overhead in the adjoint method is accounted for by the convolution required between the waveforms of the original circuit and the sensitivity circuit. SPECS inspects the number of functions and the number of parameters and automatically makes a judgment, based on a simple heuristic, as to which method will be more efficient. The heuristic favors the adjoint method if the number of parameters exceeds the number of functions by a factor of 5. This heuristic appears to be effective in practice.

## 4. Nonlinear optimization in JiffyTune

### 4.1 LANCELOT

The optimization engine of JiffyTune is based upon the large-scale nonlinear programming package LANCELOT. The kernel algorithm is an adaptation of a trust region method to the general nonlinear optimization problem subject to simple bounds. The method is extended to accommodate general constraints by using an augmented Lagrangian formulation and the bounds are handled directly and explicitly via projections that are easy to compute.

In the context of *unconstrained optimization*, trust region methods, combining an intuitive framework with a powerful and elegant theoretical foundation, have led to robust numerical implementations. An excellent reference is [27]. The basic idea of trust region methods is to approximately minimize a model

of the objective function in a local neighborhood (called the *trust region*) centered at the current point. The objective function is modeled about the current point  $x^k \in \mathbb{R}^n$ , where  $k$  is the iteration count and  $x$  is the  $n$ -vector of variables. To minimize the model in the trust region, a step  $s^k$  is taken at iteration  $k$  to arrive at the point  $x^k + s^k$ . The function is evaluated at this point to determine how well the model predicted the actual change in the objective function. If good descent is obtained, the approximate minimizer is accepted as the next iterate ( $x^{k+1} \leftarrow x^k + s^k$ ) and the trust region is expanded. If moderate descent is obtained, the trust region size remains unchanged, but the step is accepted. Otherwise, no new point is accepted and the trust region is contracted. The beauty of such an approach is that, when the trust region is small enough and the problem smooth, the approximation is good, provided the gradients are sufficiently accurate. Moreover, assuming one does at least as well as the minimum along the steepest descent direction of the model within the trust region (which determines the so-called Cauchy point), one can ensure convergence to a stationary point. In addition, the trust region is eventually expanded so that it does not interfere with the subsequent iterates, and thus, assuming that in this situation the underlying algorithm is sufficiently sophisticated, one can ensure fast asymptotic convergence.

The extension of the above ideas to *problems with simple bounds* is relatively straightforward and is illustrated in Figure 2, for a quadratic model function and an  $l_\infty$  trust region. Essentially, one generalizes the Cauchy point ( $v_k$  in the figure) to the minimum along the *projected* gradient path ( $x^k - u - w$ ) within the trust region, where the projection is with respect to the bounds (either those provided by the user or implicit in the trust region). As in the previous case, global convergence can be guaranteed, provided one does at least as well as the generalized Cauchy point ( $w$ ). If a variable, as determined by the generalized Cauchy point, is at a bound it is said to be an *activity*. Unbounded variables are *free*. Activities are fixed temporarily, thus reducing the dimensionality of the search space (from two to one in the figure). Then, using only the free variables, the model of the objective function is further minimized within the feasible region and within the trust region ( $w$  is optimal in Figure 2). Thus one obtains better convergence, and ultimately, satisfactory asymptotic convergence. Updating of the trust region size and current point is handled in exactly the same way as it is in the unconstrained case.

It has been proved [14] that this method converges to a Kuhn-Tucker point [28]. Moreover, the correct active simple bounds are identified after a finite number of iterations assuming that strict complementarity is satisfied and the activities determined by the generalized Cauchy point are kept active during the rest of the iteration when the model is further reduced. Details are given in [14].

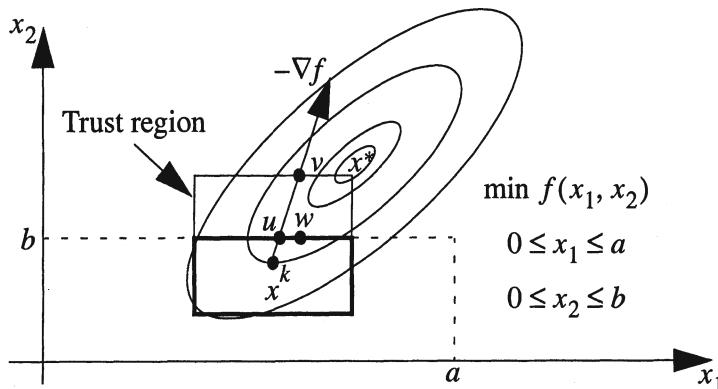


Figure 2. Illustration of generalized Cauchy point.

The extension to handle equality constraints is carried out by means of an augmented Lagrangian function

$$\Phi(x, \lambda, \mu) = f(x) + \sum_{i=1}^m \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i=1}^m c_i(x)^2. \quad (2)$$

$\Phi$  is minimized subject to the explicit bounds, using the earlier algorithm. Here  $f$  is the objective function,  $x$  the variables of the optimization,  $c_i(x)$  is an equality constraint with  $\lambda_i$  being the corresponding Lagrangian multiplier and  $\mu$  the penalty parameter used to dynamically weight feasibility. Inequality constraints are converted to equality constraints by first introducing slack or surplus variables, if necessary, and then formulating the augmented Lagrangian as before. This approach can be summarized as follows:

- 1 Test for convergence using the two following conditions. *Sufficient stationarity* – the projected gradient of the augmented Lagrangian with respect to the simple bounds is sufficiently small, and *sufficient feasibility* – the norm of the constraint violations is sufficiently small.
- 2 Use the simple bounds algorithm to find an approximate stationary point (minimizer) of  $\Phi$  subject to simple bounds.
- 3 If sufficiently feasible, update the multipliers  $\lambda_i$  and decrease the tolerances for stationarity and feasibility.

- 4 Otherwise, give more weight to feasibility (decrease  $\mu$ ) and reset tolerances for stationarity and feasibility.

It is possible to show, under suitable conditions, that convergence to a first-order stationary point for the nonlinear programming problem is attained. Further, if there is a *single limit point*, eventually the penalty parameter  $\mu$  is not reduced. Details of these and other theoretical properties are given in [15] and [29].

A significant cost in the optimization is solving a linear system of equations. Typically these arise from the necessity to determine an approximate stationary point for a quadratic function – equivalently, the necessity to solve a linear system whose coefficient matrix is symmetric. If the system is large, there are two approaches. The first is to use direct methods based on multi-frontal techniques (see Chapter 10 of [30]). Our experience to date, however, has been that an iterative approach using preconditioned conjugate gradients is more robust. All our reported numerical results with JiffyTune use this method. The appeal of conjugate-gradient methods for large-scale optimization is that they are particularly simple and only require that we store a few vectors. Moreover, they can be significantly accelerated by the use of preconditioners. Perhaps the best known conjugate basis for a convex quadratic form is the set of eigenvectors of the Hessian. The essential result is that at each iteration, the conjugate gradient method *minimizes the quadratic model* in the space spanned by the corresponding conjugate basis. If we can cluster eigenvalues (i.e., approximately have multiple eigenvalues) we can reduce the number of iterations for good approximations to minimizers from close to  $n$  to close to the number of clusters. The perfect way to do this in the quadratic case is to precondition with the Hessian inverse – but then this is equivalent to carrying out Newton’s method. Surprisingly, one can often do very well by using crude approximations to the Hessian (diagonal matrices, for instance). A good description of conjugate methods is given in [28], Sections 4.8.3 and 4.8.5. The LANCELOT package offers several preconditioners, and the Schnabel-Eskow preconditioner [31] is used in JiffyTune. A detailed reference on the LANCELOT package, including all the available options, is given in the book [16] that accompanies the original software.

## 4.2 Application of LANCELOT to JiffyTune

In the context of JiffyTune it was necessary to make certain modifications to LANCELOT to account for the fact that the function and gradient values from SPECS, although accurate to within small perturbations, are noisy. The introduced errors are small but significantly larger than machine precision. Because of the complexity of general nonlinear optimization, many initializations (such as the choice of the trust region radius or quadratic model) are based upon intelligent guesses, which cannot, of course, be ideal in all circumstances. In the worst case, for functions without noise, unfortunate choices can result in ineffi-

ciencies, but in the noisy case they can be insurmountable. A trivial example is if movement of less than 0.001 microns in transistor width is considered negligible, it may be disastrous if an automatic choice of the initial trust region size produces a radius that is much smaller. For similar reasons, we had to introduce looser tolerances for feasibility, line search discontinuities and bound activities, which are based upon machine precision in the original software. Finally, in order to stop gracefully and predictably we needed to consider step sizes beneath which further progress is unlikely and relate stopping criteria to this step size in a robust and consistent manner.

Two other enhancements deserve special mention. Slack/surplus variables corresponding to satisfied inequalities are updated at each iteration so that the corresponding equality is satisfied exactly, whenever such an update is consistent with the convergence theory; the result has been a reduction in the number of iterations to convergence.

Minimax optimization is handled by the introduction of an additional linear variable and reformulating the problem as a general nonlinear programming problem. For example, suppose one had the problem

$$\min_{x \in \mathcal{R}^n} \max_{i \in M \equiv \{1, 2, \dots, m\}} f_i(x). \quad (3)$$

This problem can be reformulated as

$$\begin{aligned} & \min_{z \in \mathcal{R}, x \in \mathcal{R}^n} && z \\ & \text{subject to} && z - f_i(x) \geq 0, \quad 1 \leq i \leq m. \end{aligned} \quad (4)$$

## 5. JiffyTune interface and environment

The JiffyTune engine as described above is driven by a textual control file that describes the optimization problem. Manual preparation and editing of such a file is tedious and error-prone. Also, the sophistication of LANCELOT and the choices of algorithms and tolerances thereof are not directly relevant to the end user. Thus, from the inception of the JiffyTune project, it was realized that a good human interface and an intuitive abstraction of its use and behavior would be crucial to acceptance of the tool by circuit designers. Interfaces were built to run the tool from the Cadence [32] and SLED [33] schematic design systems. The interface in the Cadence design environment was evolved simultaneously with the JiffyTune engine. Integrating the tool into such a framework capitalizes on the familiarity of the user with the schematic design environment, and lends a visual and interactive aspect to the tool. Many of the complexities are hidden from the designer, although care was taken to allow full access to all tool

functions, if the designer so requires. The basic functions of the interface are listed below.

*Specification of tuning parameters:* Tunable transistors are specified simply by selecting transistors or gates on a hierarchical schematic. The tunable transistors/gates are visually marked by a flag to indicate tunability. Facilities are provided to ratio transistors. Thus the two NFETs in a NAND gate can be forced to have the same width or adhere to a given tapering ratio. In addition, similar instances (transistors, gates or higher-level functional blocks) can be “grouped” together, to ensure that corresponding transistors in those blocks track during tuning.

*Specification of measurements and functions:* Presently, the interface supports delay, transition time (slew), area and power functions. For delay and transition times, net selection is done directly on the schematic. Power functions are specified by selecting the required voltage source, again directly on the schematic. In all cases, the user is prompted to provide a relation and target value as described in Section 2.1. In the schematic environment, with no knowledge of layout, area targets are approximated by the sum of the widths of the tunable transistors. The appropriate linear combination of measurements is written to the control file in each case. Minimax functions can be defined over any set of existing measurements.

*Specification of controls:* Administrative information such as the maximum number of iterations, file location of device models and layout grid for rounding transistor widths at the end of optimization can be specified in a form that is pre-filled with project-specific defaults.

*Execution of JiffyTune:* After specifying parameters, functions and controls, the designer can ask for all this information to be written to a control file, which can be inspected or edited if required. Then the designer can launch the JiffyTune engine, whereupon the progress of the optimization is displayed.

*Back-annotation of the results:* The results of a JiffyTune run are back-annotated onto the schematic as *suggested transistor widths* next to the transistors (or as new parameters next to gates). The designer can then accept these new widths/parameters, selectively or as a whole. Further, a facility is provided to back-annotate final waveform characteristics, such as delay through a gate or rise time of a net, directly onto the schematics, relieving the designer of the need to browse through simulation data using a waveform viewer.

*Utilities:* As a courtesy to the designer, the JiffyTune menu also includes facilities replicated from other areas of the schematic design environment, such as schematic checking, netlisting and automatically adjusting the number of fingers on each transistor, to create a single integrated tuning environment. Portability to various different sites and projects has been achieved by carefully separating the main code of the user interface from configurable site- and project-specific code.

Circuit requirements must be specified with care, since the optimizer will take advantage of any unspecified aspects. For example, area minimization will shrink to its minimum size a transistor that does not contribute materially to any measured transition. Thus, the tool enforces clear expression of circuit requirements that otherwise are often tacit. Since these circuit requirements and attributes logically belong with the circuit (they are indeed part of the intellectual effort of designing the circuit), the tuning parameters and functions are stored in the design database, either as instance properties (tunability, upper and lower bounds on transistor widths) or as schematic properties (grouping, functions). This practice also encourages the reuse of circuits; if a circuit has been adequately specified, it can easily be retuned.

## 6. Case study of JiffyTune use

JiffyTune was applied to tune custom circuits in the critical paths of a high-performance, dynamic-logic microprocessor. The circuits consisted of a mix of transistors and continuously parameterized gates. JiffyTune made it possible to refine the transistor sizes of circuits more quickly and thus rapidly respond to design changes late in the chip design cycle. Thus more flexibility was preserved in changing the specifications of circuits. The Cadence graphical user interface made it possible for designers to use the tool with little or no training.

JiffyTune was used by 41 designers during about 1,200 interactive sessions to tune 168 unique circuits. Over 2,200 successful JiffyTune runs were carried out, showing that some circuits were re-tuned multiple times. The results of tuning on one particular benchmark circuit are presented below.

Table 2 lists the results of running JiffyTune on a 12-way priority decode circuit under four different conditions. The circuit contains 70 MOSFETs and the simulation was run for 35 ns. The tuning runs all had 64 tunable transistors, of which 16 were independent and 48 dependent. The 17 functions to be optimized included the rising delay through four critical paths, the falling delay through those paths, the rise/fall times on each of the above 8 transitions and an area constraint. For confidentiality purposes, the delay requirement on the worst of the critical paths has been normalized to 500 time units in our report on this benchmark. The table lists the rising and falling delay of the four paths being tuned *as predicted by AS/X* on the final design (the worst of the 8 delays for each run is shown in bold), the total tunable transistor area of the circuit and the CPU time required to run JiffyTune on an IBM Risc/System 6000 model 590. The first JiffyTune run (HOT) started from a circuit that had previously been manually tuned (Manual). The worst delay through the circuit improved by 7.5% and the area decreased by 5.0%. The second JiffyTune run (COLD) started from an untuned circuit in which initial transistor sizes were set to the same default value as they would be for a "new design." Comparing the results

of (HOT) and (COLD) shows that the poor start point did not change the final results, but the optimizer had to work harder. The next JiffyTune run (DELAY) was set up to cause JiffyTune to reach the timing goal of 500 time units at all cost. JiffyTune was configured as in run (HOT), only with a weight on the area constraint that was a tenth of the previous value. The table shows that the goal was reached but at a high cost in transistor area. In general, we have found that it is important to impose an area constraint. Without an area constraint, JiffyTune converges to one of many equally fast circuits depending on the start point, with some solutions more efficient in area than others. The final run used the same start point and weights as HOT, but formulated the problem as a minimax optimization. A solution with a slightly higher delay but lower area was obtained in this case.

	Manual	HOT	COLD	DELAY	MINIMAX
Path #1, falling delay	555	494	488	483	497
Path #1, rising delay	471	475	473	469	510
Path #2, falling delay	535	495	495	483	506
Path #2, rising delay	494	488	488	472	524
Path #3, falling delay	<b>561</b>	<b>519</b>	517	<b>497</b>	<b>544</b>
Path #3, rising delay	497	<b>519</b>	<b>519</b>	<b>497</b>	527
Path #4, falling delay	497	494	491	484	516
Path #4, rising delay	462	497	496	485	476
Area	893	844	849	1148	800
# JiffyTune iterations	—	9	26	16	41
Run time (CPU s)	—	172	465	289	716

Table 2. JiffyTune results for 12-way priority decode circuit; all delays are normalized to a requirement of 500 time units.

JiffyTune in its present form is not directly applicable to designs in which gates are chosen from a fixed library of cells with a finite set of discrete power levels. JiffyTune performs well on hierarchical schematics with leaf cells containing any mix of transistors and continuously parameterized gates. In practice, JiffyTune handles circuits containing pass transistors well, in contrast to optimizers based on static timing analysis, since SPECS yields electrically true sensitivities taking into account details of the device model such as body effect. As new custom circuits are designed, JiffyTune will make it possible to speed up the design process, make more refined designs and provide better information about performance trade-offs.

## 7. Conclusions and future work

In this paper we described JiffyTune, a program that optimizes circuits by adjusting transistor sizes. JiffyTune makes use of fast simulation and time-domain

gradient computation in the circuit simulator SPECS, and advanced nonlinear numerical techniques in the optimization package LANCELOT. Delay, rise/fall time, area and power optimization have been implemented. The optimization system is flexible and allows ratioing of transistors and grouping of identical instances. An intuitive interface including back-annotation of optimization results on to the schematic has been developed.

The environment in which a circuit will be used and the required performance are estimated long before the chip is built. By the time the circuit is integrated onto the chip, it may no longer be optimally tuned, much to the frustration of the design engineer. Changes in loading, changes in the specifications, changes in parasitics after extraction, changes in technology device models and remapping to a new technology are common occurrences during the course of a project. In such situations, retuning at the push of a button without tedious re-specification is extremely useful.

JiffyTune has been successfully used to tune a number of circuits on the critical paths of a high-performance microprocessor chip which makes liberal use of dynamic logic. It has been particularly useful in tuning tricky pass-gate circuits and has been found to enhance design re-use. Further, since the optimization process has been made easy and automatic for the designer, a paradigm shift has been observed; the issue becomes how to correctly specify the optimization problem rather than solving the optimization problem itself.

There are a number of avenues for future work. "Event-driven convolution" is expected to speed up the computation of gradients by the adjoint method in SPECS. Repeated solution runs of the sensitivity or adjoint circuit are independent and therefore amenable to parallel processing. Occasionally, we encounter "non-working circuits" in the course of the optimization, when a transition to be measured does not occur; recovery from such situations is an interesting problem. Extension to semi-infinite constraints [10] would allow optimization of circuits while taking into account environment variations such as temperature and power supply voltage. Reformulating the problem to take advantage of *group partial separability* in LANCELOT [16, 34] would speed up the optimization. If the optimization could be formulated as a mixed integer/continuous problem, transistor ordering could be part of the optimization procedure. In addition, applications to IC manufacturability are being considered.

## 8. Acknowledgments

The authors would like to thank I. Elfadel, E. Chiprout, D. Brand and S. Nassif for their suggestions and careful reading of the manuscript.

## References

- [1] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *IEEE International Conference on Computer-Aided Design*, pp. 326–328, November 1985.
- [2] D. Marple, "Transistor size optimization in the Tailor layout system," *Proc. 1989 Design Automation Conference*, pp. 43–48, June 1989.
- [3] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, pp. 100–105, January 1982.
- [4] W. C. Elmore, "The transient analysis of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, 1948.
- [5] P. Penfield and J. Rubinstein, "Signal delay in RC tree networks," in *Proceedings of the 2nd Caltech VLSI Conference*, pp. 269–283, March 1981.
- [6] S. S. Sapatnekar and W. Chuang, "Power vs. delay in gate sizing: conflicting objectives?," *IEEE International Conference on Computer-Aided Design*, pp. 463–466, November 1995.
- [7] F. Najm, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE International Conference on Computer-Aided Design*, vol. CAD-9, pp. 439–450, April 1990.
- [8] J. J. Cong and C.-K. Koh, "Simultaneous driver and wire sizing for performance and power optimization," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 408–425, December 1994.
- [9] N. Menezes, R. Baldick, and L. T. Pileggi, "A sequential quadratic programming approach to concurrent gate and wire sizing," *IEEE International Conference on Computer-Aided Design*, pp. 144–151, November 1995.
- [10] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-7, pp. 501–519, April 1988.
- [11] J.-M. Shyu and A. Sangiovanni-Vincentelli, "ECSTASY: a new environment for IC design optimization," *IEEE International Conference on Computer-Aided Design*, pp. 484–487, November 1988.
- [12] C. Visweswarah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE International Conference on Computer-Aided Design*, pp. 248–251, November 1989.
- [13] C. Visweswarah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 861–870, July 1991.
- [14] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 433–460, 1988. See also same journal, pp. 764–767, volume 26, 1989.
- [15] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.
- [16] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, 1992.
- [17] J. J. Moré, "The Levenberg-Marquardt algorithm, implementation and theory," in *Numerical Analysis, Lecture Notes in mathematics 630* (G. A. Watson, ed.), Springer Verlag, 1977.
- [18] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 user's guide," Tech. Rep. SOL 83-20R, Stanford University, Stanford, CA, December 1983. Revised January 1987.

- [19] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "CUTE: constrained and unconstrained testing environment," *ACM Transactions on Mathematical Software*, vol. 21, pp. 123–160, March 1995.
- [20] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quassemzadeh, and T. R. Scott, "Algorithms for ASTAP - a network analysis program," *IEEE Transactions on Circuit Theory*, vol. CT, pp. 628–634, November 1973.
- [21] L. T. Pillage, R. A. Rohrer, and C. Visweswarah, *Electronic circuit and system simulation methods*. McGraw-Hill, 1995.
- [22] T. V. Nguyen, P. Feldmann, S. W. Director, and R. A. Rohrer, "SPECS simulation validation with efficient transient sensitivity computation," *IEEE International Conference on Computer-Aided Design*, pp. 252–255, November 1989.
- [23] P. Feldmann, T. V. Nguyen, S. W. Director, and R. A. Rohrer, "Sensitivity computation in piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 171–183, February 1991.
- [24] T. V. Nguyen, "Transient sensitivity computation and applications," Tech. Rep. CMUCAD-91-40, Carnegie Mellon University, Pittsburgh, PA, 1991.
- [25] S. W. Director and R. A. Rohrer, "The generalized adjoint network and network sensitivities," *IEEE Transactions on Circuit Theory*, vol. CT-16, pp. 318–323, August 1969.
- [26] D. A. Hocevar, P. Yang, T. N. Trick, and B. D. Epler, "Transient sensitivity computation for MOSFET circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-4, pp. 609–620, October 1985.
- [27] J. J. Moré, "Recent developments in algorithms and software for trust region methods," in *Mathematical programming: the state of the art* (A. Bachem, M. Grötschel, and B. Korte, eds.), (Berlin), pp. 258–287, Springer Verlag, 1983.
- [28] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*. London and New York: Academic Press, 1981.
- [29] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds," in *Proceedings of the 14th Biennial Numerical Analysis Conference Dundee 1991* (D. F. Griffiths and G. A. Watson, eds.), pp. 49–68, Longmans, 1992.
- [30] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices*. Oxford, U.K.: Clarendon Press, 1986.
- [31] R. B. Schnabel and E. Eskow, "A new modified cholesky factorization," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, pp. 1136–1158, 1991.
- [32] C. D. S. Inc., "Design entry: Composer users' guide 4.3," 1994.
- [33] M. Rubin, "View: A user tailorable interface for circuit design graphics," Tech. Rep. TR-19.90629, IBM, August 1990.
- [34] M. D. Matson and L. A. Glasser, "Macromodeling and optimization of digital MOS VLSI circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-5, pp. 659–678, October 1986.

## Part V

# PHYSICAL SIMULATION AND ANALYSIS

Highlights in Physical Simulation and Analysis at ICCAD <i>Kenneth S. Kundert and Jacob White</i>	367
Nonlinear Circuit Simulation in the Frequency-Domain ( <i>ICCAD 1985</i> ) <i>Kenneth S. Kundert and Alberto Sangiovanni-Vincentelli</i>	383
Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation ( <i>ICCAD 1989</i> ) <i>Peter R. O'Brien and Thomas L. Savarino</i>	393
Efficient Techniques for Inductance Extraction of Complex 3-D Geometries ( <i>ICCAD 1992</i> ) <i>M. Kamon, M. J. Tsuk, C. Smithhisler and J. White</i>	403
Time-Domain Non-Monte Carlo Noise Simulation for Nonlinear Dynamic Circuits with Arbitrary Excitations ( <i>ICCAD 1994</i> ) <i>Alper Demir, Edward W.Y. Liu and Alberto L. Sangiovanni-Vincentelli</i>	413
PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm ( <i>ICCAD 1997</i> ) <i>Altan Odabasioglu, Mustafa Celik and Lawrence T. Pileggi</i>	433
Circuit Noise Evaluation by Padé Approximation Based Model-Reduction Techniques ( <i>ICCAD 1997</i> ) <i>Peter Feldmann and Roland W. Freund</i>	451

# HIGHLIGHTS IN PHYSICAL SIMULATION AND ANALYSIS AT ICCAD

Kenneth S. Kundert

*Cadence Design Systems, Inc.*

*San Jose, CA*

Jacob White

*Massachusetts Institute of Technology*

*Cambridge, MA*

## **Abstract**

Six papers were chosen to represent twenty years of research in physical simulation and analysis, three papers addressing the problem of extracting and simulating interconnect effects and three papers describing techniques for simulating steady-state and noise behavior in RF circuits. In this commentary paper we will try to describe the contribution of each paper and place that contribution in some historical context.

## **1. Introduction**

Research in computer-aided design (CAD) is, by its nature, application-driven. Algorithmic and methodological innovation has almost always been a response to the changing challenges faced by designers. It should therefore come as no surprise that many of the best papers presented at ICCAD over its twenty year history are associated with design issues that emerged over the same time frame. For example, most of the physical simulation and analysis papers nominated for this commemorative proceedings addressed either interconnect effects or radio-frequency (RF) design, two topics that have grown enormously in importance over the last twenty years.

## **2. Inductance Extraction**

The first paper in our physical simulation and analysis commemorative collection is *Efficient Techniques for Inductance Extraction of Complex 3-D Geometries* by Matt Kamon, Michael Tsuk, Christopher Smithhisler, and Jacob White. It was published at ICCAD in 1992 [25] and can be found on page 403. This was the first public description of the *FastHenry* 3-D inductance extractor and its algorithms.

The *FastHenry* project started as an obvious follow-on to the *FastCap* project [42, 43]. *FastCap* combined the fast-multipole algorithm with iterative

Krylov-based linear solvers to provide an algorithm whose complexity increased linearly with the size of the problem. *FastHenry* was expected to be a relatively small project that applied the multipole and iterative methods of *FastCap* to the PEEC-based inductance algorithms of Ruehli [61]. The project was led by Matt Kamon, a student at MIT, and his advisor, Prof. Jacob White. The initial challenge was to determine which approach should be used to formulate the magnetic equations. Ruehli had already demonstrated the use of nodal analysis in his PEEC methods, but Kamon and White instead decided on going with mesh analysis. It proved to be a much better choice for several reasons, but perhaps most important is that it was more compatible with iterative solvers. Next, they applied GMRES, a Krylov-based iterative method, which appeared easy to do in concept, but was actually quite difficult in practice. It is vastly more difficult to construct preconditioners for inductance than for capacitance because of the long range interactions that are inherent to inductance. The initial preconditioner was based on the assumption that the couplings between the mesh currents within an individual conductor are much tighter than those that occur between conductors. It was this, along with the mesh current formulation, that was initially implemented in *FastHenry* and documented at ICCAD in 1992 [25]. In subsequent work they moved on to adapting the multipole methods to the magnetic problem while continuing to improve the preconditioner [26, 27, 28, 29].

The *FastHenry* program would eventually make a large contribution to the research community, particularly in the area of interconnect analysis. While not completely general, *FastHenry* was able to handle a wide variety of 3-D structures. It was not difficult to write an interface to and had sufficient capacity to solve very large problems by the standards of field solvers of the day. For example, if you were willing to wait you could use it to extract the inductance of relatively complex IC package lead frames. *FastHenry* had most of the physical effects that were needed, such as skin and proximity effects and provided an out-of-the-box solution for researchers interested in inductance. It worked, was reliable, was reasonably fast, was freely available, and it had little to nothing in the way of competition. As such, it quickly became a critical tool for those exploring the effect of inductance on delays in interconnect. It was used by researchers to explore and understand inductance in complex geometries, and was a golden reference for anyone developing models of interconnect that include inductance.

The impact of *FastHenry* was different in character from its forerunner, *FastCap*. *FastCap* was the first solver whose computational complexity did not increase super-linearly with the size of the problem, and so was the first truly high-capacity solver. However, capacitance is fairly easy to understand and it is easy to write solvers using a variety of methods that address the capacitance problem. As such, *FastCap* quickly had many competitors. The contribution of

*FastCap* lay mainly in the ideas it embodied and the fact that it started a wave of innovation in solver technology. Inductance was much trickier, especially in 3-D. It was exceptionally difficult to write a solver that was both reliable and efficient, and there were few choices of algorithms. And so while the ideas and algorithms were less ground-breaking, the use of the program itself resulted in a deeper understanding of inductance by the user community and a wave of innovation in interconnect analysis. A remarkably large number of the papers on inductance that followed have used *FastHenry* as the benchmark reference [31, 32, 20, 64, 2].

### 3. Including Interconnect in Timing Analysis

Although inductive effects have been important in integrated circuit packaging for some time, it has only recently become an on-chip concern for designers of high-performance digital circuits. However, interconnect resistance has been a design issue for more than two decades. Interconnect resistance was a problem in the early 1980's primarily because the then available fabrication technology provided only one layer of metal, making it necessary to use less conductive polysilicon for interconnect [40]. Technology improvements soon provided multiple layers of metal, thereby eliminating the need to use polysilicon for interconnect, but these same improvements also allowed for much higher circuit densities. To achieve these high circuit densities, it was necessary to reduce the cross-sectional area of metal interconnect, but since some signals still crossed the entire integrated circuit, maximum interconnect lengths did not scale like cross-section areas. The result was a resistance-preserving transition from fat interconnect in poor conducting polysilicon to skinny interconnect in good conducting aluminum to even skinnier interconnect in better conducting copper.

During the 1980's, one approach dominated for estimating the delay from the output of a driving logical gate, through the resistive interconnect, to the possibly many receiving gate inputs. The approach involved generating a model resistor-capacitor (RC) network, and then approximately analyzing the model [70]. In the model, the driving gate became a voltage source in series with a resistor, the interconnect became a collection of series resistors and grounded capacitors, and the receiving gates became grounded capacitors. The required delays were then estimated by approximating the step response of the model RC network, usually by exploiting the fact that the resistor network was almost always a tree and then using a recursive algorithm to walk the tree and efficiently compute Elmore delays [60, 38, 17].

In present-day timing analysis programs, the techniques used to compute delays depend on the design methodology. In analyzing full-custom designs, delays are often estimated by numerically integrating the differential equations associated with the network formed from the driver and receiver transistors and

the interconnect resistors and capacitors. For designs based on cell libraries, the numerical integration approach may not be the most efficient and can even be infeasible; the designer may not have access to transistor-level descriptions of the cells. Instead, delays in cell-based designs are analyzed using a strategy similar to the twenty-year old RC model strategy described above. The two main differences between the current strategy and that of twenty years ago are associated with determining the voltage source plus series resistor model for the driver gate, and with analyzing the resulting RC network.

The most widely used approach for generating voltage source plus series resistor models for cell library outputs is described in [8]. This approach presumes that the parameters of the simplified model must depend on the true interconnect load impedance and not just on total capacitance. This concept was first presented in the second paper in our physical simulation and analysis commemorative collection, *Modeling the Driving-Point Characteristics of Resistive Interconnect for Accurate Delay Estimation* by Peter R. O'Brien and Thomas L. Savarino, which can be found on page 393. The main contribution of O'Brien and Savarino's paper was to establish that the behavior of a driving gate is substantially modified when much of the interconnect capacitance is "screened" from the gate output by the interconnect resistance.

O'Brien and Savarino's paper had an additional contribution that is more generic. In the years before their paper appeared, many researchers were using moment-matching methods, but most of those papers were matching only the first order moments to estimate delays [60, 38, 24]. O'Brien and Savarino's paper was one of the earliest to suggest that matching progressively higher order moments, which is equivalent to matching progressively higher terms in the Taylor series expansion of a transfer function, could be used to estimate input impedance. In addition, they suggested the idea that moment-matching could be used more generally to generate a reduced order model of the interconnect. In particular, they used moment-matching to determine values for the resistor and two grounded capacitors in a  $\pi$  circuit, and then suggested that the generated circuit was a reduced model of the interconnect. As will be discussed in the next section, general model order reduction has now become a central part of strategies for handling interconnect, and O'Brien and Savarino's early observation about the subject was years ahead of its time.

#### 4. Model Order Reduction for Interconnect

In order to examine signal propagation and coupling effects due to on- or off-chip interconnect, it is usually necessary to couple an electromagnetic analysis of the three-dimensional interconnect with a circuit-level analysis of the transistors connected to that interconnect. Although there are techniques and commercial tools that couple time-domain electromagnetic simulation programs directly

to circuit simulators [67], these approaches are too computationally expensive to use in any but the simplest of scenarios. Instead, layout extraction tools are combined with model-order reduction techniques to generate low-order models for the interconnect, and then these low-order models are included with the transistors in circuit- or timing-level simulation and analysis.

When including interconnect effects in timing verification of an entire digital integrated circuit, the extraction-plus-reduction strategy must be fast enough to analyze millions of interconnect lines, but the accuracy requirements are modest. The commonly used approach is first to subdivide the interconnect into a large number of small sections, and then to apply a formula- or pattern-based algorithm to convert each of those small sections into resistors, capacitors and inductors. For complicated interconnect geometries, there can be a very large number of small sections, in which case the resulting extracted circuit will have a very large number of elements. Model reduction is used to generate low-order models of those large circuits, while still preserving the large circuit input-output behavior [37].

A very different extraction-plus-reduction strategy is used when examining interconnect coupling effects in packaging or for analog circuits. The objective of the coupling analysis is to determine if too much noise will be injected into a victim signal due to the proximity of the victim signal's interconnect to the interconnect of simultaneously active aggressor signals. In order to capture the "ganging-up" effect of many simultaneously active aggressors, it is necessary to extract interconnect coupling terms that would be too small to be of concern in timing analysis; but there is no need to extract more than a few hundred interconnect lines. The modest speed and high accuracy needed to investigate packaging and analog circuit coupling problems has led to extraction-plus-reduction strategies in which model reduction is embedded in a three-dimensional field solver, and the combination directly produces reduced models [63, 5, 50].

Although the approach to extraction is very different in digital circuit timing analysis versus packaging and analog circuit coupling examination, the approach to model reduction is very similar. In both cases the interconnect is viewed as a linear multi terminal device. The device terminals are usually transistor-interconnect interface locations, but may also simply be convenient separation points. Regardless of application, the goal of model reduction is to construct a representation of the interconnect that is inexpensive to evaluate, yet still accurately represents terminal behavior. Finally, the form of the reduced model should be appropriate for circuit or timing simulation. The reduced model could be another circuit, as in [46, 44], or a state-space model, but not tables of frequency-domain data.

The classic approach to model reduction is to select a fixed circuit topology, like the  $\pi$  model in [46], and then use some kind of fitting procedure to determine the element parameters. Such fitting techniques have uncertain ac-

curacy, and there is no way to increase the that accuracy. In the early 1980's, researchers in system theory developed methods for reducing the order of state-space models that were, in a carefully chosen metric, provably optimal in preserving input-output behavior [22]. Although these optimal methods produced excellent reduced models, the computational cost of the numerical algorithms used in the reduction grew cubically with the number of states in the *unreduced* model, making the method much too slow to use on large interconnect problems.

In [52], an algorithm was presented for generating low-order Padé approximates [1] to circuit transfer functions. The algorithm was reasonably efficient even for large circuits, and it was expected that increased accuracy could be achieved by increasing the order of the Padé approximate. Also, since a  $q^{\text{th}}$  order Padé approximate matches  $2q - 1$  moments of the original transfer function, the method could be viewed as a generalization of the moment techniques used in interconnect delay estimation. Early implementations of these Padé-based methods were sometimes unreliable when applied to interconnect problems, and occasionally generated inaccurate and/or unstable representations of the interconnect. Since a  $q^{\text{th}}$  order Padé approximate matches  $2q - 1$  terms in the *zero-frequency centered* Taylor series expansion of the original frequency response, accuracy problems were mitigated by generalizing the approximate to match Taylor expansions at multiple center frequencies [7].

Using multiple center frequencies improved the robustness of the methods in [52, 3], but the fundamental difficulty was not made clear until the seminal paper by Feldmann and Freund [18] published at ICCAD 1995 (though less well known, [21] appeared nearly simultaneously). In [18], it was shown that many of the accuracy issues associated with the Padé approximates stemmed from the numerically unstable way in which those approximates were being computed. In addition, it was shown that the Padé approximate for a transfer function could be computed in a numerically stable manner by starting with a state-space description, and then constructing bi-orthogonalized Krylov subspaces associated with the system matrix, its transpose, the input vector and the output vector.

Once the connection was made between moment-matching and the Krylov subspaces, results using Krylov subspaces for model reduction appeared rapidly. Block full orthogonalization and bi-orthogonalization methods were developed to directly generate state-space representations of multiple-input multiple-output reduced models [63, 19], methods were generated with guaranteed stability properties [62], techniques were developed that allowed multiple expansion centers, and Krylov-subspace reduction was coupled to fast electromagnetic solvers [63, 50]. These algorithmic variants were then organized into a unified theory of Krylov subspace reduction methods, the projection framework, which only appeared in a unique Ph. D. thesis that is required reading in the field [23].

In the coupled circuit-interconnect simulation scenarios described above, it is certain that many reduced-order models will be combined to describe a larger system, and therefore the reduced-order models must be passive. Stability is insufficient because the combination of stable systems is not necessarily stable, but the combination of passive systems is passive. The issue of passivity in reduced-order models first appeared in a widely circulated but still unpublished manuscript [4]. In that manuscript, an algorithm was given for generating guaranteed passive reduced models of RC circuits, and that algorithm was connected to the notion of congruence transforms in [30]. The race was then on to find a guaranteed passive reduction strategy for RLC circuits, and the third paper in our physical simulation and analysis commemorative collection is the winner of that race *PRIMA: A Passive Reduced-order Interconnect Macromodeling Algorithm*, by Altan Odabasioglu, Mustafa Celik, and Lawrence Pileggi (see page 433).

The PRIMA algorithm was more than just first, it had an elegant simplicity that stemmed from combining two key steps. First, the reduction was applied to a circuit equation formulation that generated two positive semi-definite matrices, one operating on the state vector and one operating on the state vector's derivative. Then, each of the positive semi-definite matrices were reduced with the same congruence transform. This two-step approach was influential because it became a strategy for generating passivity-preserving methods with other desirable properties, such as matching multiple moments [16] or having simple update formulas [37]. The two-step approach was also used to adapt the PRIMA algorithm to other applications, such as extraction from 3-D electromagnetic analysis [39].

## 5. Harmonic Balance

The fourth paper in our commemorative collection is *Nonlinear Circuit Simulation in the Frequency Domain* by Kundert and Sangiovanni-Vincentelli [33] and can be found on page 383. This paper and the follow-on journal paper [34] were not the first papers on harmonic balance. However, they were the first to attempt to apply harmonic balance to large scale circuits, and as such directly led to the introduction of the commercial RF simulators that are so heavily used today. Previous attempts focused on microwave circuits that contained a very small number of transistors, one or at most two, and a large number of passive components. This made sense for discrete microwave circuits, but was not appropriate for the monolithic microwave integrated circuits (MMICs) of the day, nor would it be appropriate for the coming radio frequency integrated circuits (RFICs). Engineers had been successfully designing discrete microwave circuits for years by iterating prototypes, and there was not a strong need for nonlinear circuit simulators from this community. However, for MMIC designers, iterat-

ing prototypes was both very expensive and time consuming, so they tended to use SPICE to verify their circuits before fabrication.

There were several problems with using SPICE for microwave circuits [35]. It is a time-domain simulator and so has difficulty including models of distributed components such as transmission line structures, particularly if they include loss or dispersion. It is a transient-based simulator, and so is inefficient when high frequency carrier signals are combined with low-frequency modulation signals. And, it is incapable of predicting the noise of circuits such as mixers and oscillators. *Nonlinear Circuit Simulation in the Frequency Domain* did not really address any of these issues directly, but rather it showed how harmonic balance, which was known to be an answer to the first problem, could be extended so that it had the capacity to be applied to integrated circuits. It was the ability to simulate integrated circuits that gave harmonic balance its commercial appeal. It was left to follow-on work to address the remaining issues.

The increase in capacity was achieved by reducing the computational cost of factoring the frequency-domain harmonic balance Jacobian by using a more easily factored near block diagonal matrix, one which ignored some of the coupling between frequencies. This represents the second generation harmonic balance. Many years later there was another substantial increase in the capacity of harmonic balance when it was combined with fast Krylov subspace based methods [41]. These methods, which are now pervasive, also require an approximate Jacobian to act as a preconditioner. The approach pioneered by Kundert and Sangiovanni-Vincentelli continues to live on as one of a few commonly used preconditioners for this third generation of harmonic balance simulators.

*Nonlinear Circuit Simulation in the Frequency Domain* also reported on the development of *Harmonica*, a harmonic balance simulator that was to be very influential; spawning several of today's leading simulators. The name was later changed to *Spectre* when the *Harmonica* name was co-opted by Compact Software for the name of its independently developed harmonic balance simulator. *Spectre* was successful, at least in part, because it followed a formula pioneered by SPICE. *Spectre* had a SPICE-like netlist and use-model, it placed no artificial limits on the number of nodes or components in the circuit, and most importantly, its source code was made freely available to anyone willing to pay a nominal fee. The idea that Berkeley should simply give away the source code to its simulators was first championed by Don Pederson, and is credited with much of the broad success of SPICE [49]. This allowed Berkeley *Spectre* to become the foundation of what are currently the two leading RF simulators, Agilent's *ADS* and Cadence's *SpectreRF*, and a leading integrated circuit simulator, Cadence's *Spectre*. It is interesting to note that even though Cadence's simulators are direct descendants of Berkeley's *Spectre*, neither of them currently uses harmonic balance. Rather, *Spectre* is a SPICE-class simulator and *SpectreRF* uses

shooting algorithms that were developed as part of follow-on research to the original harmonic balance work.

In another interesting aside, this paper started a spirited competition between the CAD community and the microwave community [56, 55]. Each produced papers at a relatively rapid pace on variations of, and extensions to, harmonic balance in an attempt to lay first claim to the resulting innovations. Even competing simulators were produced. While both communities had their successes, in the end it was the CAD community's deep knowledge of numerical algorithms and large-scale programming techniques that produced the biggest advances and ultimate success. In particular, it was the CAD community that substantially advanced the capabilities of RF simulators by enhancing shooting methods [65], introducing Krylov methods, and developing efficient methods for time-varying noise analysis [66, 58, 14].

## 6. Noise Analysis

In 1971, Rohrer et al [57] established the noise analysis method that eventually made its way into SPICE [45] and became the de facto standard for three decades. This method was accurate, robust, and efficient, but was limited in the type of circuits it could handle. With their approach the circuit was first linearized about the DC operating point. The noise analysis is performed on the resulting linear time-invariant (LTI) representation by computing the frequency dependent transfer functions from noise source to the output of interest. Their key insight, still important nearly thirty years later, is that for typical circuits there are many noise sources but noise is measured at only a few outputs. This many-input few-output case is much more efficiently analyzed using an adjoint formulation.

Dramatically faster computers now allow designers to routinely simulate much more complicated analog circuits, with commensurately complicated noise spectra, possibly with multiple sharp peaks and nulls. In the approach in [57], the frequency-dependent output-to-noise conjugate transfer functions are computed by solving the transpose of the linearized circuit equation for each frequency of interest. In order to capture sharp features in the noise spectra, it would be necessary solve at a large number of frequencies. The fifth paper in our commemorative collection, *Circuit noise evaluation by Padé approximation based model-reduction techniques* by Peter Feldmann and Roland Freund (see page 451), cleverly applies their well-known Padé-via-Lanczos algorithm for model reduction [18] to the problem of avoiding the multiple frequency solves in noise analysis. In this approach, a rational function description for the noise spectra is computed directly, and therefore sharp spectral features are easily captured. Extensions of this method can be used for computing noise in more complicated time-varying cases described below [59].

Analyzing the circuit about the DC operating point is adequate for simple circuits such as amplifiers and passive circuits such as filters, but cannot be used on circuits for which the noise performance is strongly affected by large signals, such as with mixers, oscillators, samplers, switched-capacitor filters, and the like. The large, generally periodic, signals present in these circuits act to modulate both the noise sources and the transfer characteristics of the circuit from the noise sources to the output. The result is cyclostationary noise, or noise with time-varying statistics, at the output of the circuit. With the recent rise of importance of RF circuits, a more general type of noise analysis became essential. What was needed was the ability to perform a noise analysis not about the DC operating point, but about a time-varying operating point.

Okumura et al [48] extended Rohrer's method to support prediction of noise of circuits linearized about a periodic operating point. These ideas were later commercialized by Telichevesky [66] and others. In this way the noise of circuits such as mixers and switched-capacitor filters could be predicted accurately. However, there were still a great number of fundamental questions left unanswered, particularly regarding oscillator phase noise.

*Time-Domain non-Monte Carlo Noise Simulations for Nonlinear Dynamic Circuits with Arbitrary Excitations* was published at ICCAD in 1994 by Alper Demir, Edward Liu, and Alberto Sangiovanni-Vincentelli. It is the last physical simulation and analysis paper highlighted in this commemorative collection and can be found on page 413. In a break from EDA tradition, Demir et al proposed to perform noise analysis by formulating and solving the stochastic differential equations for the circuit [10, 11]. The approach offered a unique advantage in that it did not require a periodic operating point, or even that the circuit be in steady state. However, his method was also perceived as being complex and was computationally expensive for large circuits. The added generality of the method was not seen as a compelling advantage in light of the disadvantages, so the method he proposed has not seen much use. Nevertheless, in these papers, Demir et al were the first in the EDA field to advocate a more rigorous approach to noise analysis. They were also the first to introduce to the design community the theoretical foundation that would be needed to address the difficult questions of which they were just becoming aware.

These papers were just the first of several by Demir that used stochastic differential equations to deeply explore questions of noise, particularly oscillator phase noise [12, 13, 14, 15]. This flurry of results led, either directly or indirectly, to papers by many others that together advanced the fundamental understanding of noise and improved both the analysis [58, 9, 68, 51, 69] and design of low-noise circuits [36, 53].

## 7. Conclusions and Acknowledgments

The area of physical simulation and analysis has been enlivened in recent years by the growing importance of problems associated with RF design and interconnect effects. Since many of the key papers in these fields did not initially appear at ICCAD, and therefore were not considered for this commemorative collection, we hope that this commentary both celebrates the selected papers and recognizes the importance of contributions that have appeared elsewhere.

It is not clear what emerging problems will provide the stimulation to open up new directions in this area, but given how often the CAD community has incorrectly predicted that physical simulation research has plateaued, the authors would like to wait and see.

We would like to thank Joel Phillips for his insightful technical comments while evolving this commentary, and for providing some first-hand historical information. In addition, we would like to thank the many contributors to physical simulation and analysis both at ICCAD and elsewhere.

## References

- [1] George A. Baker, Jr. and Peter Graves-Morris, *Padé Approximants, 2nd Edition* Cambridge University Press, 199
- [2] M. Beattie, B. Krauter, L. Alatan, and L. Pileggi. Equipotential shells for efficient inductance extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. vol. 20, no. 1, January, 2001, pp. 70-79.
- [3] J. E. Bracken, V. Raghavan, and R. A. Rohrer. Interconnect simulation with asymptotic waveform evaluation. *IEEE Trans. Circuits Syst.*, 39(11):869-878, November 1992.
- [4] J. E. Bracken. Passive modeling of linear interconnect networks. *Widely circulated notes*, 1995.
- [5] A. C. Cangellaris and L. Zhao. Passive reduced-order modeling of electromagnetic systems. *Computer Methods in Applied Mechanics and Engineering* vol. 169, no. 3-4, February 1999, pp. 345-358.
- [6] P. K. Chan. An extension of Elmore's delay. *IEEE Trans. on Circuits and Systems*, CAS-33(11):1147-1149, 1984.
- [7] Eli Chiprout and Michel S. Nakhla. Analysis of interconnect networks using complex frequency hopping (CFH). *IEEE Trans. CAD*, 14:186-200, February 1995.
- [8] F. Dartu, N. Menezes, and L. Pileggi. Performance computation for precharacterized CMOS gates with RC loads, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 5, May 1996.
- [9] A. Dec, L. Toth, and K. Suyama. Noise analysis of a class of oscillators. *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 6, pp. 757-760, June 1998.
- [10] Alper Demir, Edward W. Y. Liu, and Alberto L. Sangiovanni-Vincentelli. Time-domain non-Monte Carlo noise simulation for nonlinear dynamic circuits with arbitrary excitations. *IEEE/ACM International Conference on Computer-Aided Design*, 1994, pp. 598-603.

- [11] A. Demir, E. Liu, and A. Sangiovanni-Vincentelli. Time-domain non Monte-Carlo noise simulation for nonlinear dynamic circuits with arbitrary excitations. *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 493-505, May 1996.
- [12] A. Demir, A. Mehrotra, and J. Roychowdhury. Phase noise and timing jitter in oscillators. *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (CICC-1998)*, pp. 45-48.
- [13] A. Demir. Phase noise in oscillators: DAEs and colored noise sources. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, ICCAD-1998, pp. 170-177.
- [14] A. Demir, A. Mehrotra, and J. Roychowdhury. Phase noise in oscillators: a unifying theory and numerical methods for characterization. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 5, May 2000, pp. 655-674.
- [15] A. Demir, D. Long, and J. Roychowdhury. Computing phase noise eigenfunctions directly from steady-state Jacobian matrices. *IEEE/ACM International Conference on Computer Aided Design, 2000* (ICCAD-2000). pp. 283-288.
- [16] Ibrahim M. Elfadel and D. D. Ling, A block rational Arnoldi algorithm for multipoint passive model-order reduction of multiport RLC networks, *International Conference on Computer Aided-Design*, San Jose, California, November 1997.
- [17] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55-63, January 1948.
- [18] P. Feldmann and R. W. Freund. Efficient linear circuit analysis by Padé approximation via the Lanczos process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14:639-649, 1995.
- [19] Peter Feldmann and Roland W. Freund. Reduced-order modeling of large linear subcircuits via a block Lanczos algorithm. In *32<sup>nd</sup> ACM/IEEE Design Automation Conference*, pp. 474-479, San Francisco, California, June 1995.
- [20] K. Gala, V. Zolotov, R. Panda, B. Young, J. Wang, and D. Blaauw. On-chip inductance modeling and analysis. *2000 IEEE/ACM Design Automation Conference*, pp. 63-68.
- [21] K. Gallivan, E. Grimme, and P. Van Dooren. Asymptotic waveform evaluation via a Lanczos method. *Applied Mathematics Letters*, 7(5):75-80, 1994.
- [22] K. Glover. All optimal Hankel-norm approximations of linear multivariable systems and their  $L^\infty$ -error bounds. *Int. J. Control*, vol.39, No.6, pp.1115-1193, 1984.
- [23] Eric Grimme. *Krylov Projection Methods for Model Reduction*. PhD thesis, Coordinated-Science Laboratory, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, 1997.
- [24] M.A. Horowitz. Timing Models for MOS Circuits. PhD thesis, Stanford University, January 1984.
- [25] M. Kamon, M. J. Tsuk, C. Smithhisler, and J. White. Efficient techniques for inductance extraction of complex 3-D geometries. *IEEE/ACM International Conference on Computer-Aided Design Digest of Technical Papers 1992* (ICCAD-92), pp. 438-442.
- [26] M. Kamon and J. White. Preconditioning for multipole-accelerated 3-D inductance extraction. *Electrical Performance of Electronic Packaging*, 1993, pp. 189-192.
- [27] M. Kamon and J. R. Phillips. Preconditioning techniques for constrained vector potential integral equations, with application to 3-D magnetoquasistatic analysis of electronic packages. *Proceedings on the Colorado Conference on Iterative Methods*, Breckenridge, CO, April 1994.

- [28] M. Kamon, M. J. Tsuk, and J. K. White. FastHenry: a multipole-accelerated 3-D inductance extraction program. *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 9, part 1 of 2, Sept. 1994, pp. 1750-1758.
- [29] M. Kamon, B. Krauter, J. Phillips, L. Pileggi, J. White. Two optimizations to accelerate method-of-moments algorithms for signal integrity analysis of complicated 3-D packages. *Electrical Performance of Electronic Packaging*, 1995, pp. 213 -216.
- [30] K. J. Kerns, I. L. Wemple, and A. T. Yang. Stable and efficient reduction of substrate model networks using congruence transforms. In *IEEE/ACM International Conference on Computer Aided Design*, pp. 207-214, San Jose, CA, November 1995.
- [31] B. Krauter, L. T. Pileggi. Generating sparse partial inductance matrices with guaranteed stability. *1995 IEEE/ACM International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 45-52.
- [32] B. Krauter and S. Mehrotra. Layout based frequency dependent inductance and resistance extraction for on-chip interconnect timing analysis. *Proceedings of the 1998 Design Automation Conference*, pp. 303-308.
- [33] Kenneth S. Kundert and Alberto Sangiovanni-Vincentelli. Nonlinear circuit simulation in the frequency domain. *IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp 240-242, November 1985.
- [34] Kenneth S. Kundert and Alberto Sangiovanni-Vincentelli. Simulation of nonlinear circuits in the frequency domain. *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, pp. 521-535, Oct. 1986.
- [35] Ken Kundert. Introduction to RF Simulation and its Application. *IEEE Journal of Solid-State Circuits*, vol. 34, no. 9, September 1999.
- [36] T. H. Lee and A. Hajimiri. Oscillator phase noise: a tutorial. *IEEE Journal of Solid-State Circuits*, vol. 35. No. 3. pp. 655-674, March 2000.
- [37] H. Levy, D. MacMillen, and J. White, A rank-one update method for efficient processing of interconnect parasitics in timing analysis, *Proceedings of the Design Automation Conference*, Los Angeles, June, 2000, pp 75-7
- [38] T.-M. Lin and C.A. Mead. Signal delay in general RC networks. *IEEE Trans. on Computer Aided Design*, CAD- 3:331-349, 1984.
- [39] N. Marques, M. Kamon, J. White, L. M. Silveira, A mixed nodal-mesh formulation for efficient extraction and passive reduced-order modeling of 3D interconnects, *Proceedings of the 35th Design Automation Conference*, San Francisco, June, 1998, pp. 297-302.
- [40] C. A. Mead and L. Conway. *An Introduction to VLSI Systems*. Addison Wesley, 1980.
- [41] R. Melville, P. Feldmann and J. Roychowdhury. Efficient multi-tone distortion analysis of analog integrated circuits. *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1995.
- [42] K. Nabors and J. White. A fast multipole algorithm for capacitance extraction of complex 3-D geometries. *Proceedings of the 1989 IEEE Custom Integrated Circuits Conference*, pp. 21.7/1-21.7/4.
- [43] K. Nabors and J. White. FastCap: a multipole accelerated 3-D capacitance extraction program. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 11, Nov. 1991, pp. 1447-1459.
- [44] K. Nabors, T.-T. Fang, H.-W. Chang, K. Kundert, and J. White. A Gaussian-quadrature based algorithm for RLC-line to RLC-line reduction, *Proceedings of the 34th Design Automation Conference*, Anaheim, CA, June, 1997

- [45] L. W. Nagel. *SPICE2: a computer program to simulate semiconductor circuits*. Electronic Research Lab. Report No. ERL-M520, University of California, Berkeley, May 1975.
- [46] Peter R. O'Brien and Thomas L. Savarino. *Modeling the driving-point characteristics of resistive interconnect for accurate delay estimation IEEE/ACM International Conference on Computer-Aided Design Digest of Technical Papers 1989* (ICCAD-89), pp. 512-515.
- [47] Altan Odabasioglu, Mustafa Celik, and Lawrence Pileggi. PRIMA: A Passive reduced-order interconnect macromodeling algorithm. In *International Conference on Computer Aided-Design*, pp. 58-65, San Jose, California, November 1997.
- [48] Makiko Okumura, Hiroshi Tanimoto, Tetsuro Itakura, and Tsutomu Sugawara. Numerical noise analysis for nonlinear circuits with a periodic large signal excitation including cyclostationary noise sources. *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 40, no. 9, September 1993, pp. 581-919.
- [49] Tekla S. Perry. Profile: Donald O. Pederson. *IEEE Spectrum*, vol. 35, no. 6, June 1998.
- [50] J. R. Phillips, E. Chiprout, and D. D. Ling, Efficient full-wave electromagnetic analysis via model-order reduction of fast integral transforms. In *Proceedings of the 33rd Design Automation Conference*, Las Vegas, June, 1996.
- [51] J. Phillips, and K. Kundert. Noise in mixers, oscillators, samplers, and logic an introduction to cyclostationary noise. *Proceedings of the IEEE Custom Integrated Circuits Conference, 2000* (CICC-2000), pp. 431-438.
- [52] Lawrence T. Pillage and Ronald A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Trans. CAD*, 9(4):352-366, April 1990.
- [53] J. J. Rael and A. A. Abidi. Physical processes of phase noise in differential LC oscillators. *Proceedings of the 2000 IEEE Custom Integrated Circuits Conference* (CICC 2000), pp. 569 -572.
- [54] Curtis L. Ratzlaff and Lawrence T. Pillage. RICE: Rapid interconnect circuit evaluation using AWE. *IEEE Trans. CAD*, 13(6):763-776, June 1994.
- [55] G. Rhyne, M. Steer and B. Bates. Frequency-domain nonlinear circuit analysis using generalized power series. *IEEE Transactions on Microwave Theory and Techniques*, vol. 36, no. 2, pp. 717-720, February 1988.
- [56] V. Rizzoli and A. Neri. State of the art and present trends in nonlinear microwave CAD techniques. *IEEE Transactions on Microwave Theory and Techniques*, vol. 36, no. 2, pp. 343-365, February 1988.
- [57] R. Rohrer, L. Nagel, R. G. Meyer, and L. Weber. Computationally efficient electronic-circuit noise calculations. *IEEE Journal of Solid-State Circuits*, vol. SC-6, no. 4, pp. 204, August 1971.
- [58] J. Roychowdhury, D. Long, and P. Feldmann. Cyclostationary noise analysis of large RF circuits with multitone excitations. *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, March 1998, pp. 324-336.
- [59] J. Roychowdhury, Reduced-order modeling of time-varying systems *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 46, no. 10, October 1999, pp. 1273-1288.
- [60] J. Rubinstein, P. Penfield, Jr., and M. A. Horowitz. Signal delay in RC tree networks. *IEEE Transaction on Computer-Aided Design*, CAD-2(3):202-211, July 1983.
- [61] A. E. Ruehli. Survey of computer-aided electrical analysis of integrated circuit interconnections. *IBM Journal of Research and Development*, vol. 23, November 1979, pp. 626-639.

- [62] L. M. Silveira, I. M. Elfadel, J. White. A guaranteed stable model-order reduction algorithm for packaging and interconnect simulation. *Proceedings of the IEEE 2nd Topical Meeting on Electrical Performance of Electronic Packaging*, Monterey, CA, October, 1993, pp. 166-168.
- [63] L. Miguel Silveira, M. Kamon and J. White, Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-D interconnect structures, *Proceedings of the 32nd Design Automation Conference*, pp. 376-380, San Francisco, CA, June, 1995.
- [64] K. L. Shepard and Tian Zhong. Return-limited inductances: a practical approach to on-chip inductance extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 4, April 2000, pp. 425-436.
- [65] R. Telichevesky, K. Kundert and J. White. Efficient steady-state analysis based on matrix-free Krylov-subspace methods. *Proceedings of the 32nd Design Automation Conference*, June 1995.
- [66] Ricardo Telichevesky, Kenneth S. Kundert, and Jacob K. White. Efficient AC and noise analysis of two-tone RF circuits. *Proceedings of the 33rd Design Automation Conference*, June 1996.
- [67] V. Thomas, M. Jones, M. J. Piket-May, A. Taflove, and E. Harrigan, The use of SPICE lumped circuits as sub-grid models for FD-TD analysis. *IEEE Micro. Guided Wave Letters*, July 1994.
- [68] L. Toth, I. Yusim, K. Suyama. Noise analysis of ideal switched-capacitor networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 46, no. 3, March 1999, pp. 349-363.
- [69] L. Toth. Analytical approach for the exact phase noise analysis of oscillators. *The 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001)*, vol. 2, pp. 771-774.
- [70] J. L. Wyatt, Jr. Signal propagation delay in RC models for interconnect, chapter 11.2, pages 254-290. *Advances in CAD for VLSI*. Elsevier Science Publishers B. V., North Holland, 1987.

# NONLINEAR CIRCUIT SIMULATION IN THE FREQUENCY-DOMAIN

Kenneth S. Kundert<sup>1</sup> and Alberto Sangiovanni-Vincentelli

*University of California*

*Berkeley, CA. 94720*

## Abstract

Simulation in the frequency-domain avoids many of the severe problems experienced when trying to use traditional time-domain simulators such as SPICE [1] to find the steady-state behavior of analog, RF, and microwave circuits. In particular, frequency-domain simulation eliminates problems from distributed components and high-Q circuits by forgoing a nonlinear differential equation representation of the circuit in favor of a complex algebraic representation.

This paper describes the *spectral Newton* technique for performing simulation of nonlinear circuits in the frequency-domain, and its implementation in *Harmonica*. Also described are the techniques used by *Harmonica* to exploit both the structure of the spectral Newton formulation and the characteristics of the circuits that would be typically seen by this type of simulator. These techniques allow *Harmonica* to be used on much larger circuits than were normally attempted by previous nonlinear frequency-domain simulators, making it suitable for use on Monolithic Microwave Integrated Circuits (MMICs).

## 1. Introduction

It is common for circuits designed to operate at RF and microwave frequencies to be pseudo-linear in nature. By this it is meant that input signals are sinusoidal and small enough so that few harmonics are produced. This does not imply that the nonlinearities in the circuit can be neglected. Indeed, mixers and oscillators fit this description and yet they fundamentally depend on nonlinear effects to operate. It is also common for these circuits to have a large number of distributed components such as transmission lines, whose models often include loss, dispersion, and coupling effects. These distributed components are very difficult and often impractical to simulate in the time-domain because the partial differential equations that describe these structures often do not have closed-form solutions. In addition, time-domain simulators are not able to exploit the pseudo-linear nature of these circuits, and often require an excessive amount of time because the steady-state solution is desired. Using a time-domain simulator to find the steady-state solution requires that the circuit be simulated until the transient solution vanishes, resulting in a very expensive simulation when the circuit is high-Q or narrow-band.

---

<sup>1</sup>Author is currently with Cadence Design Systems, Inc.

Simulating these circuits in the frequency-domain avoids these problems and eases the problem of formulating the equations for distributed components by transforming the time-domain differential equations into algebraic complex equations. The pseudo-linear nature of these circuits is naturally exploited since the amount of cpu time required for a frequency-domain simulation is proportional to the number of frequencies present. Another method has been proposed to find the steady-state solution [2, 3]. The shooting method, as it is often called, iteratively solves the circuit in the time-domain for one period; on each iteration the initial condition is varied, attempting to make the signals at the end of the period exactly match those at the beginning. The shooting method does work on autonomous circuits, but does not help with distributed components and is not capable of finding almost-periodic solutions.

Previous efforts at nonlinear frequency-domain simulation were based on the use of *harmonic balance* to formulate the frequency-domain equations and an optimizer to solve them [4, 5, 6]. Using an optimizer to solve these equations results in the number of harmonics and nonlinear devices being severely limited. It is possible to remove this limit by instead solving the nonlinear equations with Newton's method [7]. When this is done the circuit equations can be reformulated in a more natural way, and by doing so the name harmonic balance becomes somewhat of a misnomer. So the more appropriate name *spectral Newton* was coined.

## 2. Spectral Newton

In order to apply the spectral Newton method, two conditions must be satisfied. First, the circuit must be asymptotically stable and must have a steady-state solution for the given excitation; chaotic and subharmonic behavior is specifically excluded. Second, all nonlinear devices must be lumped and their constitutive relationship must be algebraic, differentiable, and expressible in one of the following forms:

$$\begin{aligned} i &= i(v) & q &= q(v) & i &= i(\phi) & q &= q(\phi) \\ v &= v(i) & v &= v(q) & \phi &= \phi(i) & \phi &= \phi(q) \end{aligned}$$

Though not necessary, we will assume that the circuit has a periodic solution. Extension of these results to almost-periodic solutions is straight-forward [7]. For simplicity, we will further assume that a nodal formulation is being used and that only voltage controlled resistive and capacitive nonlinearities are allowed.

In the time-domain a circuit can be modeled as a system of  $N$  nonlinear differential equations, here written in compact form as

$$f(v, t) = i_s(t) \quad v(0) = v_0 \tag{1}$$

Let  $U = \{h | h : \mathbb{R} \rightarrow \mathbb{R}^N\}.$ <sup>1</sup> Then  $v \in U$  is the vector of unknown node voltage waveforms;  $v_0 \in \mathbb{R}^N$  is the unknown initial condition that results in the solution

being periodic, i.e.  $v(t) = v(t + T_0) \forall t$ ;  $i_s \in U$  is the vector of source current waveforms; and  $f : U \times \Re \rightarrow \Re^N$ . In order to solve this system it is traditional to discretize it in time and apply some numeric integration method. However if only the steady-state response is of interest, it is possible to transform this system into the frequency-domain and solve it without resorting to numeric integration. To solve the system in the frequency-domain, it is necessary to truncate the number of harmonics considered to a finite, and in general small,  $H$ . The truncation is analogous to discretization in the time-domain and is theoretically not a limitation because for all realizable circuits there exists a frequency beyond which there is negligible power.

Since the nonlinear devices are lumped,  $f(v, t)$  can be rewritten as

$$f(v, t) = i(v(t)) + \frac{d}{dt}q(v(t)) + \int_0^t y(t - \tau)v(\tau)d\tau \quad (2)$$

where  $i, q : \Re^N \rightarrow \Re^N$  are differentiable functions representing respectively the sum of the currents exiting the nodes due to the nonlinear conductors and the sum of the charge exiting the nodes due to the nonlinear capacitors; and  $y(t) \in \Re^N$  is the impulse response of the circuit with the nonlinear devices turned off.<sup>2</sup>

Since  $y$  is linear, the Laplace transform may be used to transform it into the frequency-domain,  $y(t) \leftrightarrow Y(s)$ . Furthermore, since  $v$  is periodic and the circuit is stable

$$\int_0^t y(t - \tau)v(\tau)d\tau \leftrightarrow YV$$

where  $v \leftrightarrow V \in C^{HN}$  contains the node voltage phasor for each node and each frequency, and  $Y \in C^{HN \times HN}$  is a block node admittance matrix for the linear portion for the circuit.

$$Y = [Y_{mn}] \quad m, n \in \{1, 2, \dots, N\}$$

$$Y_{mn} = [Y_{mn}(k\omega_o, l\omega_o)] \quad k, l \in \{0, 1, \dots, H - 1\}$$

$$Y_{mn}(k\omega_o, l\omega_o) = \begin{cases} Y_{mn}(jk\omega_o) & k = l \\ 0 & k \neq l \end{cases}$$

where  $m, n$  are the node indices;  $k, l$  are the frequency indices, and  $j = \sqrt{-1}$ .

Since  $v$ ,  $i$  and  $q$  are periodic, (1) and (2) can be transformed into the frequency domain by applying the Fourier series.

$$F(V) = I(V) + j\Omega Q(V) + YV = I_s \quad (3)$$

where  $i_s \leftrightarrow I_s \in C^{HN}$  contains the source current phasor for each node and frequency;  $f \leftrightarrow F$ ,  $i \leftrightarrow I$ ,  $q \leftrightarrow Q : C^{HN} \rightarrow C^{HN}$ ; and  $\Omega \in C^{HN \times HN}$

$$\Omega = [\Omega_{mn}] \quad m, n \in \{1, 2, \dots, N\}$$

$$\Omega_{mn} = \begin{cases} 0 & m \neq n \\ \text{diag}\{0, \omega_o, 2\omega_o, \dots, (H-1)\omega_o\} & m = n \end{cases}$$

and  $\omega_o = 2\pi/T_o$ .

The Newton-Raphson method is used to solve (3) for  $V$ , which requires that  $F(V)$  be differentiated with respect to  $V$ . However, since  $f(v, t)$  and  $v(t)$  are constrained to be real functions,  $F(V)$  is non-analytic, which implies that its derivative  $J(V)$  cannot be represented using complex numbers. To circumvent this problem each complex number is written as an equivalent vector in  $\Re^2$ . To perform this conversion, some more notation will be defined. Let  $X \in C$ . Then define  $X^R, X^I \in \Re$ ,  $\bar{X} \in \Re^2$  such that  $X^R = \text{Re}\{X\}$ ,  $X^I = \text{Im}\{X\}$ , and  $\bar{X} = [X^R \ X^I]^T$ . Similar notation is used for vectors and matrices. Using this notation,  $\bar{F}(\bar{V}), \bar{V} \in \Re^{2HN}$  and (3) is solved with the iteration

$$\bar{V}^{(k+1)} = \bar{V}^{(k)} - \bar{J}(\bar{V}^{(k)})^{-1}[\bar{F}(\bar{V}^{(k)}) - \bar{I}_s] \quad (4)$$

where  $J \in \Re^{2NH \times 2NH}$  is the spectral Jacobian, i.e.

$$\bar{J}(\bar{V}) = \frac{\partial \bar{F}(\bar{V})}{\partial \bar{V}} = \frac{\partial \bar{I}(\bar{V})}{\partial \bar{V}} + j\Omega \frac{\partial \bar{Q}(\bar{V})}{\partial \bar{V}} + \bar{Y}$$

If  $V^{(0)}$  is chosen close enough to a solution, then given certain mild conditions on (3), the sequence converges to that solution [9].

The only impediment in evaluating this expression is finding the contribution of the nonlinear elements to  $F(V)$  and  $J(V)$  because it is extremely difficult to formulate the nonlinear device equations directly in the frequency-domain. To avoid this problem, the node voltages are transformed into the time-domain and applied to the nonlinear devices. The response current of these devices is then calculated and converted back into the frequency-domain and added to  $F(V)$ . Calculation of  $J(V)$  is similar except that the node voltage waveforms are applied to the devices' derivative equations and the resulting waveforms are converted into the frequency-domain and added to  $J(V)$ . The calculation of the spectral Jacobian will be covered in more detail in the next section. Since the signals are assumed to be periodic, the Fast Fourier Transform (FFT) may be used to perform the transformations between the frequency- and time-domains. If the periodic signal restriction is loosened to allow almost-periodic signals, then the Discrete Fourier Transform (DFT) should be used.

---

#### Spectral Newton Algorithm

---

- Given: Initial guess of node voltage spectra taken from DC and small-signal AC analysis of circuit.
- Step 1: Convert node voltage spectra into time-domain.
- Step 2: Evaluate nonlinear devices for output current and derivative waveforms.
- Step 3: Convert the waveforms into the frequency-domain.
- Step 4: Build and solve the spectral Newton update equation (4).
- Step 5: Check  $F(V)$  and  $\Delta V$  for convergence, if not converged, go to step 1.
-

### 3. Spectral Jacobian

In our method, the spectral Jacobian is organized as the block matrix

$$\bar{J}(\bar{V}) = \left[ \frac{\partial \bar{F}_m(\bar{V})}{\partial \bar{V}_n} \right] \quad m, n \in \{1, 2, \dots, N\} \quad (5)$$

where  $\bar{F}_m, \bar{V}_n \in \Re^{2H}$  are vectors of phasors, one phasor for each frequency.  $\bar{F}_m$  equals the sum of currents exiting node  $m$  and  $\bar{V}_n$  equals the node voltage of node  $n$ . This block matrix is referred to as the block node admittance matrix because its structure is identical to the node admittance matrix. The blocks have the form

$$\frac{\partial \bar{F}_m}{\partial \bar{V}_n} = \left[ \frac{\partial \bar{F}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} \right] \quad k, l \in \{0, 1, \dots, H - 1\} \quad (6)$$

where  $\bar{F}_m(\bar{V}, k\omega_o) \in \Re^2$  is the  $k^{\text{th}}$  harmonic of  $\bar{F}_m$  and  $\bar{V}_n(l\omega_o) \in \Re^2$  is the  $l^{\text{th}}$  harmonic of  $\bar{V}_n$ .

$$\frac{\partial \bar{F}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} = \begin{bmatrix} \frac{\partial F_m^R(\bar{V}, k\omega_o)}{\partial V_n^R(l\omega_o)} & \frac{\partial F_m^I(\bar{V}, k\omega_o)}{\partial V_n^I(l\omega_o)} \\ \frac{\partial F_m^I(\bar{V}, k\omega_o)}{\partial V_n^R(l\omega_o)} & \frac{\partial F_m^I(\bar{V}, k\omega_o)}{\partial V_n^I(l\omega_o)} \end{bmatrix}$$

This derivative consists of the sum of terms

$$\frac{\partial \bar{F}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} = \frac{\partial \bar{I}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} + \quad (7)$$

$$\begin{bmatrix} 0 & -k\omega_o \\ k\omega_o & 0 \end{bmatrix} \frac{\partial \bar{Q}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} + \bar{Y}_{mn}(k\omega_o, l\omega_o)$$

$$\bar{Y}_{mn}(k\omega_o, l\omega_o) = \begin{bmatrix} Y_{mn}^R(k\omega_o, l\omega_o) & -Y_{mn}^I(k\omega_o, l\omega_o) \\ Y_{mn}^I(k\omega_o, l\omega_o) & Y_{mn}^R(k\omega_o, l\omega_o) \end{bmatrix} \quad (8)$$

Only the calculation of  $\frac{\partial \bar{I}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)}$  will be performed, the calculation of the other terms in  $\frac{\partial \bar{I}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)}$  and  $\frac{\partial \bar{Q}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)}$  is similar.

$$I_m(V, k\omega_o) = \frac{1}{T_o} \int_0^{T_o} i_m(v(t)) e^{-jk\omega_o t} dt$$

$$I_m^R(\bar{V}, k\omega_o) = \frac{1}{T_o} \int_0^{T_o} i_m(v(t)) \cos(k\omega_o t) dt$$

The function  $v$  is considered implicitly to be a function of its frequency-domain equivalent,  $V$ ; so the chain rule can be employed to calculate the derivative.

$$\frac{\partial I_m^R(\bar{V}, k\omega_o)}{\partial V_n^R(l\omega_o)} = \frac{1}{T_o} \int_0^{T_o} \frac{\partial i_m(v(t))}{\partial v_n(t)} \frac{\partial v_n(t)}{\partial V_n^R(l\omega_o)} \cos(k\omega_o t) dt$$

Now the derivative of  $v_n(t)$  is calculated.

$$v_n(t) = \sum_{k=-\infty}^{\infty} V_n(k\omega_o) e^{jk\omega_o t}$$

$$v_n(t) = V_n^R(0) + 2 \sum_{k=1}^{\infty} V_n^R(k\omega_o) \cos(k\omega_o t) - V_n^I(k\omega_o) \sin(k\omega_o t)$$

For  $l = 0$ , the derivative is trivial; for  $l \neq 0$

$$\frac{\partial v_n(t)}{\partial \bar{V}_n(l\omega_o)} = \begin{bmatrix} \frac{\partial v_n(t)}{\partial V_n^R(l\omega_o)} \\ \frac{\partial v_n(t)}{\partial V_n^I(l\omega_o)} \end{bmatrix} = \begin{bmatrix} 2 \cos(l\omega_o t) \\ -2 \sin(l\omega_o t) \end{bmatrix}$$

So if  $l \neq 0$

$$\begin{aligned} \frac{\partial I_m^R(\bar{V}, k\omega_o)}{\partial V_n^R(l\omega_o)} &= \frac{2}{T_o} \int_0^{T_o} \frac{\partial i_m(v(t))}{\partial v_n(t)} \cos(l\omega_o t) \cos(k\omega_o t) dt \\ &= \frac{1}{T_o} \int_0^{T_o} \frac{\partial i_m(v(t))}{\partial v_n(t)} [\cos((k+l)\omega_o t) + \cos((k-l)\omega_o t)] dt \end{aligned}$$

Now let  $G_{mn}(k\omega_o) \in C$  be the  $k^{\text{th}}$  harmonic of  $\frac{\partial i_m(v(t))}{\partial v_n(t)}$ , i.e., let

$$G_{mn}(k\omega_o) = \frac{1}{T_o} \int_0^{T_o} \frac{\partial i_m(v(t))}{\partial v_n(t)} e^{jk\omega_o t} dt \quad (9)$$

Then for  $l = 0$

$$\frac{\partial \bar{I}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(0)} = \begin{bmatrix} G_{mn}^R(k\omega_o) & 0 \\ G_{mn}^I(k\omega_o) & 0 \end{bmatrix} \quad (10)$$

and for  $l \neq 0$

$$\begin{aligned} \frac{\partial \bar{I}_m(\bar{V}, k\omega_o)}{\partial \bar{V}_n(l\omega_o)} &= \\ \begin{bmatrix} G_{mn}^R((k+l)\omega_o) + G_{mn}^R((k-l)\omega_o) & G_{mn}^I((k+l)\omega_o) + G_{mn}^I((k-l)\omega_o) \\ G_{mn}^I((k+l)\omega_o) - G_{mn}^I((k-l)\omega_o) & G_{mn}^R((k-l)\omega_o) - G_{mn}^R((k+l)\omega_o) \end{bmatrix} \end{aligned} \quad (11)$$

This completes the calculation of the spectral Jacobian. It may now be synthesized from (5), (6), (7), (8), (9), (10) and (11).

## 4. Harmonica

We are currently developing a simulator based on the spectral Newton algorithm. Unlike previous efforts [4, 7], which were aimed at circuits containing only one or two nonlinear devices, *Harmonica* is designed to quickly analyze large circuits with many nonlinear devices. This advance is made possible

by using spectral Newton, by exploiting the structure and characteristics of the spectral Jacobian, and by exploiting the linear and almost-linear behavior of the devices.

The spectral Jacobian is quite large and moderately dense, having about  $4H$  elements per row or column. Naively applying sparse matrix techniques is not enough to solve the Newton update equation (4) efficiently. It is necessary to make some judicious approximations when constructing and decomposing the Jacobian to reduce the density of the matrix. The Jacobian is only used to generate new iterates, and is not used when confirming convergence, so errors from approximations in the Jacobian only affect the rate and region of convergence, not the accuracy of the final solution. An approximate spectral Jacobian results in the loss of quadratic convergence, but the gain in efficiency more than makes up for this loss.

In a node admittance matrix, any particular element is the sum of contributions from zero or more devices. This is also true for the block node admittance matrix generated by the spectral Newton algorithm. In the block node admittance matrix, contributions from linear devices come as diagonal blocks, i.e. only the diagonal  $2 \times 2$  sub-blocks are nonzero. Nonlinear devices contribute full blocks, however if the device is behaving almost-linearly the elements on the diagonal of the block are the largest and as the distance from the diagonal increases their magnitude decreases rapidly. This results from (10) and (11), and from the bandwidth of the derivative spectrum (9) being small if the device is behaving almost-linearly.

The effort required to LU decompose the spectral Jacobian can be significantly reduced if two approximations are made. First, in those blocks that have contributions only from elements behaving linearly or almost-linearly, the small elements far from the diagonal should be set to zero and the operations that would normally be performed on these elements should be avoided. The decision of which elements are small enough to ignore can be made by comparing the magnitude of the upper harmonics of the derivative spectrum to some small fraction of the DC component. The value  $10^{-4}G_{mn}(0)$  seems to work well. Of those harmonics smaller than the cutoff criterion, only the first should be kept: all others should be set to zero. This last nonzero harmonic is called the *guard harmonic*. Second, all nonzero fill-ins that result during LU decomposition from operations involving the guard harmonic should be ignored. This prevents the bandwidth of the blocks from growing unnecessarily during the decomposition. These two approximations allow *Harmonica* to exploit linear and almost-linear behavior in the circuit. To get the most from them, pivoting of the block node admittance matrix should be done with the additional goal of exploiting the reduced bandwidth of the blocks.

The last technique used to accelerate the spectral Newton iteration is to only occasionally reevaluate the spectral Jacobian [9]. This works well if the Ja-

cobian is not changing much between iterations. It can greatly reduce the time required for an iteration because device evaluations and forward- and backward-elimination of the LU decomposed Jacobian are much faster than the decomposition of the spectral Jacobian.

*Harmonica* is written in the C programming language.

## 5. Results

Execution times for *Harmonica* are a strong function of the number of harmonics simulated, the strength of the nonlinear behavior, and the number of devices behaving nonlinearly. Before applying the techniques given in the previous section each iteration requires  $O(N^{1.5}H^3)$  operations. After applying those techniques, and measuring the execution times of only a few circuits, each iteration seems to require  $O(N^{1.5}H)$  operations. The iteration count remains relatively constant as the number of harmonics changes.

The times for three circuits are presented in Table 1. The first two circuits are well-suited to simulation in the frequency-domain and poorly suited to time-domain simulation. With the last circuit, the roles are reversed. The first is a traveling wave amplifier (TWA) [10] that contains four bipolar transistors and ten transmission lines of noncommensurate length. Note that the transmission lines are constrained to be ideal by SPICE, *Harmonica* easily handles lossy and dispersive lines. The second circuit contains a differential pair and a crystal lattice filter. This circuit demonstrates the ease with which *Harmonica* handles high-Q circuits.

The last circuit, a simple noninverting amplifier containing a  $\mu A741$ , is troublesome to *Harmonica* because the op amp is internally acting strongly nonlinear: the large load causing the output stage to operate class B. This example demonstrates that *Harmonica* is able to handle strongly nonlinear circuits, though it may run longer than traditional simulators.

Since *Harmonica* is solving an algebraic system of equations, if sufficient harmonics are computed, it can be much more accurate than a time-domain simulator. This is demonstrated in all the test circuits: when *Harmonica* was able to converge with only eight harmonics computed, the maximum error in any harmonic was less than 1ppm. Furthermore, the worst case error resulting from harmonics not computed was less than 20ppm. These numbers were greatly reduced when more than eight harmonics are computed. SPICE2 computes with a 1000ppm error tolerance.

## 6. Conclusions

The spectral Newton method for frequency-domain simulation of nonlinear circuits was described along with techniques used by *Harmonica* to increase the efficiency of the method. This method allows circuits that are behaving

Circuit	Conditions	SPICE2	<i>Harmonica</i> harmonics		
			8	16	32
TWA	$V_{out} = 1 \text{ V}$	62,500 <sup>3</sup>	7	22	56
TWA	$V_{out} = 0.5 \text{ V}$	NA <sup>4</sup>	6	16	40
Filter		2350	7	20	94
$\mu A741$	$V_{out} = 1 \text{ V}$ $R_L = \infty \Omega$	9	6	13	29
$\mu A741$	$V_{out} = 1 \text{ V}$ $R_L = 10 \text{ K}\Omega$	13	10	28	63
$\mu A741$	$V_{out} = 1 \text{ V}$ $R_L = 10 \text{ K}\Omega$	14	NA <sup>5</sup>	365	575

Table 1. Simulation times for SPICE2 and *Harmonica* for various circuits. Times are given in seconds and were measured on a VAX 11/785 running UNIX 4.3BSD.

quasi-linearly to be quickly simulated, even though they may be very high-Q or contain many transmission lines.

Work is being done to at least double the speed of the simulator by further exploiting the structure of the spectral Jacobian.

## 7. Acknowledgements

We would like to thank Peter Moore for his contributions to *Harmonica*; and Jacob White, Howard Ko, and the rest of the Berkeley CAD research group for many helpful discussions. This work was supported by Hewlett-Packard and MICRO.

## Notes

1. In function space, no one can hear you scream [8].
2. To turn a nonlinear device off, simply replace its constitutive equation  $y = f(x)$  with  $y = 0$ .
3. This number is an extrapolation made from measurements of times required for smaller simulation intervals. The desired time interval (two periods) causes memory usage to exceed UNIX's 16 MByte limit.
4. This time was not measured.
5. Circuit was behaving too nonlinearly for *Harmonica* to converge with so few harmonics.

## References

- [1] Laurence W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. Electronics Research Laboratory Publications, U.C.B., 94720, Memorandum No. UCB/ERL M520, May 1975.
- [2] T.J. Aprille, T.N. Trick. Steady-state analysis of nonlinear circuits with periodic inputs. *Proceedings of the IEEE*, vol. 60, pp. 108-114, January 1972.

- [3] Stig Skelboe. Computation of the periodic steady-state response of nonlinear networks by extrapolation methods. *IEEE Transactions on Circuits and Systems*, vol. CAS-27, no. 3, pp. 161-175, March 1980.
- [4] M. Nakhla, J. Vlach. A piecewise harmonic balance technique for determination of the periodic response of nonlinear systems. *IEEE Transactions on Circuits and Systems*, vol. CAS-23, no. 2, pp. 85-91, February 1976.
- [5] K. Gopal, M.S. Nakhla, K. Singhal, J. Vlach. Distortion analysis of transistor networks. *IEEE Transactions on Circuits and Systems*, vol. CAS-25, no. 2, pp. 99-106, February 1978.
- [6] F. Filicori, O. Monaco, C. Naldi. Simulation and design of microwave class-C amplifiers through harmonic analysis. *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-27, no. 12, pp. 1042-1051, December 1979.
- [7] A. Ushida, L.O. Chua. Frequency-domain analysis of nonlinear circuits driven by multi-tone signals. *IEEE Transactions on Circuits and Systems*, vol. CAS-31, no. 9, pp. 766-778, September 1984.
- [8] Richard J. Anobile. *Alien*, Avon, 1979. This modification of the Alien's teaser was found on a mens' bathroom wall in Cory Hall, U.C. Berkeley.
- [9] J.M. Ortega, W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, 1970.
- [10] J.B. Beyer, S.N. Prasad, R.C. Becker, J.E. Nordman, G.K. Hohenwarter. MESFET distributed amplifier design guidelines. *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-32, no. 3, pp. 268-275, March 1984.

# MODELING THE DRIVING-POINT CHARACTERISTIC OF RESISTIVE INTERCONNECT FOR ACCURATE DELAY ESTIMATION

Peter R. O'Brien<sup>1</sup>

*Digital Equipment Corporation*

Thomas L. Savarino<sup>2</sup>

*Cadence Design Systems*

## Abstract

In recent years, on-chip interconnect has had an increasingly important impact on overall system performance. Much work has been done to develop algorithms which can efficiently and accurately predict delay through on-chip interconnect. These algorithms compute a reduced-order approximation (usually based on the “Elmore delay”) for voltage-transfer ratios (from source to loads) in an RC-tree model for the interconnect. However, much less emphasis has been placed on accurately approximating the *driving-point* characteristic at the root of an RC-tree. A good driving-point approximation is needed to accurately predict how delay through a gate is influenced by the interconnect which that gate must drive. Macromodels for on-chip gates typically consider only total capacitance of the driven interconnect, completely ignoring series resistance. In this paper, we present an efficient algorithm which accounts for series resistance by computing a reduced-order approximation for the driving-point admittance of an RC-tree. Using an ECL clock buffer as an example, we demonstrate a significant improvement in accuracy.

## 1. Introduction

### 1.1 Interconnect Delay

“Interconnect delay” has two distinct components, which can best be illustrated by considering a simplified net with no branching and only one load gate (see Fig. 1). Let  $T_{AB}(0)$  represent delay through an *unloaded* source gate. Let  $T_{AB}(L)$  represent delay through the same source gate *loaded* by an interconnect net of length  $L$ . Because of loading effects on the source gate,  $T_{AB}(L)$  exceeds  $T_{AB}(0)$ . We define “interconnect delay” as follows:

$$\begin{aligned} T_{int} &\stackrel{\Delta}{=} T_{AC} - T_{AB}(0) \\ &= [T_{AB}(L) - T_{AB}(0)] + T_{BC}. \end{aligned} \quad (1)$$

---

Authors are currently with <sup>1</sup>Synopsys, Austin TX and <sup>2</sup>Sun Microsystems, Sunnyvale CA.

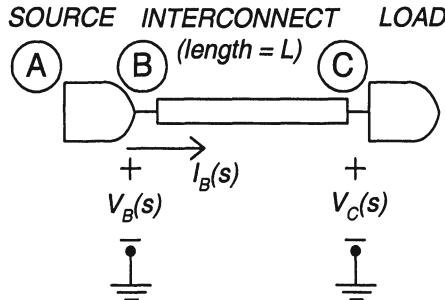


Figure 1. Simplified net (for illustrating components of  $T_{int}$ ).

The two components are *extra source gate delay* ( $T_{AB}(L) - T_{AB}(0)$ ), which is the focus of this paper; and *propagation delay* ( $T_{BC}$ ), which has been extensively analyzed [2]–[4].

## 1.2 Delay Modeling

On-chip interconnect is well modeled by RC-trees [1]. In the last several years, research has been active on fast algorithms for on-chip delay estimation and bounding using RC-tree models [2]–[6]. These algorithms have proven to be useful alternatives to “exact” numerical simulation (e.g., SPICE [7]), where computation time becomes quite large even for relatively small circuits. However, regarding interconnect delay, the cited works [2]–[6] concentrate mainly on the propagation component ( $T_{BC}$ ). A reduced-order approximation, based on the “Elmore delay” [8], is computed for the voltage-transfer ratio ( $V_C(s)/V_B(s)$ ) from the source gate output to a given load gate input. See Section 2 for a brief review of these voltage-transfer ratio approximations.

In this paper, we concentrate on the extra source gate component ( $T_{AB}(L) - T_{AB}(0)$ ) of  $T_{int}$ . We compute a reduced-order approximation for the driving-point admittance ( $I_B(s)/V_B(s)$ ) seen from the output of the source gate. This approximation for the driving-point admittance of interconnect is *independent* of any modeling approximations made for the non-linear behavior of the source gate. Our approximation accounts for distributed series resistance present in the interconnect. Earlier works take one of the following approaches:

1. The source gate is modeled simply with a linear output resistance. Delay through the loaded source gate ( $T_{AB}(L)$ ) is approximated by computing the Elmore delay to point B [2]–[4]. However, this is just the model source gate output resistance times the total load net capacitance to ground. Series resistance present in the interconnect does not influence this delay calculation.

2. The source gate is modeled more accurately with a non-linear macro-model. However, the source gate output waveform ( $v_B(t)$ ) is approximated with a macromodel response which is influenced by the driven net only through the net's total capacitance (again, series interconnect resistance is not considered) [5], [6].

## 2. Voltage-Transfer Ratio Approximations

In this section, we review two commonly used approximations for a voltage-transfer ratio in an RC-tree. The approximation which is most widely used today is based on the early work of Elmore [8] and was first developed for RC-tree models of on-chip interconnect in [2]. In situations demanding greater accuracy, a higher-order extension developed by Horowitz [5] has proven useful. Both approximations are influenced by the presence and distribution of series interconnect resistance.

Let  $h(t)$ ,  $h_{ELM}(t)$ , and  $h_{HOR}(t)$  denote respectively: the exact, the Elmore approximate, and the Horowitz approximate unit voltage impulse response at a given load in an RC-tree. Let  $H(s)$ ,  $H_{ELM}(s)$ , and  $H_{HOR}(s)$  denote the respective Laplace transforms of these impulse responses at the same load. The Elmore approximate transfer function has a single pole:

$$H_{ELM}(s) = \frac{1}{1 + s\tau_D}. \quad (2)$$

The Elmore time constant ( $\tau_D$ ) is determined by matching the *first moment* of the exact impulse response:

$$\int_0^\infty t h_{ELM}(t) dt = \int_0^\infty t h(t) dt. \quad (3)$$

The Horowitz approximate transfer function has two poles and one zero:

$$H_{HOR}(s) = \frac{1 + s\tau_z}{(1 + s\tau_1)(1 + s\tau_2)}. \quad (4)$$

The three parameters in Horowitz's approximation ( $\tau_z$ ,  $\tau_1$ , and  $\tau_2$ ) are determined by matching the *first two moments* of the exact impulse response and the “*sum of the open-circuit time constants*” (i.e., the coefficient of  $s$  in the denominator of the exact transfer function):

$$\int_0^\infty t h_{HOR}(t) dt = \int_0^\infty t h(t) dt \quad (5)$$

$$\int_0^\infty t^2 h_{HOR}(t) dt = \int_0^\infty t^2 h(t) dt \quad (6)$$

$$\frac{1 + b_1 s + b_2 s^2 + \dots}{1 + (\tau_1 + \tau_2)s + a_2 s^2 + \dots} = H(s). \quad (7)$$

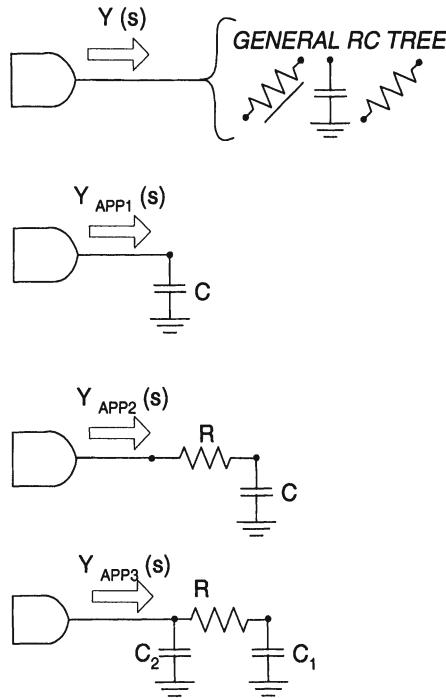


Figure 2. Circuit approximations for the driving-point admittance of a general RC-tree model for an interconnect net.

### 3. Driving-Point Admittance Approximations

In this section, we present three successively higher-order circuit approximations for the driving-point admittance of a general RC-tree (see Fig. 2). Let  $Y(s)$  denote the exact driving-point admittance of the RC-tree. Within some circle of convergence, we can represent  $Y(s)$  by its Taylor-series expansion around  $s = 0$ :

$$Y(s) = \sum_{n=1}^{\infty} y_n s^n. \quad (8)$$

For  $1 \leq i \leq 3$ , the circuit approximation  $Y_{APP_i}(s)$  shown in Fig. 2 matches expansion (8) to order  $i$ .

Let  $y(t)$  denote the inverse Laplace transform of  $Y(s)$  (i.e.,  $y(t)$  is the response current into a general RC-tree caused by an applied unit voltage impulse). Matching higher-order terms in (8) is mathematically equivalent to matching higher-order *time moments* of  $y(t)$ , since

$$y_n = \frac{(-1)^n}{n!} \int_0^{\infty} t^n y(t) dt, \quad (9)$$

so the approximations described in this section (for a driving-point admittance) are quite analogous to those described in the previous section (for a voltage-transfer ratio).

The approximation  $Y_{APP\ 1}(s) = Cs$ , where  $C = y_1 = C_{LOAD}$  and  $C_{LOAD}$  is simply the total load net capacitance to ground, is widely used. In fact, data-book descriptions of gates from semiconductor vendors use gate delay as a function of load net capacitance to characterize the drive capability of their cells. However, we show in Section 5 that significant errors can result from ignoring metal resistance.

The RC-lump approximation,

$$\begin{aligned} Y_{APP\ 2}(s) &= \frac{sC}{1+sRC} \\ &= \sum_{n=1}^{\infty} (-1)^{n-1} R^{n-1} C^n s^n, \end{aligned} \quad (10)$$

matches (8) to second order by setting

$$C = y_1 \quad (= C_{LOAD}) \quad (11)$$

$$R = -y_2/y_1^2. \quad (12)$$

The CRC pi-segment approximation,

$$\begin{aligned} Y_{APP\ 3}(s) &= sC_2 + \frac{sC_1}{1+sRC_1} \\ &= (C_1 + C_2)s + \sum_{n=2}^{\infty} (-1)^{n-1} R^{n-1} C_1^n s^n, \end{aligned} \quad (13)$$

provides even more accuracy by matching (8) to third order:

$$C_1 = y_2^2/y_3 \quad (14)$$

$$C_2 = y_1 - (y_2^2/y_3) \quad (15)$$

$$R = -y_3^2/y_2^3. \quad (16)$$

Note that  $C_1 + C_2 = y_1 = C_{LOAD}$  in the pi-segment approximation.

We have found that either second- or third-order matching provides sufficient accuracy in all cases of practical interest to us, though, in principle, arbitrarily high orders of the driving-point admittance could be matched with appropriate lumped circuit models. In this section, it is assumed the necessary series expansion coefficients of  $Y(s)$  (i.e.,  $y_1$ ,  $y_2$ , and  $y_3$ ) are known. In Section 4, we present an efficient algorithm for computing these coefficients.

#### 4. Algorithm For Series Coefficients

In this section, we present our algorithm for computing (the first three) Taylor-series expansion coefficients of  $Y(s)$  which are needed to perform the matching

to a lumped circuit approximation described in Section 3. The algorithm starts at the leaf nodes of an RC-tree and works back to the source in a finite sequence of steps.

The algorithm consists of four rules which allow the Taylor-series expansion coefficients of the driving-point admittance *looking downstream of a given point in the tree* to be correctly propagated further upstream (see Fig. 3). Rules #1–3 involve movement upstream, along a single branch, past respectively: a lumped capacitor to ground, a series lumped resistor, and a uniform distributed RC segment. Rule #4 involves combining two or more different admittance expansions in parallel at a branch point in the tree.

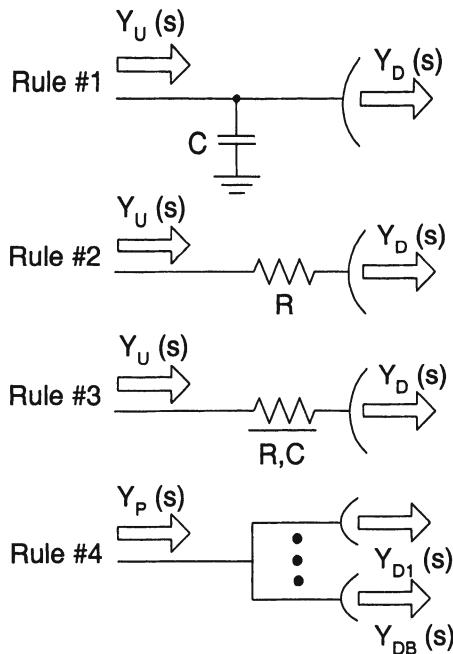


Figure 3. Four rules for upstream propagation of driving-point admittance expansion coefficients.

For rules #1–3, denote the known admittance expansion (looking downstream from the point which is immediately *downstream* of the circuit element to be traversed) by

$$Y_D(s) = \sum_{n=1}^3 (y_D)_n s^n + o(s^4). \quad (17)$$

Denote the unknown admittance expansion (looking downstream from the point which is immediately *upstream* of the circuit element to be traversed) by

$$Y_U(s) = \sum_{n=1}^3 (y_U)_n s^n + o(s^4). \quad (18)$$

Rule #1 states that for upstream traversal of a lumped capacitor ( $C$ ) to ground,

$$(y_U)_1 = (y_D)_1 + C \quad (19)$$

$$(y_U)_2 = (y_D)_2 \quad (20)$$

$$(y_U)_3 = (y_D)_3. \quad (21)$$

Rule #2 states that for upstream traversal of a series lumped resistor ( $R$ ),

$$(y_U)_1 = (y_D)_1 \quad (22)$$

$$(y_U)_2 = (y_D)_2 - R(y_D)_1^2 \quad (23)$$

$$(y_U)_3 = (y_D)_3 - 2R(y_D)_1(y_D)_2 + R^2(y_D)_1^3. \quad (24)$$

Rule #3 states that for upstream traversal of a uniform distributed RC segment (total capacitance  $C$  and total resistance  $R$ ),

$$(y_U)_1 = (y_D)_1 + C \quad (25)$$

$$(y_U)_2 = (y_D)_2 - R \left[ (y_D)_1^2 + C(y_D)_1 + \frac{1}{3}C^2 \right] \quad (26)$$

$$\begin{aligned} (y_U)_3 &= (y_D)_3 - R [ 2(y_D)_1(y_D)_2 + C(y_D)_2 ] + \\ &\quad R^2 \left[ (y_D)_1^3 + \frac{4}{3}C(y_D)_1^2 + \frac{2}{3}C^2(y_D)_1 + \frac{2}{15}C^3 \right]. \end{aligned} \quad (27)$$

For rule #4, let  $B$  ( $\geq 2$ ) denote the number of branches to be combined in parallel. Denote the  $B$  known downstream admittance expansions by

$$Y_{D_i}(s) = \sum_{n=1}^3 (y_{D_i})_n s^n + o(s^4), \quad 1 \leq i \leq B. \quad (28)$$

Denote the admittance expansion of the parallel combination by

$$Y_P(s) = \sum_{n=1}^3 (y_P)_n s^n + o(s^4). \quad (29)$$

Parallel admittances simply add together, and so do corresponding terms of their Taylor-series expansions. Hence, rule #4 states:

$$(y_P)_1 = \sum_{i=1}^B (y_{D_i})_1 \quad (30)$$

$$(y_P)_2 = \sum_{i=1}^B (y_{D_i})_2 \quad (31)$$

$$(y_P)_3 = \sum_{i=1}^B (y_{D_i})_3. \quad (32)$$

## 5. Results

To conclude, we show plots of driver delay versus metal loading. The metal loading we consider is an unbranched uniform distributed RC segment, which has a driving-point admittance of precisely

$$Y(s) = \sqrt{\frac{sC_{LOAD}}{R_{LOAD}}} \tanh\left(\sqrt{sC_{LOAD}R_{LOAD}}\right), \quad (33)$$

where  $C_{LOAD}$  is the total capacitance and  $R_{LOAD}$  is the total series resistance of the segment.

The first-order (purely capacitive) approximation is  $C = C_{LOAD}$ , which ignores resistance in the metal load. The second-order (RC-lump) approximation works out to be

$$C = C_{LOAD} \quad (34)$$

$$R = \frac{1}{3}R_{LOAD}. \quad (35)$$

The third-order (pi-segment) approximation works out to be

$$C_1 = \frac{5}{6}C_{LOAD} \quad (36)$$

$$C_2 = \frac{1}{6}C_{LOAD} \quad (37)$$

$$R = \frac{12}{25}R_{LOAD}. \quad (38)$$

We compare the driver delay using each of these approximate loads to driver delay using the fully distributed load.

The driver we use is an ECL differential clock buffer. The two differential outputs are each loaded identically (by (33), or by one of its reduced-order lumped circuit approximations). Gate delay is measured as the crossing time of the signals at the buffer input to the crossing time at the buffer output. Gate

delay ( $T_{AB}$ ) is normalized to unloaded gate delay ( $T_{AB}(0)$ ). Total metal capacitance ( $C_{LOAD}$ ) is normalized to the maximum allowable load capacitance ( $C_{MAX}$ ), given the sizing of our clock buffer.

To illustrate the importance of metal *resistance*, we use two different metal widths: a high-resistance “narrow” metal and a low-resistance “wide” metal. Physically, wide metal is twice the width of narrow metal. For a given total metal capacitance, the total series resistance of a narrow metal segment is 3.2 times that of a wide segment (not the ideal factor of 4, because of a fringing-field capacitance component). As can be seen in Figs. 4 and 5, progressively longer metal lengths require progressively higher-order lumped circuit approximations to accurately model the “resistive shielding” of capacitance which is located far away from the driver.

## References

- [1] I. T. Ho and S. K. Mullick. Analysis of Transmission Lines on Integrated Circuit Chips. *IEEE Jour. Solid-State Circuits*, vol. SC-2, no. 4, pp. 201–208, December 1967.
- [2] J. Rubinstein, P. Penfield, Jr., and M. A. Horowitz. Signal Delay in RC Tree Networks. *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, pp. 202–211, July 1983.
- [3] T. Lin and C. A. Mead. Signal Delay in General RC Networks. *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 4, pp. 331–349, October 1984.
- [4] P. K. Chan. An Extension of Elmore’s Delay. *IEEE Trans. Circuits and Systems*, vol. CAS-33, no. 11, pp. 1147–1149, November 1986.
- [5] M. A. Horowitz. *Timing Models for MOS Circuits*. PhD Dissertation, Stanford University, Department of Electrical Engineering, January 1984.
- [6] M. D. Matson. *Macromodeling and Optimization of Digital MOS VLSI Circuits*. PhD Dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 1985.
- [7] L. W. Nagel. SPICE2: A Computer Program to Simulate Semiconductor Circuits. Memo ERL-M520, University of California, Berkeley, May 1975.
- [8] W. C. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers. *Jour. Appl. Physics*, vol. 19, no. 1, pp. 55–63, January 1948.

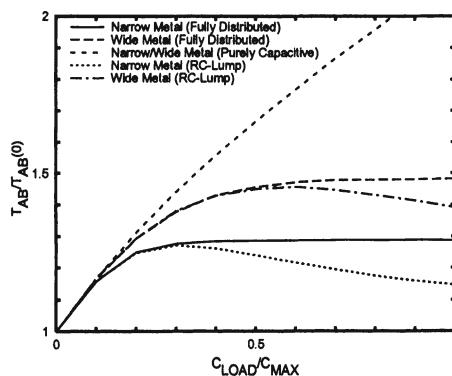


Figure 4. Comparisons for  $Y_{APP\,1}(s)$  and  $Y_{APP\,2}(s)$ .

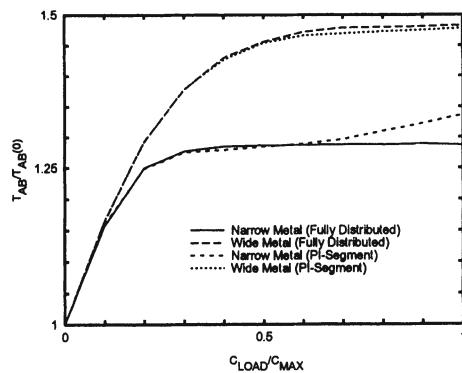


Figure 5. Comparison for  $Y_{APP\,3}(s)$ .

# **EFFICIENT TECHNIQUES FOR INDUCTANCE EXTRACTION OF COMPLEX 3-D GEOMETRIES**

M. Kamon<sup>1</sup>

*Massachusetts Institute of Technology  
Cambridge, MA 02139*

M. J. Tsuk<sup>2</sup>

*Digital Equipment Corporation  
Tewksbury, MA 01876*

C. Smithhisler and J. White

*Massachusetts Institute of Technology  
Cambridge, MA 02139*

## **Abstract**

In this paper we describe combining a mesh analysis equation formulation technique with a pre-conditioned GMRES matrix solution algorithm to accelerate the determination of inductances of complex three-dimensional structures. Results from FASTHENRY, our 3-D inductance extraction program, demonstrate that the method is more than an order of magnitude faster than the standard solution techniques for large problems.

## **1. Introduction**

In high performance VLSI integrated circuits and integrated circuit packaging, there are many cases where accurate estimates of the coupling inductances of complicated three dimensional structures are important for determining final circuit speeds or functionality, the most obvious example being the pin-connect structures used in advanced packaging. For the past decade, volume-element techniques have been used to compute self and coupling inductances of complex three dimensional geometries, but the techniques were intended for geometries which could be represented with at most a few hundred volume filaments. However, the complex structures currently used in integrated circuit packaging can require up to ten thousand filaments to be accurately analyzed. Existing pro-

---

Authors are currently with <sup>1</sup>Coventor, Inc., Cambridge, MA, and <sup>2</sup>Hewlett-Packard Company, Marlborough, MA.

grams become extraordinarily computationally expensive for such large problems, and new algorithms whose computational cost grows more slowly with problem size must be developed.

In this paper we describe how an old equation formulation technique, mesh analysis, can be combined with preconditioned GMRES, a relatively new iterative matrix solution technique, to make FASTHENRY, a fast 3-D inductance extraction program for general packaging structures. We start in the next section by describing a standard approach to the frequency dependent inductance and resistance calculation, in Section 3 we describe the mesh formulation approach, and in Section 4 we briefly describe GMRES. Results from FASTHENRY are given in Section 5, followed by conclusions and acknowledgments.

## 2. Inductance calculation

One approach to computing the frequency dependent inductance and resistance matrix, denoted  $Z_r$ , associated with the terminal behavior of a collection of conductors involves first approximating each conductor as a set of piecewise straight conducting sections. The volume of each straight section is then discretized into a collection of parallel thin filaments through which current is assumed to flow uniformly. The interconnection of these current filaments can be represented with a planar graph, where the  $n$  nodes in the graph are associated with connection points between conductor segments, and the  $b$  branches in the graph represent the current filaments into which each conductor segment is discretized.

To derive a system of equations from which the resistance and inductance matrix can be deduced, we start by assuming the applied currents and voltages are sinusoidal, and that the system is in sinusoidal steady-state. Following the partial inductance approach in [1, 2], the branch current phasors can be related to branch voltage phasors (hereafter, phasors will be assumed and not restated) by

$$ZI_b = V_b, \quad (1)$$

where  $V_b$ ,  $I_b \in C^b$ ,  $b$  is the number of branches (number of current filaments), and  $Z \in C^{b \times b}$  is the complex impedance matrix given by

$$Z = R + j\omega L, \quad (2)$$

where  $\omega$  is excitation frequency. The entries of the diagonal matrix  $R \in \Re^{b \times b}$  represent the dc resistance of each current filament, and  $L \in \Re^{b \times b}$  is the dense matrix of partial inductances [3]. Specifically,

$$L_{i,j} = \frac{\mu_0}{4\pi a_i a_j} \int_{\text{filament}_i} \int_{\text{filament}_j} \frac{l_i(X_i) \cdot l_j(X_j)}{|X_i - X_j|} d^3x_i d^3x_j, \quad (3)$$

where  $X_i, X_j \in \mathbb{R}^3$  are the positions in filament  $i$  and  $j$  respectively,  $l_i, l_j$  are the unit vectors in the direction of current flow in filaments  $i$  and  $j$ , and  $a_i, a_j$  are the filament cross sectional areas.

The statement that the branch currents must satisfy Kirchhoff's current law, that is, the currents entering each node must sum to zero, can be written using the branch incidence matrix as

$$AI_b = I_s, \quad (4)$$

where  $I_s \in C^n$  is the mostly zero vector of source currents,  $n$  is the number of nodes (points where conductor sections meet or a conductor terminates) excluding any reference or ground nodes, and  $A \in \mathbb{R}^{b \times n}$  is the branch incidence matrix.

The node voltages can be related to the branch voltages by

$$A^t V_n = V_b, \quad (5)$$

where  $A^t$  is the transpose of the branch incidence matrix, and  $V_n \in C^n$  is the vector of  $n$  referenced node voltages. Combining (5) with (4) and (1) yields

$$AZ^{-1}A^t V_n = I_s. \quad (6)$$

The complex impedance matrix which describes the terminal behavior of the conductor system,  $Z_r$ , can be derived from (6) by noting that

$$Z_r \tilde{I}_s = \tilde{V}_s, \quad (7)$$

where  $\tilde{I}_s$  and  $\tilde{V}_s$  are the vectors of source currents and voltages. Therefore, the  $i^{\text{th}}$  column of  $Z_r$  can be computed by solving (6) with an  $I_s$  whose only nonzero entry corresponds to  $\tilde{I}_{s_i}$ , and then extracting the elements of  $V_n$  corresponding to  $\tilde{V}_s$ .

In most programs, the dense matrix problem in (6) is solved with some form of Gaussian elimination, and this implies that the calculation grows as  $b^3$ , where again  $b$  is the number of current filaments into which the system of conductors is discretized [5]. For complicated packaging structures,  $b$  can exceed ten thousand, and solving (6) with Gaussian elimination can take days, even using a high performance scientific workstation.

### 3. Mesh current approach

The approach to calculating the frequency dependent inductance and resistance matrix described above has some disadvantages if (6) is to be solved with an iterative method. It is difficult to apply the iterative method, because the matrix  $AZ^{-1}A^t$  contains  $Z^{-1}$ , which can only be computed by forming the dense matrix  $Z$ , and then somehow inverting it. Another approach to generating a system of equations for the currents and voltages in the network representing the

conductor system discretization is mesh analysis [4], and the mesh approach has some advantages which will be made clear below.

To begin, mesh analysis is easiest to describe if it is assumed that the sources attached to the conductor system's terminals generate explicit branches in the graph representing the discretized problem. Kirchhoff's voltage law, which implies that the sum of branch voltages around each mesh in the network (a mesh is any loop of branches in the graph which does not enclose any other branches) is represented by

$$MV_b = V_s \quad (8)$$

where  $V_b$  is the vector of voltages across each branch except for the source branches,  $V_s$  is the mostly zero vector of source branch voltages, and  $M \in \mathbb{R}^{b \times m}$  is the mesh matrix, where  $m = b - s + c$ ,  $s$  is the number of conductor sections and  $c$  is the number of conductors. The  $M$  matrix has the property that all of its nonzero entries are +1 or -1 and that most of its rows ( $m - c$  of them) have no more than two nonzero entries.

The relationship between branch currents and branch voltages given in (1) still holds, and the mesh currents, that is, the currents around each mesh loop, satisfy

$$M^t I_m = I_b, \quad (9)$$

where  $I_m \in C^m$  is the vector of mesh currents. Note that one of the entries in the mesh current vector will be identically equal to the source branch current. Combining (9) with (8) and (1) yields

$$M Z M^t I_m = V_s. \quad (10)$$

The matrix  $M Z M^t$  is *easily* constructed directly. To compute the  $i^{th}$  column of the reduced admittance matrix,  $Y = Z_r^{-1}$ , solve (10) with a  $V_s$  whose only nonzero entry corresponds to  $\tilde{V}_{s_i}$ , and then extract the entries of  $I_m$  associated with the source branches.

#### 4. Using an iterative solver

The standard approach to solving the complex linear system in (10) is Gaussian elimination, but the cost is  $m^3$  operations. For this reason, inductance extraction of packages requiring more than a few thousand filaments is considered computationally intractable. To improve the situation, consider using a conjugate-residual style iterative method like GMRES [7]. Such methods have the general form given in Algorithm 1.

Note that the GMRES algorithm can be directly applied to solving (10), because the matrix  $M Z M^t$  is easily constructed explicitly. This is not the case for (6). Just to form (6), the  $Z$  matrix must first be inverted. This suggests that either some kind of nested GMRES algorithm would be required to solve (6)

**Algorithm 1 (GMRES Algorithm for  $Ax = b$ )**

```

guess  $\vec{x}^0$ 
for  $k = 0, 1, \dots$  until converged {
    Compute the error,  $\vec{r}^k = \vec{b} - A\vec{x}^k$ 
    Find  $\vec{x}^{k+1}$  to minimize  $\vec{r}^{k+1}$ 
        based on  $\vec{x}^i$  and  $\vec{r}^i$ ,  $i = 0, \dots, k$ 
}

```

iteratively, or the matrix would have to be expanded into the sparse tableau, and the GMRES algorithm applied to solving that expanded matrix.

## 5. Accelerating iteration convergence

In general, the GMRES iterative method applied to solving (10) can be significantly accelerated by *preconditioning* if there is an easily computed good approximation to the inverse of  $MZM^t$ . We denote the approximation to  $(MZM^t)^{-1}$  by  $P$ , in which case preconditioning the GMRES algorithm is equivalent to using GMRES to solve

$$P(MZM^t)I_m = PV_s \quad (11)$$

for the unknown vector  $I_m$ . Clearly, if  $P$  is precisely  $(MZM^t)^{-1}$ , then (11) is trivial to solve, but then  $P$  will be very expensive to compute.

A good approximation to  $(MZM^t)^{-1}$  that is easily computed can be derived by exploiting the fact that a mesh current in a given conductor is tightly coupled to the other mesh currents within that conductor. With an appropriate numbering of the mesh currents, the interactions between meshes of the same conductor can be clustered into blocks along the diagonal of  $MZM^t$ . The inverse of the block diagonal matrix so generated is then an approximation to  $(MZM^t)^{-1}$ .

Preconditioning with this simple preconditioner proved extremely effective. Table 1 shows the average number of iterations with  $tol = 10^{-3}$  for a single solve of the pin package example described in the next section. The number of iterations required by GMRES without the preconditioner increased rapidly with problem size, but with the preconditioner, the iterations remained constant. This result easily makes up for the small cost of calculating the preconditioner.

## 6. Results

In this section we present our results from FASTHENRY. To test the mesh formulation approach, we began with two parallel rectangular wires and compared the results to those in [8]. As a more interesting example, FASTHENRY was used to analyze 35 pins of a 68-pin package from Digital Equipment Corpo-

Size of $MZM^t (m)$	Iterations WITH precond.	Iterations WITHOUT precond.
210	8	20
560	8	38
910	8	60
1435	8	123
1960	8	164

Table 1. Comparison of the average number of iterations per conductor with and without the preconditioner.

Filaments per conductor section	Size of $MZM^t (m)$	Solution time, direct inversion	Solution time, preconditioned GMRES
1	35	0.0003	0.007
2	210	0.339	0.147
4	560	8.02	1.08
6	910	35.9	3.08
9	1435	135	7.85
12	1960	344	14.4

Table 2. Execution time comparison for the pin package example. Execution times are in IBM RS6000/540 CPU minutes.

ration. To demonstrate the effectiveness of preconditioned GMRES, times are compared against direct inversion for finer and finer spatial discretization.

For the rectangular wire problem, we considered two parallel copper wires with a 2 mm by 2 mm cross section and 4 mm separation between their centers. The problem is treated as a one conductor problem, with one wire acting as the return path. The data in [8] is for a two dimensional analysis with wires of infinite length, so for this problem the lengths were chosen to be 100 meters and the results scaled appropriately.

To observe how the results varied due to skin and proximity effects, the conductor was divided into various numbers of parallel thin filaments. To follow the decaying nature of the skin effect, the dimensions of the filaments were chosen to decrease geometrically toward the outer edge of the conductor. FASTHENRY was run with 1, 4, 25, and 100 filaments per wire with the results compared in Figures 1 and 2 against those from [8]. For each case, as the skin depth became much smaller than the smallest filament, the calculated resistance and inductance stopped changing with frequency, as expected. For relatively few filaments per section (e.g. 25) one can determine much of the nature of the impedance. Both the resistance and inductance are accurately determined up to around  $10^5$  Hz. The resistance results accurately determine the knee of the

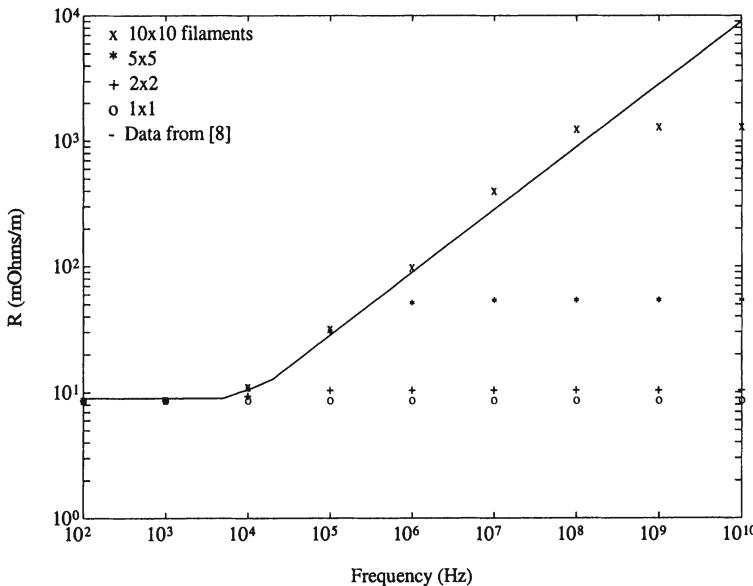


Figure 1. Resistance of Two Long Rectangular Wires.

resistance curve, which shows the beginning of the skin effect. Since it is well known that the resistance increases as the square root of the frequency after the knee, higher frequency resistance values could easily be extracted.

Thirty-five pins of a 68-pin package from DEC, shown in Figure 3, proved a good test of the utility of FASTHENRY. Each pin consists of five conductor sections. Neglecting skin effects and choosing one filament (or branch) per section gives a system of 175 filaments and 35 meshes. Notice that for mesh analysis, the solution is obtained by solving only a 35x35 system, while for the branch incidence matrix approach as in (6) inversion of a 175x175 system is necessary.

To accurately model skin and proximity effects, each conductor section is divided into multiple filaments. As the discretization is refined, the size of the problem grows quickly. For these problems, the advantage of the GMRES algorithm becomes apparent (see Table 2). For twelve filaments per section, GMRES is already 23 times faster than direct inversion. In fact, the number of operations required by GMRES grows only as  $m^2$ , while the direct method grows as  $m^3$ .

Notice that twelve sections per filament is barely enough to observe skin effects, however memory requirements limit any finer discretizations. It is worth noting that future work to implement multipole algorithms [9] will further reduce both the computational costs and memory requirements, thus allowing finer discretizations.

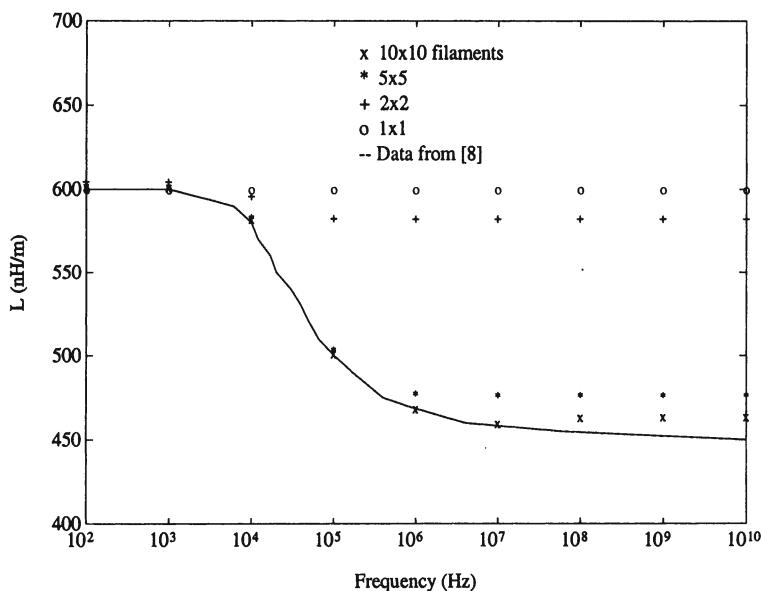


Figure 2. Inductance of Two Long Rectangular Wires.

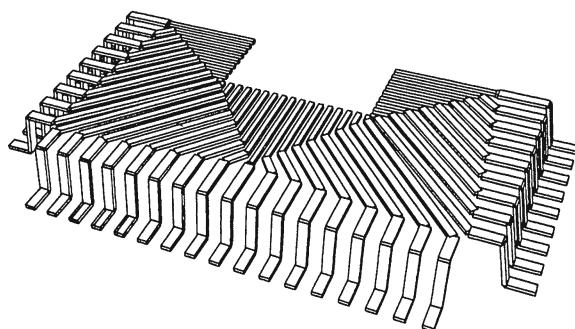


Figure 3. Half of a pin-connect structure. Thirty-five pins shown.

In this paper, we do not give a comparison to the branch incidence approach in (6) since it would require inverting  $Z$ , the branch impedance matrix.  $Z$  is always larger than  $MZM^t$  and would thus be more expensive than direct inversion of  $MZM^t$  which is shown in Table 2.

## 7. Conclusions and acknowledgments

It is shown in this paper that if mesh rather than nodal analysis is used to form the system of equations which must be solved to determine inductance, then the equations can be solved easily with the iterative GMRES algorithm. Addition of the preconditioner to GMRES can reduce the cost of solution to  $m^2$  operations compared to  $m^3$  for direct inversion. Results from FASTHENRY, our 3-D inductance extraction program, demonstrate that the iterative approach can accelerate solution times by more than an order of magnitude.

Future work using multipole algorithms will exploit the fact that the off-diagonal elements of  $Z$  are the partial inductances generated from integrals of  $\frac{1}{r}$ . Such methods will avoid forming and storing most of the entries in the dense matrix  $MZM^t$ , and reduce the cost of calculating matrix-vector products required for the GMRES procedure to order  $b$  operations.

Currently, FASTHENRY is being extended to include ground planes. Results will be presented at the conference.

The authors would like to thank Keith Nabors, Songmin Kim, Dr. Sami Ali, and Joel Phillips for their help in understanding inductance. In addition, the authors would like to thank Dr. Albert Ruehli, Prof. Raj Mittra, and Dr. Colin Gordon for their helpful suggestions.

This work was supported by the Defense Advanced Research Projects Agency contract N00014-91-J1698, the National Science Foundation contract (MIP-8858764 A02), a National Science Foundation fellowship, and grants from Digital Equipment Corporation and I.B.M.

A more complete description of this work appeared later as a full journal article [10].

## References

- [1] W. T. Weeks, L. L. Wu, M. F. McAllister, and A. Singh, "Resistive and inductive skin effect in rectangular conductors," *IBM Journal of Res. and Develop.*, vol. 23, pp. 652-660, November 1979.
- [2] A. E. Ruehli, "Survey of computer-aided electrical analysis of integrated circuit interconnections," *IBM Journal of Research and Development*, vol. 23, pp. 626-639, November 1979.
- [3] P. A. Brennan, N. Raver, and A. Ruehli, "Three dimensional inductance computations with partial element equivalent circuits," *IBM Journal of Res. and Develop.*, vol. 23, pp. 661-668, November 1979.
- [4] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory*. New York: McGraw-Hill, 1969.

- [5] A. E. Ruehli and P. A. Brennan, "Efficient capacitance calculations for three-dimensional multiconductor systems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 21, pp. 76–82, February 1973.
- [6] S. M. Rao, T. K. Sarkar, and R. F. Harrington, "The electrostatic field of conducting bodies in multiple dielectric media," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-32, pp. 1441–1448, November 1984.
- [7] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, pp. 856–869, July 1986.
- [8] M. J. Tsuk, "Propagation and Interference in Lossy Microelectronic Integrated Circuits," PhD Thesis, Massachusetts Institute of Technology, June 1990.
- [9] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, pp. 325–348, December 1987.
- [10] M. Kamon, M. J. Tsuk, and J. White, "Fasthenry: A multipole-accelerated 3-d inductance extraction program", *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 9, pp. 1750–1758, September 1994.

# TIME-DOMAIN NON-MONTE CARLO NOISE SIMULATION FOR NONLINEAR DYNAMIC CIRCUITS WITH ARBITRARY EXCITATIONS

Alper Demir<sup>1</sup>, Edward W.Y. Liu and Alberto L. Sangiovanni-Vincentelli

*Department of Electrical Engineering & Computer Sciences*

*University of California*

*Berkeley, CA 94720, USA*

## Abstract

A new, time-domain, non-Monte Carlo method for computer simulation of electrical noise in nonlinear dynamic circuits with arbitrary excitations is presented. This time-domain noise simulation method is based on the results from the theory of stochastic differential equations. The noise simulation method is general in the sense that any nonlinear dynamic circuit with any kind of excitation, which can be simulated by the transient analysis routine in a circuit simulator, can be simulated by our noise simulator in time-domain to produce the noise variances and covariances of circuit variables as a function of time, provided that noise models for the devices in the circuit are available. Noise correlations between circuit variables at different time points can also be calculated. Previous work on computer simulation of noise in integrated circuits is reviewed with comparisons to our method. Shot, thermal and flicker noise models for integrated-circuit devices, in the context of our time-domain noise simulation method, are described. The implementation of this noise simulation method in a circuit simulator (SPICE) is described. Two examples of noise simulation (a CMOS ring-oscillator and a BJT active mixer) are given.

## 1. Introduction

This paper presents a new, time-domain, non-Monte Carlo method for computer simulation of electrical noise in nonlinear dynamic circuits with arbitrary excitations. This time-domain noise simulation method is based on the results from the theory of stochastic differential equations. The noise phenomena considered in this work are caused by the small current and voltage fluctuations that are generated within the integrated-circuit devices themselves. The existence of noise is basically due to the fact that electrical charge is not continuous but is carried in discrete amounts equal to the electron charge. Electrical noise is associated with fundamental processes in integrated-circuit devices [1]. Noise represents a lower limit to the size of electrical signal that can be amplified by a circuit without significant deterioration in signal quantity. It also results in an

---

<sup>1</sup>Author is currently with Koç University, Istanbul, Turkey.

upper limit to the useful gain of an amplifier, because if the gain is increased without limit, the output stages of the circuit will eventually begin to cut off or saturate on the amplified noise from the input stages [1]. The influence of noise on the performance is not limited to amplifier circuits. For instance, active integrated mixer circuits, which are widely used for down conversion in UHF and microwave receivers, add noise to their output. It is desirable to be able to predict the noise performance of a given mixer design [2, 3]. Most of the time, amplifier circuits operate in small-signal conditions, that is, the operating point of the circuit does not change. For analysis and simulation, the amplifier circuit with a fixed operating-point can be modeled as a linear time-invariant network by making use of the small-signal models of the integrated-circuit devices. On the other hand, for a mixer circuit, the presence of a large local-oscillator signal causes substantial change in the active devices operating points over time. So, a linear time-invariant network model is not accurate for a mixer circuit. There are many other kinds of circuits which do not operate in small-signal conditions, such as a volt-age-controlled-oscillator (VCO) composed of delay cells in a ring configuration. Noise simulation of these circuits requires a method which can handle nonlinear dynamic circuits with arbitrary excitations. The three important types of noise in integrated circuits are shot noise, thermal noise and flicker noise which will all be considered in this work.

In Section 2 below, previous work on computer simulation of noise in integrated circuits is reviewed with comparisons to our method. In Section 3, shot, thermal and flicker noise models for integrated-circuit devices, in the context of our time-domain noise simulation method, are described. Section 4 describes our noise simulation method. In Section 5, the implementation of the noise simulation method, in the context of a nodal-analysis circuit simulation program (SPICE), is described. Two examples of noise simulation are presented in Section 6. Finally, future work is stated in Section 7.

## 2. Previous work

The electrical noise sources in passive elements and integrated-circuit devices have been investigated extensively. Small-signal equivalent circuits, including noise, for many integrated-circuit components have been constructed [1]. The noise performance of a circuit can be analyzed in terms of these small-signal equivalent circuits by performing sinusoidal circuit analysis in frequency domain in the usual fashion. This analysis is done separately for each of the uncorrelated noise sources, and for a range of frequencies. For a complicated circuit, the large number of noise sources and circuit complexity completely preclude hand calculation. In fact, even machine computation of the noise contributions from all noise sources can be time consuming. Fortunately, an extremely efficient computational technique, based on the inter-reciprocal adjoint network

concept, was proposed [4, 5]. This technique calculates the noise contribution from an arbitrarily large number of noise sources at a given frequency with little more computer time than is normally required for a single noise source. The noise analysis in SPICE is based on this method. Unfortunately, this method is only applicable to linear time-invariant circuits (e.g. the small-signal equivalent circuits corresponding to circuits with fixed operating points). It is not appropriate for noise simulation of circuits with changing bias conditions, or circuits which are not meant to operate in small-signal conditions.

[2, 3] and [6] present noise analysis techniques for nonlinear circuits with a periodic large signal excitation. The noise analysis for a nonlinear circuit with a periodic large signal excitation reduces to the analysis of a linear periodically time-varying circuit with cyclostationary [2, 3][6] noise sources. This is arrived by a first-order Taylor's expansion of the circuit equations around the periodic steady-state solution of the circuit without the noise sources and the small-signal excitations. This Taylor's approximation is similar to the one we will present in Section 4.1. The noise analysis methods described in [2, 3] and [6] use frequency-domain methods based on manipulating impulse responses and transfer functions for a linear periodically time-varying system, and spectral densities for cyclostationary noise sources. These noise analysis techniques are applicable to only a limited class of nonlinear circuits with two excitations, where one of the excitations is large and periodic and the other is small (e.g., mixer circuits, switched capacitor circuits). The previous work on noise simulation in time-domain is restricted to techniques which employ the Monte Carlo method [7]. This method has several drawbacks. Pseudo-random number generators often do not generate a large sequence of independent numbers, but reuse old random numbers instead. This becomes a problem if a circuit with many noise sources is simulated. This is usually the case, because every device has several noise sources associated with its model. In this method, the same circuit is simulated many times by obtaining "different" sample paths for each noise source. Then a statistical analysis is carried out to calculate averages and variances over these many simulations. The noise content in a waveform will be much smaller when compared with the magnitude of the waveform itself. As a result, the waveforms obtained for different sample paths of noise generators will be very close to each other. It is known that, in a simulator, these waveforms are only numerical approximations to the actual waveforms, therefore they contain numerical noise. The RMS value of noise is calculated by taking a difference of these waveforms. That is, two large numbers, which have uncertainty in them, are being subtracted from each other. Consequently, the RMS noise calculated with this method, in fact, includes the noise generated by the numerical algorithms. This furthermore degrades the accuracy of the results obtained by this method. This method has one advantage when compared with the frequency domain methods discussed above: It is not restricted to linear

time-invariant, or to nonlinear circuits with a large signal periodic excitation. In theory, it is applicable to the general class of nonlinear dynamic circuits with any kind of excitation.

Our method, unlike the frequency domain methods, is not restricted to linear time-invariant or nonlinear circuits with a large signal periodic excitation. Our time-domain noise simulation method is based on the results from the theory of stochastic differential equations. There are no pseudo-random number generators involved in the simulation, therefore the problems associated with them do not exist. The simulation of the average waveforms (without noise in the circuit) and the simulation of noise are separated, even though they are done concurrently. Thus, the numerical noise problem that arises in Monte Carlo methods is avoided. Our method is capable of calculating variances and covariances (that is, the covariance matrix) for the noise content in the node voltages and other circuit variables in a circuit as a function of time. Furthermore, correlations between circuit variables at different time points can also be calculated. Finally, the implementation of our method fits naturally into a circuit simulator (such as SPICE) which is capable of doing time-domain transient simulations. Noise simulation is done along with the transient simulation over the time interval specified by the user.

### 3. Noise models

The electrical noise sources in passive elements and integrated-circuit devices have been investigated extensively, and appropriate models have been derived [1, 8]. Traditionally, these noise models are presented as stationary noise sources in the small-signal equivalent (at an operating point) circuits of the devices [1]. In this section, we describe the adaptation of these noise models for use in our time-domain noise simulation method. In our method, the noise sources are inserted in the large-signal models of the integrated-circuit devices and they are, in general, non-stationary. In Section 3.1, the adaptation of shot, thermal and flicker noise models for resistors and junction diodes will be described. The noise models for these two simple devices are representative of noise models for all other integrated-circuit devices such as BJTs and MOSFETs, because all kinds of noise we consider (shot, thermal and flicker noise) exist in these devices [16]. The noise source models we use in our method are adapted from [1] and [8]. As it will become clear in Section 4, our noise simulation method requires that noise sources are white. The thermal and shot noise sources are modeled as white noise sources, hence they can be directly included in the simulation. However, the flicker noise sources can not be included in the simulation as they are. The inclusion of flicker noise sources into the noise simulation method will be described in Section 3.2.

### 3.1 Shot, thermal and flicker noise models

**3.1.1 Resistors.** Monolithic and thin-film resistors display thermal noise. The thermal noise in a resistor can be modeled by a white Gaussian noise current source with intensity

$$IN_{thermal}^R = \sqrt{2kT/R} \quad (1)$$

where  $k$  is Boltzmann's constant,  $T$  is the absolute temperature and  $R$  is the resistance [1]. The thermal noise source associated with a resistor is a stationary white noise process, assuming that the resistance value is a constant as a function of time. The intensity of a stationary white Gaussian noise process is equal to the square root of the power spectral density. For a stationary white Gaussian noise process, the power spectral density (a function of frequency) is a constant on the entire real axis.

**3.1.2 Junction diodes.** The series resistance , in the model of a junction diode [1], is a physical resistor due to the resistivity of silicon, hence it exhibits thermal noise. The thermal noise in can be modeled as in Section 3.1.1. The exhibits shot noise which is associated with the current flow through the diode. The intensity of the shot noise current, which is white Gaussian, is given by

$$IN_{shot}^D = \sqrt{qI_D(t)} \quad (2)$$

where  $q$  is the electronic charge ( $1.6 \times 10^{-19}$ ) and  $I_D(t)$  is the noiseless diode current. Note that, in this case, intensity is a function of time, hence this white noise source is not stationary. The square of the time-varying intensity for a non-stationary white noise source as above can be thought to be the time-varying power spectral density, which is a constant (as a function of frequency) on the entire real axis. During nonlinear operation, the current through the diode shows variations as a function of time, so does the intensity. In this way, shot noise associated with a time-varying current is modeled as a non-stationary white Gaussian noise, which is also the case for thermal noise associated with a time-varying resistance. (1) is also valid for a time-varying resistance [16]. The flicker noise source in a diode is modeled by a non-stationary noise process which has a time-varying power spectral density given by

$$S_{flicker}^D = KFI_D(t)^a/f \quad (3)$$

where  $KF$  is a constant for a particular device,  $a$  is a constant in the range 0.5 to 2 and  $f$  is the frequency. This noise source can not be included in the noise simulation directly, because it is not white (i.e. the time-varying power spectral density is not a constant as a function of frequency). A way of synthesizing this source from white noise sources will be discussed in Section 3.2.

### 3.2 Flicker noise sources

In our noise simulation method, only white noise sources are allowed. Flicker noise sources have a power spectral density which is not a constant as a function of frequency. The natural way to include flicker noise sources into simulation is, somehow, to synthesize them using white noise sources. A promising approach for  $1/f$  (flicker) noise generation is to use the summation of Lorentzian spectra which is defined by (4) [9]. It has been shown that a constant distribution of 1.4 poles per decade gives a  $1/f$  spectrum with less than 1% error [9]. A sum of  $N$  Lorentzian spectra is given by

$$S(f) = \frac{2\sigma^2}{\pi} \sum_{h=1}^N \frac{\Phi_h}{\Phi_h^2 + f^2} \quad (4)$$

where  $\Phi_h$ s designate the pole-frequencies and  $f$  is the frequency. It has been shown in [9] that  $N = 20$  poles uniformly distributed over 14 decades are sufficient to generate  $1/f$  noise over 10 decades with a maximum error less than 1%. Each Lorentzian spectrum in the summation in (4) can be easily obtained by using the thermal noise generator of a resistor  $R_h$  connected in parallel to a capacitance  $C_h = C$ , and their sum can be achieved by putting  $N$  of such  $R_h - C_h$  groups (Figure 1) in series [9]. In the noise simulation, a flicker noise source in

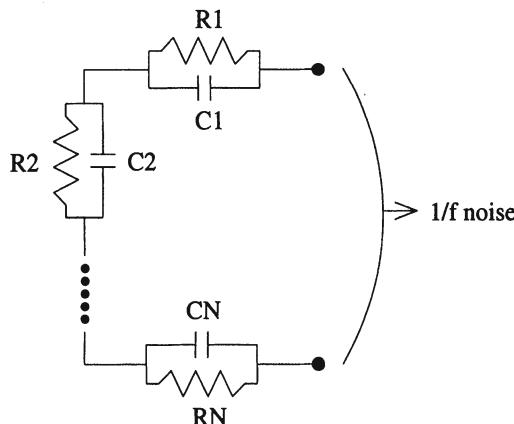


Figure 1. Noise Synthesizing Circuit.

the model of an integrated- circuit device is built by using the circuit in Figure 1 with an ideal voltage-controlled current source. This is illustrated in Figure 2. The voltage-controlled current source is connected between the two nodes of a device where the flicker noise source is modeled. The spectral density of the noise obtained from the circuit in Figure 1 is approximately

$$S(f) = 2\sigma^2/\pi f \quad (5)$$

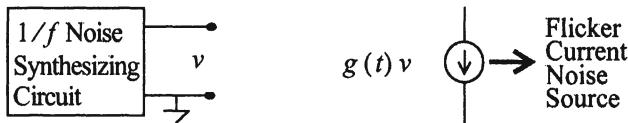


Figure 2. Flicker Current Noise Source Synthesis.

where  $\sigma^2 = kT/(2C)$ . This spectral density is time-invariant. The flicker noise model given in Section 3.1.2 requires a time-varying spectral density. This is achieved by having a time-varying transconductance ( $g(t)$ ) for the voltage-controlled current source in Figure 1. For instance, for a diode, we require that the flicker noise source spectral density is in the form given by (3). This is assured with

$$g(t) = \sqrt{\pi KFI_D(t)^a / (2\sigma^2)} \quad (6)$$

## 4. Development of the simulation method

The noise simulation method will be described assuming that modified nodal analysis (MNA) [10] is used for the formulation of circuit equations. MNA is the method for circuit equation formulation in most of the circuit simulators (such as SPICE) available. Translation of the noise simulation method into other ways of circuit equation formulation is straightforward.

### 4.1 Derivation of the stochastic differential equation for noise from MNA formulation of the nonlinear circuit equations

The MNA equations for any circuit, *without the noise sources*, can be written compactly as

$$F(\dot{x}, x, t) = 0 \quad x(0) = x_0 \quad (7)$$

where  $x$  is the vector of the circuit variables with dimension  $n$ ,  $\dot{x}$  is the time derivative of  $x$ ,  $t$  is time and  $F$  is mapping  $x$ ,  $\dot{x}$  and  $t$  into a vector of real numbers of dimension  $n$ . The time dependence of  $x$  and  $\dot{x}$  will not be written explicitly for notational simplicity. In MNA, the circuit variables consist of node voltages and some branch currents, e.g. currents through inductors and voltage sources. The circuit equations consist of the node equations (KCL) and branch equations of the elements for which branch currents are included in the circuit variables vector. Under some rather mild conditions (which are satisfied by well modeled circuits) on the continuity and differentiability of  $F$ , it can be proven that there exists a unique solution to (7) assuming that a fixed initial value  $x(0) = x_0$  is given [10]. Let  $x_s$  be the solution to (7). Transient analysis in circuit simula-

tors solves for  $x_s$  using numerical methods for ordinary differential equations (ODEs) [10]. The initial value vector is obtained by a DC analysis of the circuit before the transient simulation is started. For a circuit, there may be several different DC operating-points.

The first-order Taylor's expansion of  $F$  around  $x_s$  is expressed as

$$\begin{aligned} F(\dot{x}, x, t) \cong & F(\dot{x}_s, x_s, t) + \frac{\partial}{\partial x} F(\dot{x}, x, t) \Big|_{\begin{array}{l} x = x_s \\ \dot{x} = \dot{x}_s \end{array}} (x - x_s) \\ & + \frac{\partial}{\partial \dot{x}} F(\dot{x}, x, t) \Big|_{\begin{array}{l} x = x_s \\ \dot{x} = \dot{x}_s \end{array}} (\dot{x} - \dot{x}_s) \end{aligned} \quad (8)$$

which will be used later.

If the noise sources are included in the circuit, the MNA formulation of the circuit equations can be written as

$$F(\dot{x}, x, t) + B(x, t)v = 0 \quad x(0) = x_0 + x_{noise,0} \quad (9)$$

where  $B(x, t)$  is an  $n \times p$  matrix, the entries of which are a function of  $x$ , and  $v$  is a vector of  $p$  standard white Gaussian stochastic processes. A one-dimensional standard Gaussian white noise is a *stationary* Gaussian process  $\xi(t)$ , for  $-\infty < t < \infty$ , with mean  $E[\xi(t)] = 0$  and a constant spectral density on the entire real axis. The autocorrelation function of  $\xi(t)$  is given by  $E[\xi(t + \tau)\xi(t)] = \delta(\tau)$ , where  $\delta$  is Dirac's delta function [11].

The white Gaussian noise  $\xi(t)$  is a very useful mathematical idealization for describing random influences that fluctuate rapidly and hence are virtually uncorrelated for different instants of time. A white Gaussian noise model is appropriate for thermal and shot noise in integrated circuits [1]. Flicker noise sources are taken care of in the way described in Section 3.2.  $v$  in (9) is simply a combination of  $p$  independent one-dimensional white Gaussian noise processes as defined above. These noise processes actually correspond to the current noise sources which are included in the models of the integrated-circuit devices. Since the noise models for the integrated-circuit devices are to be employed here in the context of an MNA circuit simulator (SPICE), noise sources in the devices are all modeled as uncorrelated current sources.

$B(x, t)$ , in (9), contains the intensities, as described in Section 3.1, for the white noise sources in  $v$ . The intensities for these noise sources are, in general, a function of time (not a constant). Because of intensity variations, these noise sources are not stationary. Thus, the non-stationarity of the noise sources in the circuit are captured in  $B(x, t)$ . Every column in  $B(x, t)$  corresponds to a noise source in  $v$  and has either one or two nonzero entries [16].

(9) is a system of nonlinear stochastic differential equations (SDEs) where the forcing is an irregular stochastic process (white noise). This kind of SDEs

require fundamentally different and complex methods of analysis and numerical solution [12]. Fortunately, some characteristics of our problem help us simplify the numerical solution of (9): The noise content in the signals in any useful circuit is, almost always, much smaller when compared with the signal itself.

Let  $x_{sn}$  be the solution of (9).  $x_{sn}$  is a vector of stochastic processes, since it is the solution of the circuit equations with the noise sources included, and satisfies

$$F(\dot{x}_{sn}, x_{sn}, t) + B(x_{sn}, t) v = 0 \quad x_{sn}(0) = x_0 + x_{noise,0} \quad (10)$$

where  $x_0$  is deterministic, and  $x_{noise,0}$  is a vector of  $n$  zero-mean random variables. We use (8) in (10) to approximate  $F(\dot{x}_{sn}, x_{sn}, t)$ , and obtain

$$\begin{aligned} & F(\dot{x}_s, x_s, t) + \frac{\partial}{\partial x} F(\dot{x}, x, t) \Big|_{\substack{x=x_s \\ \dot{x}=\dot{x}_s}} (x_{sn} - x_s) \\ & + \frac{\partial}{\partial \dot{x}} F(\dot{x}, x, t) \Big|_{\substack{x=x_s \\ \dot{x}=\dot{x}_s}} (\dot{x}_{sn} - \dot{x}_s) + B(x_{sn}, t) v \cong 0 \end{aligned} \quad (11)$$

$$x_{sn}(0) = x_0 + x_{noise,0}$$

Define

$$x_{noise} = x_{sn} - x_s. \quad (12)$$

$x_{noise}$  is the difference between the solutions of the circuit equations, with and without the noise sources. In other words,  $x_{noise}$  is the noise content in  $x_{sn}$ .  $x_{noise}$  is much smaller when compared with  $x_s$ , which validates the above approximation.

For notational simplicity, define

$$A(t) = \frac{\partial}{\partial x} F(\dot{x}, x, t) \Big|_{\substack{x=x_s \\ \dot{x}=\dot{x}_s}} \quad C(t) = \frac{\partial}{\partial \dot{x}} F(\dot{x}, x, t) \Big|_{\substack{x=x_s \\ \dot{x}=\dot{x}_s}} \quad (13)$$

where  $A(t)$  and  $C(t)$  are  $n \times n$  matrices with time-dependent entries. Furthermore, we approximate

$$B(x_{sn}, t) \cong B(x_s, t) \quad (14)$$

and define

$$B(t) = B(x_s, t). \quad (15)$$

If (12), (13), (14) and (15) are substituted in (11) we obtain

$$\begin{aligned} & F(\dot{x}_s, x_s, t) + A(t) x_{noise} + C(t) \dot{x}_{noise} + B(t) v \cong 0 \\ & x_{noise}(0) = x_0 + x_{noise,0} - x_s(0). \end{aligned} \quad (16)$$

Since  $x_s$  is the solution of (7) we have

$$F(\dot{x}_s, x_s, t) = 0 \quad x_s(0) = x_0 \quad (17)$$

and if we substitute (17) in (16), we obtain

$$\begin{aligned} A(t)x_{noise} + C(t)\dot{x}_{noise} + B(t)v &\cong 0 \\ x_{noise}(0) &= x_{noise,0}. \end{aligned} \quad (18)$$

(18) is a linear SDE [11] in  $x_{noise}$  with time-varying coefficient matrices.  $A(t)$ ,  $B(t)$  and  $C(t)$  are functions of  $x_s$ , and they do not depend on  $x_{noise}$ . The solution of this equation will be discussed in the next four subsections.

## 4.2 Transformation of the stochastic differential equation for noise into state-equation form

To make use of some of the results from the theory of SDEs, (18) will be put into the form

$$\dot{y} = E(t)y + F(t)v \quad y(0) = y_0. \quad (19)$$

If  $C(t)$  is a full-rank matrix, this can be easily done by premultiplying both sides of (18) by the inverse of  $C(t)$ . However, this is not true in general;  $C(t)$  may have zero rows and columns. For instance, if a circuit variable is a node voltage, and if this node does not have any capacitors connected to it in the circuit, then all of the entries in the column of  $C(t)$  corresponding to this circuit variable will be zero for all  $t$ . Also, the node equation (KCL) corresponding to this node will not contain any time-derivatives, hence the row of  $C(t)$  corresponding to this node equation will be zero for all  $t$ . Some of the rows and columns of  $C(t)$  are structurally zero, independent of  $t$ . Moreover, the number of zero rows is equal to the number of zero columns. If we reorder the variables in  $x_{noise}$  in such a way that the zero columns of  $C(t)$  are grouped at the right-hand side of the matrix, and reorder the equations in such a way that the zero rows of  $C(t)$  are grouped in the lower part of the matrix, (18) becomes

$$\begin{bmatrix} A_{11}(t) & A_{12}(t) \\ A_{21}(t) & A_{22}(t) \end{bmatrix} \begin{bmatrix} x_{noise}^1 \\ x_{noise}^2 \end{bmatrix} + \begin{bmatrix} C_{11}(t) & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_{noise}^1 \\ \dot{x}_{noise}^2 \end{bmatrix} + \begin{bmatrix} B_1(t) \\ B_2(t) \end{bmatrix} v = 0$$

$$\begin{bmatrix} x_{noise}^1(0) \\ x_{noise}^2(0) \end{bmatrix} = \begin{bmatrix} x_{noise,0}^1 \\ x_{noise,0}^2 \end{bmatrix} \quad (20)$$

where  $A_{11}(t)$  and  $C_{11}(t)$  are  $m \times m$ ,  $A_{22}(t)$  is  $k \times k$ ,  $A_{12}(t)$  is  $m \times k$ ,  $A_{21}(t)$  is  $k \times m$ ,  $B_1(t)$  is  $m \times p$ ,  $B_2(t)$  is  $k \times p$ ,  $x_{noise}^1$  is an  $m$ -dimensional vector,  $x_{noise}^2$  is a  $k$ -dimensional vector,  $m$  is the number of nonzero columns (rows) in  $C(t)$  and  $k$  is the number of zero columns (rows). Naturally,  $n = m + k$ . Then, expanding

(20) and performing straightforward operations on this equation [16], we arrive at the SDE for noise in the state equation form, which is given by

$$\dot{x}_{noise}^1 = E(t) x_{noise}^1 + F(t) v \quad x_{noise}^1(0) = x_{noise,0}^1 \quad (21)$$

$$\begin{aligned} x_{noise}^2 &= D_1(t) x_{noise}^1 + D_2(t) v \\ x_{noise,0}^2 &= D_1(0) x_{noise,0}^1 + D_2(0) v(0) \end{aligned} \quad (22)$$

with

$$\begin{bmatrix} x_{noise}^1 \\ x_{noise}^2 \end{bmatrix} = x_{noise} \quad (\text{reordered}) \quad (23)$$

Here,  $E(t)$  is  $m \times m$ ,  $F(t)$  is  $m \times p$ ,  $D_1(t)$  is  $k \times m$ ,  $D_2(t)$  is  $k \times p$  and they are obtained from  $A_{11}(t)$ ,  $A_{12}(t)$ ,  $A_{21}(t)$ ,  $A_{22}(t)$ ,  $C_{11}(t)$ ,  $B_1(t)$ , and  $B_2(t)$  by performing some matrix algebra operations [16].

### 4.3 Solution of the stochastic differential equation for noise

(21) is a linear differential equation where the forcing is an irregular stochastic process which is white noise. A mathematically rigorous treatment of equations of this type requires a new theory. In 1951, Ito defined the Ito or stochastic integral and in doing so put the theory of SDEs on a solid foundation [11]. (21) is written symbolically as a linear SDE, but it is interpreted as an integral equation with Ito or Stratonovich stochastic integrals [11]. The solution of (21) obtained by the Stratonovich interpretation is equal to the one obtained by the Ito interpretation, because it is a linear SDE in the narrow sense [11]. A detailed explanation of Ito and Stratonovich stochastic integrals and stochastic differential equations can be found in [11], [12] and [13]. In the following development, we state and use some of the results from the theory of SDEs. (21) is often written in the form

$$dx_{noise}^1 = E(t) x_{noise}^1 dt + F(t) dw \quad x_{noise}^1(0) = x_{noise,0}^1 \quad (24)$$

where  $w$  is a vector of  $p$  independent one-dimensional Wiener processes. A  $p$ -dimensional Wiener process can be defined as a process with independent and stationary,  $N(0, (t_1 - t_2)I_p)$ -distributed increments  $w(t_1) - w(t_2)$ , with initial value  $w(0) = 0$ . Here,  $N(\text{Mean}, \text{Cov})$  denotes the  $p$ -dimensional normal distribution with expectation vector *Mean* and covariance matrix *Cov* [11]. A Wiener process can be thought to be the “integral” of a white noise, or, alternatively, white noise is the “derivative” of a Wiener process in the sense of coincidence of the covariance functionals [11]. In our case, we have

$$w(t) = \int_0^t v(\tau) d\tau \quad v(t) = \dot{w}(t) \quad (25)$$

As with ordinary differential equations, the general solution of a linear SDE can be found explicitly. The method of solution also involves an integrating factor or, equivalently, a fundamental solution of an associated homogeneous differential equation. The solution of (21) is given by

$$x_{noise}^1(t) = \Phi(t, t_0) x_{noise}^1(t_0) + \int_{t_0}^t \Phi(t, \tau) F(\tau) dw(\tau) \quad (26)$$

where  $\Phi(t, \tau)$  is the matrix determined as a function of  $t$  by the homogeneous differential equation

$$d\Phi/dt = E(t) \Phi, \quad \Phi(\tau, \tau) = I_m. \quad (27)$$

(26) involves an Ito integral as opposed to a Riemann integral [11]. The integral in (26) can not be interpreted as an ordinary Riemann integral, because almost all sample functions of  $w(t)$  are of unbounded variation. Ito's definition of the stochastic integral includes the ordinary Riemann integral as a special case [11]. If the functions  $E(t)$  and  $F(t)$  are "measurable" and bounded on the time interval of interest, there exists a unique solution for every initial value  $x_{noise}^1(0)$  [11]. We are interested in the case where

$$x_{noise}^1(0) = x_{noise,0}. \quad (28)$$

In our problem, it is sufficient to find the probabilistic characteristics of  $x_{noise}^1$  as a function of  $t$ . In other words, we would like to determine the mean and the covariance matrix of  $x_{noise}^1$  as a function of time in the time interval desired. If  $x_{noise}^1$  is a Gaussian stochastic process, then it is completely characterized by its mean and covariance function as a function of time. Further explanation on this topic will be given in Section 4.5. If we substitute (28) in (26) with  $t_0 = 0$  we obtain

$$x_{noise}^1(t) = \Phi(t, 0) x_{noise,0} + \int_0^t \Phi(t, \tau) F(\tau) dw(\tau). \quad (29)$$

If we take the expectation of both sides of (29) we get the mean of  $x_{noise}^1$  which is a function of  $t$ . Noting that  $E[v(t)] = 0$  and  $E[x_{noise,0}^1] = 0$ , we get

$$m^1(t) = E[x_{noise}^1(t)] = 0. \quad (30)$$

Next, we would like to determine the correlation matrix of the components of  $x_{noise}^1$  as a function of  $t$ , which is given by

$$K^1(t) = E[x_{noise}^1(t) x_{noise}^{1T}(t)]. \quad (31)$$

Consider the differential

$$dx_{noise}^1 x_{noise}^{1T} = x_{noise}^1 dx_{noise}^{1T} + (dx_{noise}^1) x_{noise}^{1T} + F(t) F(t)^T dt. \quad (32)$$

Notice that there is an extra term in (32) which would not be there if we were using ordinary calculus instead of stochastic (Ito) calculus. This equation is obtained from Ito's Theorem on stochastic differentials [11] using (24). We use (24) to expand (32) and obtain

$$\begin{aligned} dx_{noise}^1 x_{noise}^{1T} = & (E(t) x_{noise}^1 x_{noise}^{1T} + x_{noise}^1 x_{noise}^{1T} E(t)^T) dt + \\ & F(t) F(t)^T dt + x_{noise}^1 (F(t) dw)^T + (F(t) dw) x_{noise}^{1T}. \end{aligned} \quad (33)$$

If we take the expectation of both sides of this equation, noting that  $x_{noise}^1$  and  $dw$  are uncorrelated and using (31), we get

$$\dot{K}^1(t) = E(t) K^1(t) + K^1(t) E(t)^T + F(t) F(t)^T \quad (34)$$

where  $K^1(t)$  is the unique symmetric nonnegative-definite solution of the matrix equation (34) with the initial value  $K^1(0) = E[x_{noise,0}^1 (x_{noise,0}^1)^T] = K_0^1$ . Calculation of the initial value  $K_0^1$  will be described later. The differential equation for  $\dot{K}^1(t) = K^1(t)^T$ , (34), satisfies the Lipschitz and boundedness conditions in the time interval of interest, so that a unique solution exists [11]. (34) represents (in view of symmetry of  $K^1(t)$ ) a system of  $m(m+1)/2$  linear ordinary differential equations. (34) can be solved for using a numerical method (such as Backward Euler) for the solution of ODEs.

$K^1(t)$  represents the noise correlation matrix of circuit variables as a function of time. So, the information about the noise variances of circuit variables, or the noise correlations between circuit variables at a given time point are contained in  $K^1(t)$ . In some problems, one might be interested in the noise correlations of circuit variables at different time points, which can be expressed as

$$K^1(t_1, t_2) = E[x_{noise}^1(t_1) x_{noise}^1(t_2)^T]. \quad (35)$$

In a similar way to the derivation of (34), one can derive

$$\frac{\partial}{\partial t_2} K^1(t_1, t_2) = K^1(t_1, t_2) E(t_2)^T \quad (36)$$

with the initial condition  $K^1(t_1, t_1) = K^1(t_1)$  [13]. Integrating (36) at various values of  $t_1$ , one can obtain a number of sections of the correlation function  $K^1(t_1, t_2)$  at  $t_2 > t_1$ . Then,  $K^1(t_1, t_2)$  at  $t_2 < t_1$  is determined by

$$K^1(t_1, t_2) = K^1(t_2, t_1)^T. \quad (37)$$

#### 4.4 Calculation of the initial value for the linear ODE for the covariance matrix of the components of $x_{noise}^1$

In the previous section, we have derived a linear ODE, (34), for the correlation matrix of  $x_{noise}^1$ . In order to be able to solve (34), we need to know the initial value  $K_0^1$ . We set  $K_0^1$  to the solution of the following matrix equation in  $P$

$$E(0)P + PE(0)^T + F(0)F(0)^T = 0. \quad (38)$$

The matrix equation (38) has a symmetric nonnegative-definite solution  $P$ , if the equation  $\dot{z} = E(0)z$  is asymptotically stable (that is, if all the eigenvalues of  $E(0)$  have negative real parts) [11]. (38) represents (in view of symmetry of  $P$ ) a system of  $m(m + 1)/2$  linear equations.

It is interesting to analyze the special case of noise simulation when the circuit is linear time-invariant, or nonlinear dynamic with DC excitations [16]. In this case, noise simulation reduces to solving the linear equation system (38) [16].

#### 4.5 The condition for $x_{noise}^1$ to be Gaussian

The noise in the circuit (solution of (21)) is a Gaussian stochastic process if and only if the initial value  $x_{noise,0}^1$  is normally distributed or constant [11]. Up to this point, we have characterized the initial value  $x_{noise,0}^1$  as being an  $m$ -dimensional vector of zero-mean random variables with the covariance matrix given by the solution of (38). Here, we restrict  $x_{noise,0}^1$  to be a vector of zero-mean normally distributed random variables with the covariance matrix given by the solution of (38). With this restriction on the initial value  $x_{noise,0}^1$ ,  $x_{noise}^1$  (solution of (21)) is a vector of Gaussian stochastic processes, non-stationary in general, and it is completely characterized by its mean, (30), and correlation function (given as the solution of (34) and (36) as a function of time). For circuits with time-invariant large-signal waveforms,  $x_{noise}^1$  is a vector of stationary (in the strict sense) Gaussian processes, completely characterized by its covariance matrix (a constant function of time as given by the solution of (38)).

### 5. Implementation in SPICE

The noise simulation method, along with the noise models described, was implemented inside the circuit simulator SPICE3 [14]. Time-domain noise simulation is performed along with the transient simulation in the time interval specified by the user. The transient simulation in SPICE3 solves for  $x_s$ , which is the solution of (7). The initial value vector  $x(0) = x_0$  in (7) is obtained by a DC analysis before the transient simulation is started. The numerical methods for solving (7) subdivide the time interval  $[0, T]$ , in which the transient simulation

is to be performed, into a finite set of distinct points:

$$t_0 = 0, \quad t_R = T, \quad t_{r+1} = t_r + h_{r+1} \quad r = 0, 1, \dots, R \quad (39)$$

where  $h_{r+1}$ s are the time steps. At each time point  $t_r$ , the numerical methods compute an “approximation”  $x_s[r]$  of the exact solution  $x_s(t_r)$  [10].

The noise simulation (numerical solution of (34) and (36)) is performed concurrently with the transient simulation. (34) represents a system of  $m(m+1)/2$  linear differential equations. We currently use the Backward Euler scheme to discretize these equations in time.

At each time point  $t_r$  of the noise simulation, after the transient simulation routines have calculated  $x_s[r]$ , the matrices  $A[r] \cong A(t_r)$ ,  $C[r] \cong C(t_r)$  and  $B[r] \cong B(t_r)$ , as defined by (13) and (15), are calculated using the values in  $x_s[r]$ . These matrices are stored in sparse matrix data structures. The routines for loading these matrices have been written for each device. The routines for loading  $B[r]$  contain the noise models for the devices. Then all the operations described in Section 4.2 are performed to calculate  $E[r] \cong E(t_r)$  and  $F[r] \cong F(t_r)$  from  $A[r]$ ,  $C[r]$  and  $B[r]$ . The numerical operations actually done somewhat differ from what has been described because of efficiency reasons. All of these operations are performed using sparse matrix data structures. Then,  $E[r]$  and  $F[r]$  are used to calculate  $K^1[r] \cong K^1(t_r)$  in the discretized solution of (34) with the Backward Euler scheme. This last operation requires the solution of  $m(m+1)/2$  simultaneous linear equations, because Backward Euler is an implicit method [10]. Here,  $m$  is, roughly, the number of nodes to which a capacitor is connected. Simulations have shown that, for larger circuits, the CPU time spent for this last operation at a time point heavily dominates the CPU time required by the other operations. Most of the CPU time is used for solving systems of linear equations. We currently use a general-purpose, direct method, sparse matrix solver to solve systems of linear equations. With this direct method linear solver, the computational cost of noise simulation is still high for large-scale circuits. Experiments with several circuits have shown that significant speedup can be obtained by using a parallel iterative linear solver (running on a CM-5) [17], especially for larger circuits. CPU times obtained with this parallel iterative solver suggest that even using a sequential version of this iterative solver will reduce the computational cost of noise simulation considerably when compared with the CPU times obtained with the direct solver we currently use.

The operations described in the above paragraph are performed at every time point. Upon completion,  $x_s[r] \cong x_s(t_r)$ ,  $r = 0, \dots, R$  contains the mean waveforms for the circuit variables as a function of time, which is the usual SPICE transient simulation output. And  $K^1[r] \cong K^1(t_r)$ ,  $r = 0, \dots, R$  contains the waveforms for the covariance matrix of the noise contents in the circuit variables, as defined by (31), as a function of time, which is the noise simulation output.

## 6. Noise simulation examples

In this section, we present two examples of noise simulation. In particular, noise simulations for a CMOS ring-oscillator circuit and a BJT active mixer circuit will be presented. For both of these circuits, we have included only the shot and thermal noise sources in the simulation. One reason for this is that flicker noise has little effect on the noise performance of these circuits. Secondly, including the flicker noise sources increases the simulation time because of the extra nodes created for flicker noise source synthesis.

### 6.1 CMOS ring-oscillator

Three CMOS inverters loaded with 1 pF capacitors were connected in a ring-oscillator configuration and a noise simulation was done. In Figure 3, the mean and noise variance of one of the taps of this ring-oscillator can be seen. As seen in Figure 3, the noise at one of the taps of the ring-oscillator is non-stationary, that is, the noise variance is not a constant as a function of time. The noise variance is highest during low-to- high and high-to-low transitions of the tap voltage. Ring-oscillator based VCOs and delay-lines are used in many phase/ delay-locked systems such as clock generators and clock recovery circuits. Phase noise/jitter is a major concern in the design of such systems. Behavioral models which capture noise effects, and behavioral simulation is used to predict the phase noise/jitter performance of these systems [15]. Our transistor-level noise simulator can be used to simulate ring-oscillator VCOs and delay-lines to obtain the timing jitter at the outputs of the delay cells (as well as the correlations between the jitters.) This information is then used in behavioral simulation [15].

### 6.2 BJT Active Mixer

This circuit was obtained from industry sources. It contains 14 BJTs, 21 resistors, 5 capacitors, and 18 parasitic capacitors connected between some of the nodes and ground. The LO (local oscillator) input is a sine-wave at 1.75 GHz with an amplitude of 178 mV. The RF input is a sine-wave at 2 GHz with an amplitude of 31.6 mV. Thus, the IF frequency is 250 MHz.  $1/f$  noise sources are not included in the simulation, because  $1/f$  noise is rarely a factor at RF and microwave frequencies [2]. This circuit was simulated to calculate the noise variance at the output as a function of time. (Figure 4: This waveform is periodic with a period of 4 nsecs: IF frequency is 250 MHz.) The noise at the output of this circuit is not stationary, because the signals applied to the circuit are large enough to change the operating point. The noise analysis of this circuit by assuming a small-signal equivalent circuit around a fixed operating point does not give correct results. Such an analysis would predict the noise at the output as

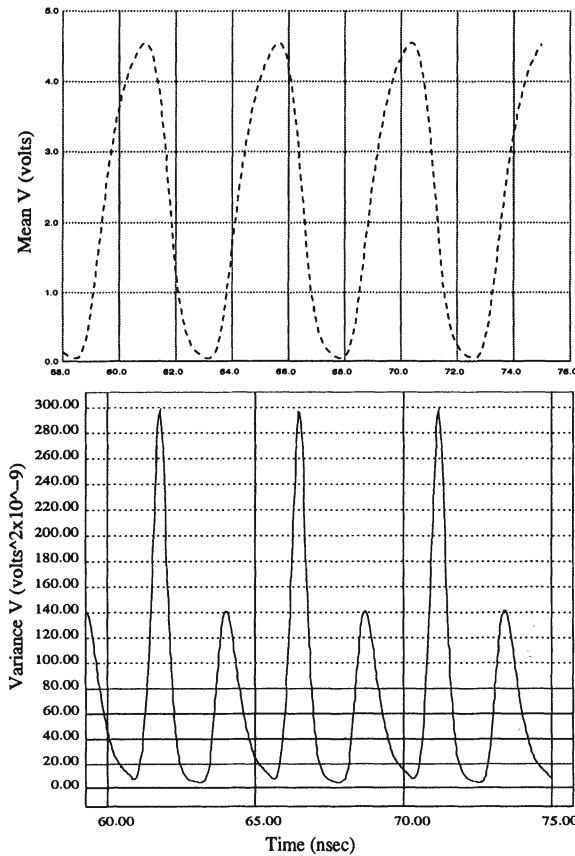


Figure 3. Noise Simulation for the CMOS Ring-oscillator.

stationary, i.e. a constant noise variance as a function of time. The noise performance of a mixer circuit is commonly characterized by its noise figure which can be defined by [1]

$$NF = \frac{\text{total output noise power}}{\text{that part of output noise power due to the source resistance}}. \quad (40)$$

This definition is intended for circuits in small-signal operation. For such circuits, noise figure is a scalar quantity. In our case, the noise at the output of the mixer circuit changes as a function of time over one period. We can generalize the noise figure definition such that noise figure is a quantity that is a function of time. For the mixer circuit we have simulated, the noise figure turns out to be a periodic function of time. To calculate the noise figure as defined, we simulate the mixer circuit again to calculate the noise variance at the output with all the

noise sources turned off except the noise source for the source resistance at the RF port. Then we can calculate the noise figure as below, and the result is shown in Figure 5.

$$NF = 10 \log \left( \frac{\text{Average of Total Noise Variance}}{\text{Average of Noise Variance due to the source resistance only}} \right). \quad (41)$$

As observed in Figure 5, the maximum and minimum value of the noise figure over one period differs by over 4 dB. This BJT mixer circuit has 65 nodes (including the internal nodes for BJTs) which are connected to capacitors. The noise simulation requires the solution of 2145 ( $65 \times 66/2$ ) simultaneous linear equations at every time point, as it was explained in Section 5. The simulation (with 250 time points) took approximately 17 hours on a DECstation 5900/260 with our current implementation (with the direct method linear solver).

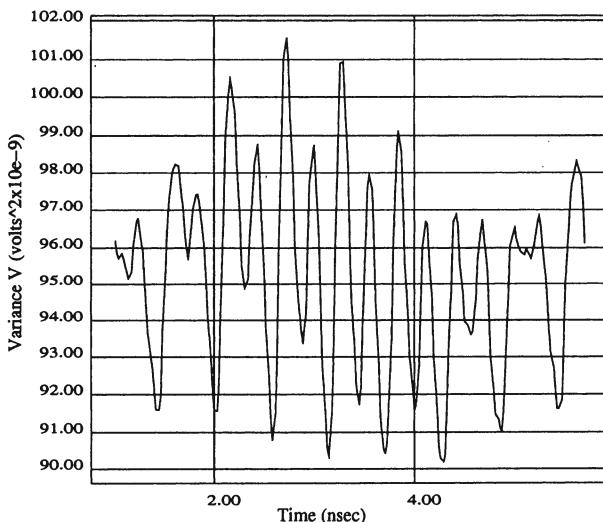


Figure 4. BJT Active Mixer - Noise Variance at the Output.

## 7. Future Work

We plan to compare the results from this noise simulator with noise measurements on actual circuits. The numerical methods used in the noise simulator will be modified to make it more efficient (as explained in Section 5). We will be using our transistor-level noise simulator in the top-down constraint-driven design of a clock generator circuit for a RAMDAC. The noise simulator will be used to extract noise parameters in the behavioral modeling of phase/delay-locked loops [15].

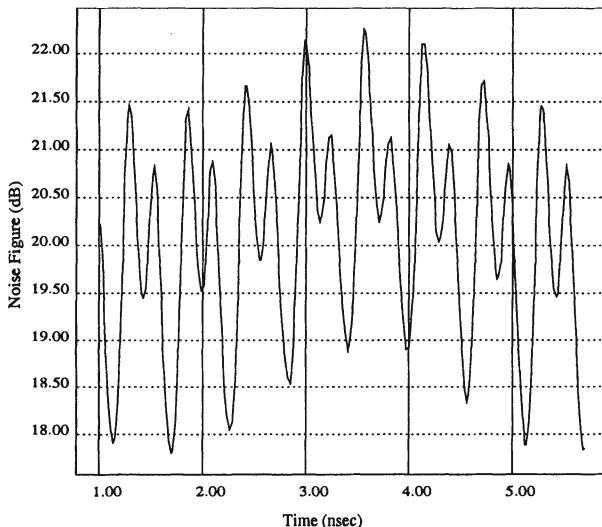


Figure 5. BJT Active Mixer - Noise Figure.

## References

- [1] P.R. Gray and R.G. Meyer, *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, second edition, 1984.
- [2] C.D. Hull, *Analysis and Optimization of Monolithic RF Down Conversion Receivers*. PhD thesis, University of California, Berkeley, 1992.
- [3] C.D. Hull and R.G. Meyer, "A systematic approach to the analysis of noise in mixers," *IEEE Transactions on Circuits and Systems-1: Fundamental Theory and Applications*, vol. 40, no. 12, December 1993.
- [4] R. Rohrer, L. Nagel, R.G. Meyer, and L. Weber, "Computationally efficient electronic-circuit noise calculations," *IEEE Journal of Solid-State Circuits*, vol. SC-6, no. 4, August 1971.
- [5] R.G. Meyer, L. Nagel, and S.K. Lui, "Computer simulation of  $1/f$  noise performance of electronic circuits," *IEEE Journal of Solid-State Circuits*, June 1973.
- [6] M. Okumura, H. Tanimoto, T. Itakura, and T. Sugawara, "Numerical noise analysis for nonlinear circuits with a periodic large signal excitation including cyclostationary noise sources," *IEEE Transactions on Circuits and Systems-1: Fundamental Theory and Applications*, vol. 40, no. 9, September 1993.
- [7] P. Bolcato and R. Poujois, "A new approach for noise simulation in transient analysis," in *Proc. IEEE International Symposium on Circuits and Systems*, May 1992.
- [8] A. Jordan and N. Jordan, "Theory of noise in metal oxide semiconductor devices," *IEEE Transactions on Electron Devices*, March 1965.
- [9] B. Pellegrini, R. Saletti, B. Neri, and P. Terreni, " $1/f$ " noise generators," in *Noise in Physical Systems and  $1/f$  Noise*, 1985.
- [10] A.L. Sangiovanni-Vincentelli, "Circuit simulation," in *Computer Design Aids for VLSI Circuits*. Sijthoff & Noordhoff, The Netherlands, 1980.

- [11] L. Arnold, *Stochastic Differential Equations: Theory and Applications*. John Wiley & Sons, 1974.
- [12] P.E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*. Springer-Verlag, 1992.
- [13] V.S. Pugachev and I.N. Sinitsyn, *Stochastic Differential Systems: Analysis and Filtering*. Wiley, 1987.
- [14] T.L. Quarles, *Analysis of Performance and Convergence Issues for Circuit Simulation*. PhD thesis, University of California, Berkeley, 1989.
- [15] A. Demir, E. Liu, A.L. Sangiovanni-Vincentelli and I. Vassiliou, “Behavioral Simulation Techniques for Phase/Delay-Locked Systems,” *Proc. IEEE Custom Integrated Circuits Conference*, May 1994.
- [16] A. Demir. *Time-Domain non-Monte Carlo Noise Simulation for Nonlinear Dynamic Circuits with Arbitrary Excitations*. Technical Report UCB/ERL M94/39, University of California, Berkeley, May 1994.
- [17] E. Tomacruz, J. Sanghavi, and A. Sangiovanni-Vincentelli, “A parallel iterative linear solver for solving irregular grid semiconductor device matrices,” in *Supercomputing*, 1994.

# PRIMA: PASSIVE REDUCED-ORDER INTERCONNECT MACROMODELING ALGORITHM

Altan Odabasioglu<sup>1</sup>, Mustafa Celik<sup>2</sup> and Lawrence T. Pileggi

*Department of Electrical and Computer Engineering,*

*Carnegie Mellon University,*

*Pittsburgh, PA 15213, USA*

## Abstract

This paper describes PRIMA, an algorithm for generating provably passive reduced order  $N$ -port models for RCL interconnect circuits. It is demonstrated that, in addition to requiring macro-model stability, macromodel passivity is needed to guarantee the overall circuit stability once the active and passive driver/load models are connected. PRIMA extends the block Arnoldi technique to include guaranteed passivity. Moreover, it is empirically observed that the accuracy is superior to existing block Arnoldi methods. While the same passivity extension is not possible for MPVL, we observed comparable accuracy in the frequency domain for all examples considered. Additionally, a path tracing algorithm is used to calculate the reduced order macromodel with the utmost efficiency for generalized RLC interconnects.

## 1. Introduction

As integrated circuits and systems are designed with smaller feature sizes and for faster operation, RLC interconnect effects have a more dominant impact on signal propagation than ever before. In addition, parasitic coupling effects and reduced power supply voltage levels make interconnect modeling increasingly important. Since these interconnect models can contain thousands of tightly coupled R-L-C components, reduced order macromodels are imperative [1] [2] [3] [4]. Ideally, a simulator would isolate the large linear portions of the circuit from the nonlinear elements (e.g., transistor models) and preprocess them into reduced order multiport macromodels.

It is well known that an  $N$ -port can be fully represented by its admittance parameters in the Laplace domain, however, the objective is to apply model order reduction to produce low order rational approximations for each entry in  $\mathbf{Y}(s)$  (see Fig. 1). To find  $\mathbf{Y}(s)$ , voltage sources are connected to the ports and the currents into the ports are measured. The voltage sources are the inputs to the system and the port currents are the outputs. A single-input single-output (SISO)

---

Authors are currently with <sup>1</sup>Synopsys and <sup>2</sup>Magma Design Automation.

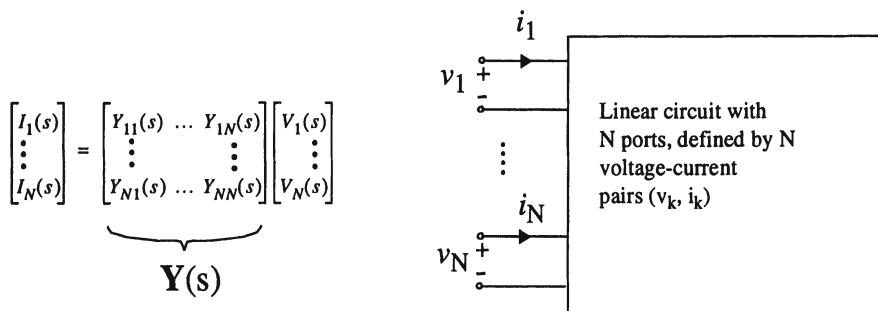


Figure 1. The multiport representation of a linear circuit.

$N$ -port model approach would perform model order reduction on each term  $Y_{ij}$  individually. Both Asymptotic Waveform Evaluation (AWE) [1] and Padé via Lanczos (PVL) [2], which are Padé approximations, can perform SISO reduction by matching  $2q$  moments for a  $q$ th order approximation of each  $Y_{ij}$  term. The Arnoldi Algorithm [4] can also be used to obtain SI $\bar{S}$ O approximations, however it matches only  $q$  moments for a  $q$ th order approximation. MPVL (Matrix Padé via Lanczos) [5] and Block Arnoldi [6] are multi-input multi-output (MIMO) versions of PVL and Arnoldi respectively. In the block techniques, the system Modified Nodal Analysis (MNA) matrices are directly reduced by matrix transformations.

Regardless of the reduction method used in all of the approaches cited above, the reduced order model of an RLC circuit can have unstable poles. It is always possible to obtain an asymptotically stable model by simply discarding the unstable poles, however, passivity is not guaranteed. In addition, discarding unstable poles requires re-adjustment of the residues to improve the quality of the approximation. Passivity uncertainty is problematic since even the test for  $N$ -port passivity can be very costly for a large number of ports [7]. The coordinate transformed Arnoldi Algorithm [8] was introduced as a remedy for the instability problem, but it does not guarantee passivity. The passivity extension of this stable Arnoldi algorithm was recently developed in [9], however its applicability is limited to RC circuits only. The PACT algorithm [3] proposed a new direction for passive reduced-order model for RC circuits based on congruence transformations. The same authors proposed Split Congruence Transformations [10] for passive reductions of RLC circuits, producing equivalent circuit realizations. In [10], however, the extra steps required to split the transformation matrix can result in a decrease in accuracy and efficiency. Moreover, the passivity proof is somewhat controversial, and we will consider a more complete proof in this paper.

A passive system denotes a system that is incapable of generating energy, and hence one that can only absorb energy from the sources used to excite it [11]. As we will show in Section 2.2., passivity is an important property to satisfy because stable, but not passive macromodels can produce unstable systems when connected to other stable, even passive, loads. A property in classical circuit theory states that: interconnections of stable systems may not necessarily be stable; but (strictly) passive circuits are (asymptotically) stable; and arbitrary interconnections of (strictly) passive circuits are (strictly) passive, and, therefore, (asymptotically) stable [12].

In this paper, we propose a Passive Reduced-order Interconnect Macromodeling Algorithm, PRIMA, based on the Block Arnoldi Algorithm but with congruence transformations that produce provably passive reduced order macromodels for arbitrary RLC circuits. PRIMA has accuracy comparable to MPVL and superior to Block Arnoldi. Furthermore, the block Arnoldi vectors are generated with the utmost efficiency following the algorithms in RICE [13] that are used to calculate moments. This includes efficient handling of interconnect trees and meshes, as in RICE, but with renewed focus on efficient handling of large problems with a huge number of mutual inductances.

## 2. Background

To obtain the admittance matrix of a multiport, voltage sources are connected to the ports. The multiport, along with these sources, constitutes the Modified Nodal Analysis (MNA) equations:

$$\begin{aligned} \mathbf{C}\dot{\mathbf{x}}_n &= -\mathbf{G}\mathbf{x}_n + \mathbf{B}\mathbf{u}_p \\ \mathbf{i}_p &= \mathbf{L}^T \mathbf{x}_n \end{aligned} \quad (1)$$

The  $\mathbf{i}_p$  and  $\mathbf{u}_p$  vectors denote the port currents and voltages respectively and

$$\mathbf{G} \equiv \begin{bmatrix} \mathbf{N} & \mathbf{E} \\ -\mathbf{E}^T & \mathbf{0} \end{bmatrix} \quad \mathbf{C} \equiv \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \quad \mathbf{x}_n \equiv \begin{bmatrix} \mathbf{v} \\ \mathbf{i} \end{bmatrix} \quad (2)$$

where  $\mathbf{v}$  and  $\mathbf{i}$  are the MNA variables corresponding to the node voltages and, inductor and voltage source currents, respectively. The  $n \times n$  matrices  $\mathbf{G}$  and  $\mathbf{C}$  represent the conductance and susceptance matrices (except that the rows corresponding to the current variables are negated as in [8]).  $\mathbf{N}$ ,  $\mathbf{Q}$  and  $\mathbf{H}$  are the matrices containing the stamps for resistors, capacitors and inductors respectively.  $\mathbf{E}$  consists of ones, minus ones and zeros, which represent the current variables in KCL equations. Provided that the original  $N$ -port is composed of passive linear elements only,  $\mathbf{N}$ ,  $\mathbf{Q}$ , and  $\mathbf{H}$  are symmetric nonnegative definite matrices. This implies  $\mathbf{C}$  is also symmetric and nonnegative definite. Since this is an  $N$ -port formulation, whereby the only sources are the voltage sources at

the  $N$  port nodes,  $\mathbf{B} = \mathbf{L}$ . But we maintain the separate  $\mathbf{B}$  and  $\mathbf{L}$  notation for the generality of the equations.

Returning to equation (1), following the notation in [2] we define

$$\mathbf{A} = -\mathbf{G}^{-1}\mathbf{C} \text{ and } \mathbf{R} = \mathbf{G}^{-1}\mathbf{B}. \quad (3)$$

With unit voltages at the ports, taking the Laplace transformation of (1) and solving for the port current variables, the y-parameter matrix is given as

$$\mathbf{Y}(s) = \mathbf{L}^T(\mathbf{I}_n - s\mathbf{A})^{-1}\mathbf{R} \quad (4)$$

where  $\mathbf{I}_n$  is the  $n \times n$  identity matrix. It is apparent from (4) that the eigenvalues of  $\mathbf{A}$  represent the inverses of the poles of  $\mathbf{Y}(s)$ .

Using any of the aforementioned model-order reduction techniques, we can find reduced order rational approximations to  $Y_{jk}(s)$  terms, for all  $j, k \leq N$ . The reduced-order  $\mathbf{Y}(s)$  can then be simulated along with other nonlinear and linear portions of the complete circuit using a simulator that employs either recursive convolution [14] or state-space realization [7], both of which have linear complexity. If the reduction is block, the reduced order multi-input multi-output circuit can also be realized using linear circuit elements.

## 2.1 Block Arnoldi Algorithm

The Block Arnoldi algorithm reduces the system matrix  $\mathbf{A}$  in (3) to a small block upper Hessenberg matrix  $\mathbf{H}_q$ . To do so requires an orthonormal basis,  $\mathbf{X}$ , for the corresponding Krylov space which satisfies the following:<sup>1</sup>

$$\begin{aligned} \text{colsp}(\mathbf{X}) &= \text{Kr}(\mathbf{A}, \mathbf{R}, \lfloor \frac{q}{N} \rfloor) \\ \mathbf{X}^T \mathbf{A} \mathbf{X} &= \mathbf{H}_q \\ \mathbf{X}^T \mathbf{X} &= \mathbf{I}_q \end{aligned} \quad (5)$$

where  $N$  is the number of ports and  $\mathbf{I}_q$  is a  $q \times q$  identity matrix. The Krylov space is defined as

$$\text{Kr}(\mathbf{A}, \mathbf{R}, k) = \text{colsp}[\mathbf{R}, \mathbf{A}\mathbf{R}, \mathbf{A}^2\mathbf{R}, \dots, \mathbf{A}^k\mathbf{R}]. \quad (6)$$

Finding the reduced order admittance matrix can be explained by a change of variable,

$$\mathbf{x}_n = \mathbf{X}_{\{n \times q\}} \mathbf{z}_q. \quad (7)$$

where  $\mathbf{z}_q$  is now the reduced order system variable, which reduces the number of unknowns in the system ( $q$  is generally much smaller than  $n$ ). Substituting (7) into (1), then multiplying the first equation by  $\mathbf{X}^T \mathbf{G}^{-1}$  yields

$$\mathbf{H}_q \dot{\mathbf{z}}_q = \mathbf{z}_q - \mathbf{X}^T \mathbf{R} \mathbf{u}_p \mathbf{i}_p = \mathbf{L}^T \mathbf{X} \mathbf{z}_q \quad (8)$$

---

<sup>1</sup>The operator  $\lfloor \cdot \rfloor$  is the truncation to the nearest integer towards zero.

Therefore, in the Laplace domain,

$$\hat{\mathbf{Y}}(s) = \mathbf{L}^T \mathbf{X} (\mathbf{I}_q - s \mathbf{H}_q)^{-1} \mathbf{X}^T \mathbf{R} \quad (9)$$

where  $\mathbf{I}_q$  is the  $q \times q$  identity matrix.

The reduced order system equations and admittance matrix are given by (8) and (9) respectively. The poles of the reduced order system are the reciprocal eigenvalues of  $\mathbf{H}_q$ . A complete pole/residue decomposition can be obtained by eigendecomposing  $\mathbf{H}_q$ . Using the information in [6], it can be shown that the first  $\lfloor \frac{q}{N} \rfloor$  moments of  $\hat{\mathbf{Y}}(s)$  in (9) match those of  $\mathbf{Y}(s)$  in (4).

## 2.2 Importance of Passivity

It is always possible to come up with stable reduced order macromodels by utilizing a number of heuristics, however, none of these tricks can be used to obtain provably passive approximations. Moreover, in [7] it was shown that the test for passivity of an AWE-reduced  $N$ -port macromodel is prohibitive in terms of CPU run time cost. Fig. 2 is a numerical example generated in [7] that demonstrates the passivity problem.  $Y_1(s)$  in this figure represents a reduced order transfer function which has all poles and zeros in the left half plane.  $Y_{dr}(s)$  represents a capacitor and resistor in parallel. If we drive this circuit with a current source, it will oscillate at  $2.5/\pi$  Hz. To show that it is also a practical problem, we took a simple interconnect and connected the load and the nonlinear driver as shown in the examples section in Fig. 5. The interconnect is represented by a fifth order approximation obtained by PVL [2]. The figure clearly shows the growing oscillations at the output (instability) although all of the poles obtained from PVL were stable. A Thevenin equivalent linear driver with a resistance of 2 ohms generates a similar instability for this 2-port example.

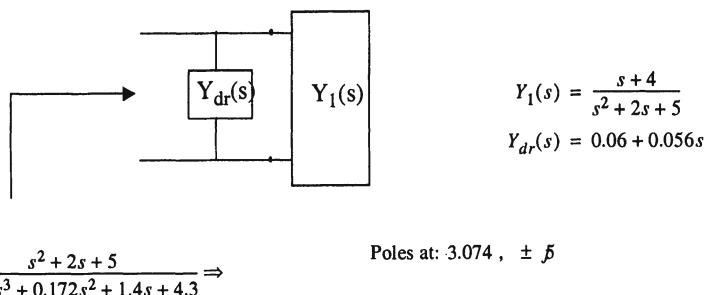


Figure 2. A non-passive system example demonstrating potential instability.

◇ Connect voltage sources to the multiport and obtain the MNA matrices as in (2)

◇ Set  $[\mathbf{b}_1|\mathbf{b}_2|...|\mathbf{b}_p] = \mathbf{B}$  and  $[\mathbf{l}_1|\mathbf{l}_2|...|\mathbf{l}_p] = \mathbf{L}$

◇ Solve  $\mathbf{GR} = \mathbf{B}$  for  $\mathbf{R}$

◇  $(\mathbf{X}_0, \mathbf{K}) = qr(\mathbf{R})$ ; qr factorization of  $\mathbf{R}$

◇ Set  $n = \text{int}(q/N) + 1$

◇ For  $k = 1, 2, \dots, n$

Set  $\mathbf{V} = \mathbf{CX}_{k-1}$

Solve  $\mathbf{GX}_k^{(0)} = \mathbf{V}$  for  $\mathbf{X}_k^{(0)}$

For  $j = 1, \dots, k$

$$\mathbf{H} = \mathbf{X}_{k-j}^T \mathbf{X}_k^{(j-1)}$$

$$\mathbf{X}_k^{(j)} = \mathbf{X}_k^{(j-1)} - \mathbf{X}_{k-j} \mathbf{H}$$

$(\mathbf{X}_k, \mathbf{K}) = qr(\mathbf{X}_k^{(k)})$ ; qr factorization of  $\mathbf{X}_k^{(k)}$

◇ Set  $\mathbf{X} = [\mathbf{X}_0|\mathbf{X}_1|...|\mathbf{X}_{k-1}]$  and truncate  $\mathbf{X}$  so that it has  $q$  columns only.

◇ Compute  $\tilde{\mathbf{C}} = \mathbf{X}^T \mathbf{CX}$ ,  $\tilde{\mathbf{G}} = \mathbf{X}^T \mathbf{GX}$

◇ Find eigendecomposition of  $\tilde{\mathbf{G}}^{-1} \tilde{\mathbf{C}}$ :  $\tilde{\mathbf{G}}^{-1} \tilde{\mathbf{C}} = \mathbf{S} \lambda \mathbf{S}^{-1}$  [inversion of  $\tilde{\mathbf{G}}$  can be avoided]

$$\lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_q)$$

◇ To find poles and residues for  $\hat{Y}_{i,j}(s)$ :

Solve  $\tilde{\mathbf{G}}\mathbf{w} = \mathbf{X}^T \mathbf{b}_j$  for  $\mathbf{w}$

Set  $\mu = \mathbf{S}^T \mathbf{X}^T \mathbf{l}_i$  and  $\mathbf{v} = \mathbf{S}^{-1} \mathbf{w}$

$$\hat{Y}_{i,j}(s) = \sum_{n=1}^q \frac{\mu_n v_n}{1+s\lambda_n}$$

◇ Set  $\hat{\mathbf{Y}}(s) = \begin{bmatrix} \hat{Y}_{1,1} & \dots & \hat{Y}_{1,p} \\ \dots & \dots & \dots \\ \hat{Y}_{p,1} & \dots & \hat{Y}_{p,p} \end{bmatrix}$

Figure 3. The passive reduction algorithm.

### 3. PRIMA: Passive Reduced-order Interconnect Macromodeling Algorithm

The Block Arnoldi Algorithm is employed in PRIMA to generate the orthonormal basis for a congruence transformation matrix. After  $\lfloor \frac{q}{N} \rfloor + 1$  (the extra step is not necessary when  $\frac{q}{N}$  is an integer) iterations of PRIMA, the  $n \times q$  matrix  $\mathbf{X}$  is found such that:

$$\begin{aligned} \text{colsp}(\mathbf{X}) &= Kr(\mathbf{A}, \mathbf{R}, \lfloor \frac{q}{N} \rfloor) \\ \mathbf{X}^T \mathbf{X} &= \mathbf{I}_q \end{aligned} \tag{10}$$

In the classical Arnoldi approach [4], the reduced order  $\mathbf{Y}(s)$  is calculated using the  $q \times q$  upper Hessenberg matrix in (5) as shown in (9):

$$\hat{\mathbf{Y}}(s) = \mathbf{L}^T \mathbf{X} (\mathbf{I}_q - s\mathbf{H})^{-1} \mathbf{X}^T \mathbf{R} \quad (11)$$

In our variation, the conductance and susceptance matrices are directly reduced so that passivity is preserved during reduction. Applying the change of variable  $\mathbf{x}_n = \mathbf{X}_{\{n \times q\}} \tilde{\mathbf{x}}_q$  in (1), and multiplying the first row by  $\mathbf{X}^T$  from (10) yields

$$\begin{aligned} (\mathbf{X}^T \mathbf{C} \mathbf{X}) \dot{\tilde{\mathbf{x}}}_q &= -(\mathbf{X}^T \mathbf{G} \mathbf{X}) \tilde{\mathbf{x}}_q + \mathbf{X}^T \mathbf{B} \mathbf{u}_p \\ \mathbf{i}_p &= \mathbf{L}^T \mathbf{X} \tilde{\mathbf{x}}_q \end{aligned} \quad (12)$$

The reduced order MNA matrices, are, therefore,

$$\begin{aligned} \tilde{\mathbf{C}} &= \mathbf{X}^T \mathbf{C} \mathbf{X} & \tilde{\mathbf{G}} &= \mathbf{X}^T \mathbf{G} \mathbf{X} \\ \tilde{\mathbf{B}} &= \mathbf{X}^T \mathbf{B} & \tilde{\mathbf{L}} &= \mathbf{X}^T \mathbf{L} \end{aligned} \quad (13)$$

These types of transformations are known as congruence transformations. Congruence transformations were first introduced by [3] for order reduction of circuits. From (12) and (13), the reduced  $\mathbf{Y}(s)$ , namely  $\hat{\mathbf{Y}}(s)$ , is now

$$\hat{\mathbf{Y}}(s) = \tilde{\mathbf{L}}^T (\tilde{\mathbf{G}} + s\tilde{\mathbf{C}})^{-1} \tilde{\mathbf{B}} \quad (14)$$

Since the sizes of  $\tilde{\mathbf{G}}$  and  $\tilde{\mathbf{C}}$  are typically very small, it is easy to find the poles and zeros of  $\hat{\mathbf{Y}}(s)$  by eigendecomposition. The complete algorithm is given in Fig. 3. It employs the Block Arnoldi Algorithm using modified Gram-Schmidt orthogonalization [6], which is mathematically equivalent to ordinary Gram-Schmidt process, but behaves better numerically [15]. In addition, it is possible to avoid the inversion of  $\tilde{\mathbf{G}}$  to find the poles and residues by using a generalized eigendecomposition. In this case, the computation of  $\tilde{\mathbf{G}}^{-1} \tilde{\mathbf{B}}$  can be avoided by using (31) and replacing it by  $\mathbf{X}^T \mathbf{R}$ . It is observed that this scheme is numerically much better.

The complexity of the algorithm to produce  $q$  poles for an  $N$ -port is slightly less than AWE, PVL and MPVL. It requires 1 LU factorization (or path tracing equivalent as explained in Section 4.) of the  $\mathbf{G}$  (MNA conductance) matrix, which dominates all the other computational costs and is common in all reduction techniques. However, to find  $q$  poles, only  $q$  backward-forward substitutions are needed, whereas in MPVL, PVL and AWE, twice as many are required. As in MPVL, there will be only one eigendecomposition to find the poles and residues, whereas PVL requires  $N^2$  eigendecompositions, since for each  $Y_{ij}(s)$ , there will be a different  $T_q$ . AWE will solve the  $N^2$  different Hankel matrices to get to the poles.

### 3.1 Preservation of Passivity

If the system described by (1) and (2) is reduced by the transformations in (13), it can be shown that the reduced system is always passive. In [16], neces-

sary and sufficient conditions for the system admittance matrix  $\hat{\mathbf{Y}}(s)$  (eqn. (14)) to be passive are:

1.  $\hat{\mathbf{Y}}(s^*) = \hat{\mathbf{Y}}^*(s)$  for all complex  $s$ , where  $*$  is the complex conjugate operator.

2.  $\hat{\mathbf{Y}}(s)$  is a positive matrix, that is,  $\mathbf{z}^{*T}(\hat{\mathbf{Y}}(s) + \hat{\mathbf{Y}}^{*T}(s))\mathbf{z} \geq 0$  for all complex values of  $s$  satisfying  $\Re(s) > 0$  and for any complex vector  $\mathbf{z}$ .

The second condition also implies the analyticity of  $\hat{\mathbf{Y}}(s)$  for  $\Re(s) > 0$ , since  $\hat{\mathbf{Y}}(s)$  is a rational function of  $s$  (details in [16]). Therefore, the test of analyticity is unnecessary.

Due to the fact that the reduced matrices  $\tilde{\mathbf{G}}$ ,  $\tilde{\mathbf{C}}$ ,  $\tilde{\mathbf{B}}$ , and  $\tilde{\mathbf{L}}$  are all real since the transformation matrix,  $\mathbf{X}$ , is real, condition 1 is automatically satisfied. To show that condition 2 is satisfied, we first set  $\mathbf{Y}_h(s) = \hat{\mathbf{Y}}(s) + \hat{\mathbf{Y}}^{*T}(s^*)$  and use the property  $\tilde{\mathbf{B}} = \tilde{\mathbf{L}}$  (since  $\mathbf{B} = \mathbf{L}$  in our formulation when there are no sources inside the  $N$ -port,  $\mathbf{X}^T \mathbf{B} = \mathbf{X}^T \mathbf{L}$ ) and some algebra to obtain,

$$\begin{aligned}\mathbf{z}^{*T} \mathbf{Y}_h(s) \mathbf{z} &= \mathbf{z}^{*T} (\tilde{\mathbf{B}}^T (\tilde{\mathbf{G}} + s\tilde{\mathbf{C}})^{-1} \tilde{\mathbf{B}} + \tilde{\mathbf{B}}^T (\tilde{\mathbf{G}} + s^* \tilde{\mathbf{C}})^{-T} \tilde{\mathbf{B}}) \mathbf{z} \\ &= \mathbf{z}^{*T} \tilde{\mathbf{B}}^T (\tilde{\mathbf{G}} + s\tilde{\mathbf{C}})^{-1} ((\tilde{\mathbf{G}} + s\tilde{\mathbf{C}}) + (\tilde{\mathbf{G}} + s^* \tilde{\mathbf{C}})^T) (\tilde{\mathbf{G}} + s^* \tilde{\mathbf{C}})^{-T} \tilde{\mathbf{B}} \mathbf{z}\end{aligned}\quad (15)$$

Setting  $\mathbf{w} = (\tilde{\mathbf{G}} + s^* \tilde{\mathbf{C}})^{-T} \tilde{\mathbf{B}} \mathbf{z}$  and  $s = j\omega + \sigma$  yields,

$$\begin{aligned}\mathbf{z}^{*T} \mathbf{Y}_h(s) \mathbf{z} &= \mathbf{w}^{*T} [(\tilde{\mathbf{G}} + (j\omega + \sigma) \tilde{\mathbf{C}}) + (\tilde{\mathbf{G}} + (j\omega - \sigma) \tilde{\mathbf{C}})^T] \mathbf{w} \\ &= \mathbf{w}^{*T} [\tilde{\mathbf{G}} + \tilde{\mathbf{G}}^T + \sigma(\tilde{\mathbf{C}} + \tilde{\mathbf{C}}^T)] \mathbf{w} \\ &= \mathbf{w}^{*T} \mathbf{X}^T [\mathbf{G} + \mathbf{G}^T + \sigma(\mathbf{C} + \mathbf{C}^T)] \mathbf{X} \mathbf{w}\end{aligned}\quad (16)$$

Similarly, let  $\mathbf{y} = \mathbf{X} \mathbf{w}$  to get

$$\mathbf{z}^{*T} \mathbf{Y}_h(s) \mathbf{z} = \mathbf{y}^{*T} [\mathbf{G} + \mathbf{G}^T + \sigma(\mathbf{C} + \mathbf{C}^T)] \mathbf{y} \quad (17)$$

Since  $\mathbf{C}$  is symmetric,  $\mathbf{C}^T + \mathbf{C} = 2\mathbf{C}$ .  $\mathbf{C}$  is known to be nonnegative definite (since we negate the rows corresponding to current variables as in (2)), so

$$\mathbf{y}^{*T} \sigma(\mathbf{C} + \mathbf{C}^T) \mathbf{y} = 2\sigma \mathbf{y}^{*T} \mathbf{C} \mathbf{y} \geq 0 \quad (18)$$

for any complex vector  $\mathbf{y}$  and  $\sigma = \Re(s) > 0$ .  $\mathbf{N}$  (the resistor stamps) is a symmetric nonnegative definite matrix, therefore

$$\begin{aligned}\mathbf{y}^{*T} (\mathbf{G} + \mathbf{G}^T) \mathbf{y} &= \mathbf{y}^{*T} \left( \left[ \begin{array}{cc} \mathbf{N} & \mathbf{E} \\ -\mathbf{E}^T & \mathbf{0} \end{array} \right]^T + \left[ \begin{array}{cc} \mathbf{N} & \mathbf{E} \\ -\mathbf{E}^T & \mathbf{0} \end{array} \right] \right) \mathbf{y} \\ &= \mathbf{y}^{*T} \left[ \begin{array}{cc} 2\mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{array} \right] \mathbf{y}\end{aligned}\quad (19)$$

is also nonnegative definite for any complex vector  $\mathbf{y}$ . From (17), (18), and (19), it follows that the second passivity condition is satisfied.

### 3.2 Preservation of Moments

The transformation in (13) preserves  $\lfloor \frac{q}{N} \rfloor$  moments of the original system, which is the same as the classical Block Arnoldi reduction and half of that in MPVL. The proof is as follows. The exact (block) moments,  $\mathbf{M}_i$ , of the circuit are given as:

$$\mathbf{M}_i = \mathbf{L}^T \mathbf{A}^i \mathbf{R} \quad (20)$$

where  $\mathbf{A} \equiv -\mathbf{G}^{-1}\mathbf{C}$ ,  $\mathbf{R} \equiv \mathbf{G}^{-1}\mathbf{B}$ , and  $\mathbf{G}$ ,  $\mathbf{C}$ ,  $\mathbf{B}$ ,  $\mathbf{L}$  are the system matrices as defined in (1).

Likewise, the moments of the PRIMA reduced order system are given by

$$\hat{\mathbf{M}}_i = \tilde{\mathbf{L}}^T \tilde{\mathbf{A}}^i \tilde{\mathbf{R}} \quad (21)$$

where  $\tilde{\mathbf{A}} \equiv \tilde{\mathbf{G}}^{-1}\tilde{\mathbf{C}}$ ,  $\tilde{\mathbf{R}} \equiv \tilde{\mathbf{G}}^{-1}\tilde{\mathbf{B}}$  and  $\tilde{\mathbf{G}}$ ,  $\tilde{\mathbf{C}}$ ,  $\tilde{\mathbf{B}}$ ,  $\tilde{\mathbf{L}}$  are as defined in (13). Substitution of (13) in (21) yields:

$$\tilde{\mathbf{M}}_i = \mathbf{L}^T \mathbf{X} \left[ -(\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{C} \mathbf{X}) \right]^i (\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{B} \quad (22)$$

It is shown in [6] that the Arnoldi algorithm yields

$$\mathbf{A}^i \mathbf{R} = \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R}, \quad 0 \leq i < \lfloor \frac{q}{N} \rfloor \quad (23)$$

Rearranging the terms and using the definitions from (13):

$$\begin{aligned} \mathbf{A} \mathbf{A}^{i-1} \mathbf{R} &= \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R} \\ -\mathbf{G}^{-1} \mathbf{C} \mathbf{A}^{i-1} \mathbf{R} &= \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R} \\ -\mathbf{C} \mathbf{A}^{i-1} \mathbf{R} &= \mathbf{G} \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R} \end{aligned} \quad (24)$$

$$-\mathbf{X}^T \mathbf{C} \mathbf{A}^{i-1} \mathbf{R} = \mathbf{X}^T \mathbf{G} \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R} \quad (25)$$

$$-\mathbf{X} (\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{C} \mathbf{A}^{i-1} \mathbf{R} = \mathbf{X} \mathbf{H}_q^i \mathbf{X}^T \mathbf{R} \quad (25)$$

Inserting (23) in (25) results in:

$$\mathbf{K} \mathbf{A}^{i-1} \mathbf{R} = \mathbf{A}^i \mathbf{R}, \quad 0 \leq i < \lfloor \frac{q}{N} \rfloor \quad (26)$$

where

$$\mathbf{K} = -\mathbf{X} (\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{C}. \quad (27)$$

From (26), it can be shown by recursion that

$$\mathbf{K}^i \mathbf{R} = \mathbf{A}^i \mathbf{R}, \quad 0 \leq i < \lfloor \frac{q}{N} \rfloor \quad (28)$$

Therefore, using (27) it follows that

$$\mathbf{X} \left[ -(\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{C} \mathbf{X}) \right]^i = \mathbf{K}^i \mathbf{X} \quad (29)$$

Replacing  $\mathbf{X} \left[ -(\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{C} \mathbf{X}) \right]^i$  in (22) with  $\mathbf{K}^i \mathbf{X}$  yields

$$\hat{\mathbf{M}}_i = \mathbf{L}^T \mathbf{K}^i \mathbf{X} (\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{B} \quad (30)$$

Evaluating (26) when  $i = 0$  gives

$$\mathbf{X} (\mathbf{X}^T \mathbf{G} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{B} = \mathbf{R} \quad (31)$$

Then from (30) and (31),

$$\hat{\mathbf{M}}_i = \mathbf{L}^T \mathbf{K}^i \mathbf{R}, \quad 0 \leq i < \lfloor \frac{q}{N} \rfloor \quad (32)$$

Finally, combining (32) and (28) with (20), it follows that

$$\hat{\mathbf{M}}_i = \mathbf{M}_i, \quad 0 \leq i < \lfloor \frac{q}{N} \rfloor \quad (33)$$

Note that the number of poles in each entry of  $\mathbf{Y}_S$  is  $q$ , and we have matched the first  $\lfloor \frac{q}{N} \rfloor$  moments at all  $N$  ports, yielding a total of  $q$  moments. The number of moments matched in PRIMA is, therefore, the same as that for the Block Arnoldi algorithm and half as many as matched by MPVL.

#### 4. Integration of PRIMA within RICE

For all of the model order reduction schemes, the LU decomposition of the MNA conductance matrix ( $\mathbf{G}$  in (2)) dominates the run time. In [13], RICE (Rapid Interconnect Circuit Evaluation) was described as a general path tracing algorithm to obtain moments with optimal efficiency for interconnect trees and mesh structures. Using RICE to calculate moments, the explicit construction and inversion of  $\mathbf{G}$  is avoided, and the moments are more accurate than those obtained via matrix factorization.

The moments of the circuit can be obtained recursively from:

$$\begin{aligned} \mathbf{M}_0 &= \mathbf{G}^{-1} \mathbf{B} \\ \mathbf{M}_k &= \mathbf{G}^{-1} \mathbf{C} \mathbf{M}_{k-1}, \quad k > 0 \end{aligned} \quad (34)$$

where the matrices  $\mathbf{G}$ ,  $\mathbf{C}$ , and  $\mathbf{B}$  are as defined in (2). As shown in [1], this can be viewed as recursive dc circuit solutions, when capacitors and inductors are replaced by current and voltage sources respectively, with the values derived from the columns of  $\mathbf{C} \mathbf{M}_{k-1}$ . The Krylov vectors, which can be viewed as well conditioned moments, can be obtained from a very similar recursive scheme:

1. Obtain zeroth moment and orthonormalize it:

$$\text{Solve } \mathbf{M}_0 \text{ from } \mathbf{GM}_0 = \mathbf{B} \quad (35)$$

$$\mathbf{X}_0 = \text{orth}(\mathbf{M}_0) \quad (36)$$

2. Recursively obtain higher order Krylov vectors:

$$\text{Solve } \mathbf{M}_k \text{ from } \mathbf{GM}_k = \mathbf{CX}_{k-1} \quad (37)$$

$$\mathbf{X}_k^\zeta = \mathbf{M}_k - \mathbf{X}_{k-1} (\mathbf{X}_{k-1}^T \mathbf{M}_k) - \dots - \mathbf{X}_0 (\mathbf{X}_0^T \mathbf{M}_k) \quad (38)$$

$$\mathbf{X}_k = \text{orth}(\mathbf{X}_k^\zeta) \quad (39)$$

The “orth” operator can be implemented as a simple Gram-Schmidt orthonormalization procedure. The space spanned by the block Krylov terms  $(\mathbf{X}_k, \mathbf{X}_{k-1}, \dots, \mathbf{X}_0)$  is called the Krylov space. Therefore, the Krylov vectors can be obtained via a path tracing procedure using RICE-like routines to solve for equations (35) and (37).

The Krylov space constitutes the congruence transformation matrix,  $\mathbf{X}$  in PRIMA. The reduced MNA matrices  $\tilde{\mathbf{G}}$  and  $\tilde{\mathbf{C}}$  are

$$\tilde{\mathbf{C}} = \mathbf{X}^T \mathbf{C} \mathbf{X} \quad \tilde{\mathbf{G}} = \mathbf{X}^T \mathbf{G} \mathbf{X} \quad (40)$$

Note, however, that the matrices  $\mathbf{CX}$  and  $\mathbf{GX}$  are obtained using RICE without explicitly constructing  $\mathbf{C}$  and  $\mathbf{G}$ . The columns of  $\mathbf{CX}$  are the values of current and voltage sources that are used to replace capacitors and inductors at each moment computation stage. This information is easily obtained during a path trace [13]. The  $k^{th}$  block of  $\mathbf{GX}$  (i.e.  $\mathbf{GX}_k$ ) is a function of previous blocks of  $\mathbf{GX}$  and  $\mathbf{CX}_{k-1}$  since from (38),

$$\mathbf{GX}_k^\zeta = \mathbf{GM}_k - \mathbf{GX}_{k-1} (\mathbf{X}_{k-1}^T \mathbf{M}_k) - \dots - \mathbf{GX}_0 (\mathbf{X}_0^T \mathbf{M}_k) \quad (41)$$

and using  $\mathbf{GM}_k = \mathbf{CX}_{k-1}$ ,

$$\mathbf{GX}_k^\zeta = \mathbf{CX}_{k-1} - \mathbf{GX}_{k-1} (\mathbf{X}_{k-1}^T \mathbf{M}_k) - \dots - \mathbf{GX}_0 (\mathbf{X}_0^T \mathbf{M}_k) \quad (42)$$

## 5. Time-domain Simulation of Macromodels

For a complete circuit simulation, the nonlinear elements should be simulated along with the reduced order macromodels. There are two ways to include the PRIMA macromodels into circuit simulators such as SPICE [17]. One approach is in terms of the frequency domain y-parameters. Combining the nonlinear time domain analysis in SPICE with the frequency dependent y-parameters requires convolution of O(T<sup>2</sup>) complexity, where T is the number of simulation time-points. For this reason, recursive convolution [14] and time-domain y-parameter

macromodels [7] were developed, where the complexity is linear with the number of time-points. The second method is the direct stamping (i.e. circuit model realization). Since the reduction method we use is block, the reduced matrices can be directly stamped into the SPICE MNA matrices. Noticing that the reduced order q-variable system has the equation shown in (12) and (13), and recognizing that it is possible to introduce as a circuit variable into the MNA matrix, the direct stamps for the macromodel can be generated as below:

$$\begin{bmatrix} \text{Stamps for } f(x_{NL}, u_p) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & -\tilde{\mathbf{B}} & (\tilde{\mathbf{G}} + \tilde{\mathbf{C}} \frac{d}{dt}) \end{bmatrix} \begin{bmatrix} \mathbf{x}_{NL} \\ \mathbf{u}_p \\ \mathbf{i}_p \\ \tilde{\mathbf{x}}_q \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{NL} \\ \mathbf{v}_p \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (43)$$

In (43),  $\mathbf{x}_{NL}$  denotes the other variables of the circuit (other node voltages and currents)  $\mathbf{u}_p$  and  $\mathbf{i}_p$  are port voltages and currents respectively, and  $\tilde{\mathbf{x}}_q$  denotes the extra variables that are introduced from the inclusion of realized macromodel into the circuit. Since  $\tilde{\mathbf{C}}$  is a symmetric and real matrix, it can be diagonalized using singular value decomposition. In this case, all the capacitance values will be real and positive, since they will be the singular values of  $\tilde{\mathbf{C}}$ . Note that it is also possible to come up with a realization scheme similar to (43) starting from  $\hat{\mathbf{Y}}(s)$ .

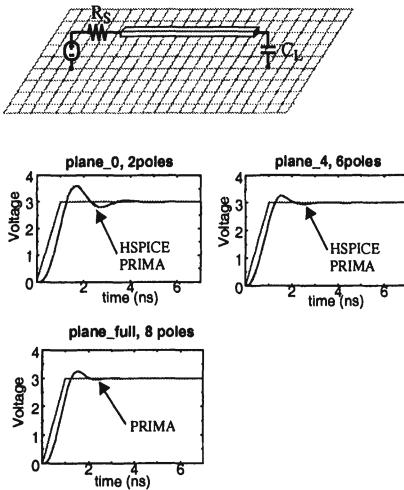
## 6. Results

In this section, PRIMA is demonstrated and compared with other approaches. All reductions are done using RICE v5.0, a program which integrates the PRIMA algorithm with the RICE moment calculation routines. For the frequency domain examples, the y-parameters are compared with the reduced order models from different reduction methods. Time domain results via recursive convolution are obtained using a modified version of SPICE3f4 [18]. For all the examples, the poles obtained via PRIMA were observed to be stable.

### 6.1 Mesh Ground Plane

With the ability to calculate a large number of poles accurately, PRIMA can be applied to analysis problems which include complex, high frequency responses. One such application is the R-L mesh plane encountered in MCM and packaging problems. Since such a problem is strongly coupled, the L-matrix is dense and thereby destroys the matrix sparsity in a classical SPICE simulation. In this example, the ground plane is modeled by a 20x20 mesh, and each square is modeled as a resistor and an inductor. The coupling can be adjusted to make the inductance matrix sparse as described in [19].

There is an RL line over the ground plane that is terminated with a capacitor load, as shown in Fig. 4. Also shown in the figure are time domain results



Circuit complexity

Circuit name	# of R	# of L	# of K
plane_0	859	858	0
plane_4	859	858	17,892
plane_full	859	858	183,693

Run time comparisons

Circuit name	HSPICE	PRIMA	RecConv
plane_0	39.97 secs	0.25 secs	0.03 secs
plane_4	17,343 secs	1.23 secs	0.05 secs
plane_full	can not run	11.73 secs	0.05 secs

Figure 4. Mesh ground plane example.

from PRIMA and HSPICE for various levels of L-matrix sparsity. The full matrix response is also shown for PRIMA, but the HSPICE simulation would not complete its run due to memory and run-time limits.

This circuit is a worst-case interconnect topology for a path tracing algorithm [13] (all loops), however, RICE v5, our path tracing implementation of PRIMA, showed excellent speed-up over HSPICE, a commercial circuit simulation tool. The table in this figure also includes the time required for recursive convolution of the reduced-order model in SPICE3f, denoted by RecConv.

## 6.2 Nonlinear Driver Driving a Transmission Line

Fig. 5 shows a lossy transmission line represented by 40 lumped RLC sections and reduced to five poles using both PVL and PRIMA. Although all of the poles from PVL were stable (i.e. negative real parts), the overall PVL response was clearly unstable as shown in Fig. 5. The fifth order approximation from PRIMA is indistinguishable from the exact response, which was obtained by an HSPICE simulation for this example.

## 6.3 Coupled Noise for a Two-bit Bus

Next consider the two-bit bus driven by CMOS inverters in Fig. 6. One of the drivers is switching while the other is quiet. The interconnect, consisting of 40 coupled RLC sections, is modeled as a 4-port and reduced by PRIMA. Transient analysis is done using recursive convolution. The time domain waveforms at the load end are compared for various order of approximations. Since this is

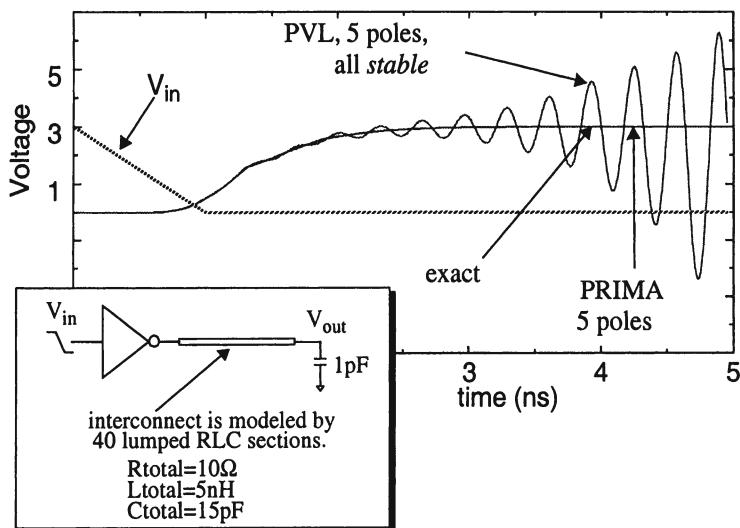


Figure 5. Instability in the time domain for non-passive model.

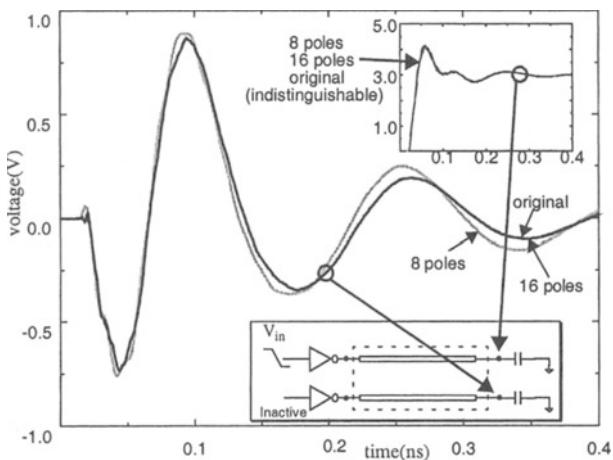


Figure 6. Waveform comparisons for a four port.

a 4-port, an 8 pole approximation corresponds to matching only  $m_0$  and  $m_1$  generated by four different sources. The plot shows that in the time domain, even the coupled noise can be accurately simulated using the 8 poles from PRIMA. Although the interconnect inductance was exaggerated in this example to make the approximation more difficult, it is observed that an 8th order approximation

Exact	Reduced	
Full Simulation	Simulated After Direct Realization	Simulated by Y-parameter based
17.78 sec	0.6 s. with 8 poles 3.98 s. with 16 poles 10.29 s. with 24 poles	0.18 s. with 8 poles 0.28 s. with 16 poles 0.32 s. with 24 poles

Table 1. Run time comparisons.

is sufficient to capture the coupled noise from the active driver to the quiet load end.

To compare the difference between direct realization and y-parameter based simulation (i.e. recursive convolution here), the reduced order circuit (via PRIMA) is simulated using both techniques. In the direct realization, is diagonalized to increase the speed. The run times are given in Table 1. Although the circuit is relatively small (i.e.  $G$  is only  $300 \times 300$ ), the gain in using a PRIMA reduced macromodel and y-parameter based simulation is about 50x over direct realization. For larger circuits such as the mesh plane example, this gain is expected to be much larger. Direct realization is inferior when the order of approximation gets bigger, mainly because the dense matrix gets larger.

## 6.4 Six Coupled Transmission Lines

The second example is a 12-port containing six coupled transmission lines modeled by 40 coupled RLC sections. The input admittance ( $Y_{11}(s)$ ), reduced by Block Arnoldi, MPVL and PRIMA are compared with the exact input admittance in Fig. 7 using 48th order approximations in all cases. Block Arnoldi captures the exact response up to 16 GHz, while MPVL and PRIMA match up to 28 GHz. When the order of approximation is increased to 72 poles, it is observed that the frequency spectrum is captured up to 60 GHz by MPVL and PRIMA.

## 6.5 Large Coupled RLC Circuit

The third example in Fig. 8 displays the responses for a 3-port composed of densely coupled RLC circuits. Approximations are done using 25 poles for the three methods. As can be observed from the figure, both PRIMA and MPVL capture the entire frequency spectrum.

## 7. Conclusions

This paper presented a novel algorithm for producing provably passive macromodels for arbitrary RLC circuits. The method uses a Block Arnoldi algorithm to generate the vectors needed for applying a transformation to the macromodel

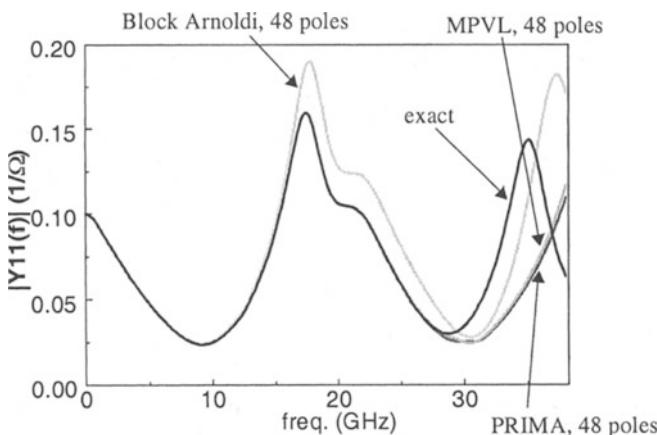


Figure 7.  $Y_{11}(s)$  in the frequency domain for six coupled transmission lines.

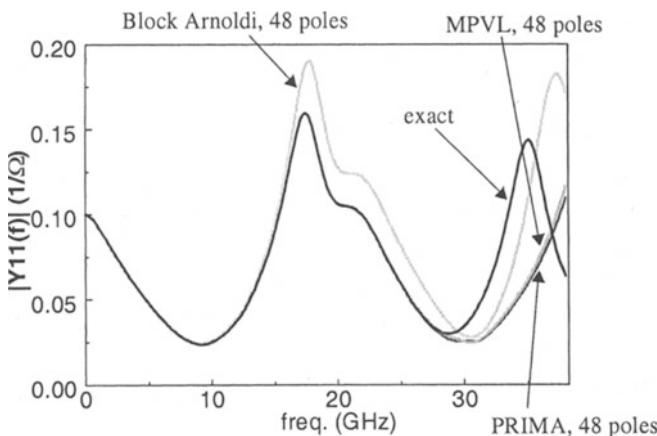


Figure 8. 3-Port consisting of a large lumped RLC circuit.

MNA matrices. Empirical results show that PRIMA produces comparable or superior results in terms of accuracy with respect to all other known reduction techniques, but superior in that it guarantees the passivity that is critical for time domain analyses. The implementation of PRIMA with path tracing algorithms from RICE enables extremely accurate high frequency response approximations of enormous, complex, RLC circuits with excellent efficiency.

The PRIMA algorithm presented in this paper can be easily extended to implement a number of heuristics such as moment shifting [13] and frequency shifting [20]. However, these heuristics are unnecessary and merely increase the complexity.

## References

- [1] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 4, pp. 352-366, Apr. 1990.
- [2] P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Padé approximation via the Lanczos process," *IEEE Trans. on CAD*, vol. CAD-14, pp. 639-649, May 1995.
- [3] K. J. Kerns, I. L. Wemple, and A. T. Yang, "Stable and efficient reduction of substrate model networks using congruence transforms," *IEEE/ACM Proc. ICCAD*, pp. 207-214, Nov. 1995.
- [4] L. M. Silveira, M. Kamon and J. White, "Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-D interconnect structures," *IEEE/ACM Proc. DAC*, pp. 376-380, Jun. 1995.
- [5] P. Feldmann and R. W. Freund, "Reduced-order modeling of large linear subcircuits via a block Lanczos algorithm", *IEEE/ACM Proc. DAC*, pp. 474-479, Jun. 1995.
- [6] D. L. Boley, "Krylov space methods on state-space control models," *Circuits Syst. Signal Process*, vol.13, no.6, pp. 733-758, 1994.
- [7] S. Y. Kim, N. Gopal and L. T. Pillage, "Time-Domain Macromodels for VLSI Interconnect Analysis," *IEEE Trans. on CAD*, vol 13, No. 10, pp. 1257-1270, Oct. 1994.
- [8] L. M. Silveira, M. Kamon, I. Elfadel and J. White, "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of arbitrary RLC circuits," *IEEE/ACM Proc. ICCAD*, pp. 288-294, Nov. 1996.
- [9] I. M. Elfadel and D. Ling, "Zeros and passivity of Arnoldi-reduced-order models for interconnect networks," *IEEE/ACM Proc. DAC*, pp. 28-33, Jun. 1997.
- [10] K. J. Kerns and A. T. Yang, "Preservation of Passivity During RLC Network Reduction via Split Congruence Transformations", *IEEE/ACM Proc. DAC*, pp. 34-39, Jun. 1997.
- [11] B. D. Anderson and S. Vongpanitlerd, *Network analysis and synthesis*. Prentice-Hall, Inc., 1973
- [12] R. A. Rohrer and H. Nosrati, "Passivity considerations in stability studies of numerical integration algorithms," *IEEE Trans. on Circuits and Systems*, vol. CAS-28, no. 9, pp. 857-866, Sep. 1981.
- [13] C. L. Ratzlaff and L. T. Pillage, "RICE: Rapid interconnect circuit evaluation using AWE," *IEEE Trans. CAD*, vol. 13, no. 6, pp. 763-776, Jun. 1994.
- [14] V. Raghavan, J. E. Bracken and R. A. Rohrer, "AWESpice: A general tool for the accurate and efficient simulation of interconnect problems", *IEEE/ACM Proc. DAC*, Jun. 1992.
- [15] G. H. Golub and C. F. Van Loan, *Matrix Computations*. John Hopkins Univ. Press, 2nd ed., 1989.
- [16] E. S. Kuh and R. A. Rohrer, *Theory of Linear Active Networks*. Holden-Day Inc., 1967
- [17] L. W. Nagel, "SPICE2, a computer program to simulate semiconductor circuits," *Technical Report ERL-M520*, UC-Berkeley, May 1975.
- [18] T. L. Quarles, "The SPICE3 implementation guide", *Tech. Rep. Memo ERL-M89/44*, U. of California, Berkeley, 1989.

- [19] B. Krauter and L. T. Pileggi, "Generating sparse partial inductance matrices with guaranteed stability," *IEEE/ACM Proc. ICCAD*, pp. 45-52, Nov. 1995.
- [20] X. Huang, "Padé approximation of linear(ized) circuit responses", Ph.D. Thesis, Carnegie Mellon Univ., Nov. 1990.

# CIRCUIT NOISE EVALUATION BY PADÉ APPROXIMATION BASED MODEL-REDUCTION TECHNIQUES

Peter Feldmann<sup>1</sup> and Roland W. Freund<sup>2</sup>

*Bell Laboratories*

*Murray Hill, New Jersey, USA*

## Abstract

This paper introduces a new circuit noise analysis and modeling method. The noise analysis method computes an analytic expression of frequency, in rational form, which represents the Padé approximation of the noise power spectral density. The approximation can be carried out efficiently, to the required accuracy, using a variant of the PVL [1] or MPVL [2] algorithms. The new method is significantly more efficient than traditional methods for noise computation at numerous frequency points. In addition, it allows for a compact and cascadable modeling of noise that can be used in system-level simulations.

## 1. Introduction

Noise is a fundamental phenomenon in electronic circuits caused by the small fluctuations in currents and voltages that occur within the devices in the circuit. The fluctuations are due mainly to the discontinuous nature of electric charge. Determining the effects of noise is very important, as noise often represents the fundamental limit of circuit or system performance.

Noise analysis algorithms for circuits in DC steady-state have been available for a long time in programs such as SPICE [6]. The results of such programs is the noise power over a range of frequencies in tabulated form. Circuit or system designers typically reduce the information contained in the noise spectrum to one single number such as the *noise figure* [3]. While such compact representations offer good insight, and are very convenient for back-of-the-envelope calculations, CAD tools at both the circuit and the system level can take advantage of the more accurate and complete information available in the noise spectrum, and, in return, offer more accurate analysis.

In this paper we introduce an algorithm that computes the noise power spectral density as a closed-form rational expression. More specifically, the algorithm computes the Padé approximation of the noise power spectral density using the numerically robust and efficient Lanczos [4] method. The spectrum is computed to the required accuracy over the frequency range of interest. This

---

Authors are currently with <sup>1</sup>IBM T. J. Watson Research Center and <sup>2</sup>Bell Laboratories.

method is significantly more efficient than repeatedly evaluating the noise power over a fine frequency grid. However, the real advantage of the new approach consists in the compact noise models that get produced. These compact noise models with the spectral density specified as rational expressions can be accepted by the algorithm as input noise sources. In other words, the algorithm can consume its own output, thus lending itself to a hierarchical analysis methodology.

For example, the circuit designer designs an amplifier, analyzes it, and produces a high-level model, say, of its transfer function and of its output noise spectrum. The system designer uses models for all the components to perform, using the same noise analysis algorithm, a system-level simulation. No information is lost due to a too narrow interface between circuit-level and system-level models.

The paper is organized as follows. In the next section we review the circuit noise analysis problem. In Section 3 we reformulate the noise spectral density expression in a form compatible with Padé via Lanczos (PVL) model reduction. Section 4 discusses the application of the PVL algorithm to this particular problem. Finally, in Section 5, we illustrate the noise analysis problem with a few circuit examples and then present concluding remarks in Section 6.

## 2. Review of circuit noise analysis

The principal mechanisms of noise in integrated circuits are [3]: thermal, shot, and flicker noise. Mathematically, device noise is modeled by stochastic processes [5]. Stochastic processes represent ensembles of functions of time,  $n(t)$ , and are characterized in terms of statistical averages, such as the mean and autocorrelation, in the time domain, or the noise power spectral density in the frequency domain.

*Thermal* noise is caused by the thermal agitation of the electrons and occurs in almost all devices. Thermal noise is modeled by a parallel current source, the value of which is a zero mean stochastic process with a frequency-independent (*white-noise*) spectral density equal to

$$S_{\text{th}}(\omega) = 4kTG. \quad (1)$$

Here  $k$  is Boltzman's constant,  $T$  the absolute temperature, and  $G$  the conductance.

*Shot* noise is due to the fact that the current through a PN junction consists of discrete charge carriers randomly crossing a potential barrier. Shot noise in a junction is also modeled by a parallel, white-noise current source. The spectral density of shot noise is

$$S_{\text{sh}}(\omega) = 2qI_d. \quad (2)$$

Here  $q$  is the electron charge and  $I_d$  the average junction current.

*Flicker* (or  $\frac{1}{f}$ ) noise occurs in all active devices and even in some resistors due to mechanisms that are not well understood. Flicker noise is modeled by a stochastic process with a frequency-dependent spectral density

$$S_{\text{fl}}(\omega) = K_1 \frac{I^a}{(2\pi\omega)^b}. \quad (3)$$

Here  $I$  is the average direct current,  $K_1$  is a technology-dependent constant characterizing a particular device and process,  $a$  is a constant in the range  $0.5 - 2.0$ , and  $b$  is a constant of about one (hence the name  $\frac{1}{f}$  noise).

In this paper we only consider circuits with a constant excitation in DC steady-state. The methods presented below can also be applied to circuits with time-varying bias conditions [7], but such an extension is beyond the scope of the present paper. Moreover, noise is assumed to represent a “small” perturbation to the circuit.

The circuit equations under these assumptions are

$$f(x(t), \frac{d}{dt}x(t), b_0, n(t)) = 0. \quad (4)$$

Here  $x(t)$  is the vector of circuit variables, typically currents and voltages,  $b_0$  is the constant DC excitation, and  $n(t)$  is a vector of “small” perturbations caused by the noise sources.

Moreover, we assume that the circuit is stable, and therefore the solution,  $x_0$ , of the noiseless circuit is constant in time and satisfies

$$f(x_0, 0, b_0, 0) = 0, \quad (5)$$

since, obviously,  $\frac{d}{dt}x_0 = 0$ .

The response of the circuit in the presence of the perturbation  $n(t)$  is the perturbation,  $z(t)$ , of the DC solution. The perturbation  $z(t)$  satisfies

$$f(x_0 + z(t), \frac{d}{dt}z(t), b_0, n(t)) = 0. \quad (6)$$

Assuming that the noise perturbation is “small”, the first-order Taylor expansion of the circuit equations (6) around the DC solution is sufficiently accurate. Thus

$$f(x_0, 0, b_0, 0) + Gz(t) + C\frac{d}{dt}z(t) - Bn(t) = 0 \quad (7)$$

where

$$G = \left. \frac{\partial f}{\partial x} \right|_{x_0, 0, b_0, 0}, \quad C = \left. \frac{\partial f}{\partial \dot{x}} \right|_{x_0, 0, b_0, 0}, \quad \text{and} \quad B = - \left. \frac{\partial f}{\partial n} \right|_{x_0, 0, b_0, 0}$$

Subtracting (5) from (7), we are left with just the system

$$Gz(t) + C \frac{d}{dt} z(t) = Bn(t) \quad (8)$$

of linear differential equations for the perturbation signals.

The vector stochastic process  $n(t)$  is specified in terms of its frequency-domain cross-spectral density matrix  $S_{xx}(\omega)$ . The diagonal elements in  $S_{xx}(\omega)$  represent the power spectral density of each noise source, and the off-diagonal elements describe statistical coupling of noise signals. In practical cases,  $S_{xx}$  will almost always be a diagonal matrix.

The noise analysis problem reduces to that of the propagation of a stochastic process through a linear system. The general expression of the noise power spectral density,  $S_{yy}$ , at the output of the linear system is given by the well-known formula [5]

$$S_{yy}(j\omega) = H(j\omega)S_{xx}(j\omega)H^H(j\omega). \quad (9)$$

When only one output is analyzed,  $S_{yy}(j\omega)$  is just a scalar function of frequency. For more than one output,  $S_{yy}(j\omega)$  is a full matrix, the dimension of which is the number of outputs. The diagonal elements of  $S_{yy}(j\omega)$  represent the power spectral density of the noise at each output and the off-diagonal elements represent the cross-spectral density.

The many-to-one vector transfer function of the linear system from the noise sources to an output port of interest is

$$H(j\omega) = l^T(G + j\omega C)^{-1}B, \quad (10)$$

where  $l$  denotes the incidence vector that corresponds to the output port of interest. More generally, when more than one output is considered, we have a many-to-many matrix-transfer-function from noise sources to the outputs,

$$H(j\omega) = L^T(G + j\omega C)^{-1}B, \quad (11)$$

where  $L$  is the incidence matrix of the output ports.

From formula (9), using (10), we obtain the following expression for the noise power spectral density at the output of the system:

$$S_{yy}(j\omega) = l^T(G + j\omega C)^{-1}BS_{xx}(j\omega)B^T(G + j\omega C)^{-H}l. \quad (12)$$

The noise analysis method implemented in SPICE [6] evaluates this expression efficiently, for a given  $\omega$ , using the solution

$$x_a(j\omega) = (G + j\omega C)^{-H}l$$

of the adjoint system. In terms of  $x_a(j\omega)$ , (12) reduces to

$$S_{yy}(j\omega) = x_a^H(j\omega)BS_{xx}(j\omega)B^Tx_a(j\omega). \quad (13)$$

The SPICE noise-computation algorithm computes the noise power only at a given frequency. When we are interested in the spectrum of the noise, we must repeat the procedure for a large number of frequencies.

We now introduce a novel noise analysis method that computes a closed-form rational expression for the noise power spectral density over a wide frequency range. This method is more efficient than the classical method, when noise needs to be computed over a frequency grid. Moreover, the closed-form expression represents a compact model of the noise spectrum, and can be used hierarchically in system-level simulations.

### 3. Reformulated noise spectral density

For the following, it will be convenient to introduce the new variable  $s := j\omega$ . Note that, in order to be “physically” meaningful, the variable  $s$  has to be purely imaginary. For now, we thus assume that  $s$  is purely imaginary. Later on, we will drop this constraint and treat  $s$  as a general complex variable.

The new noise analysis and modeling algorithm relies on the computation of a Padé approximation of the noise power spectral density expression (12). The Padé approximation of a general transfer function expression of the form

$$F(s) = \tilde{l}^T (\tilde{G} + s\tilde{C})^{-1} \tilde{r}, \quad (14)$$

where  $\tilde{r}$  and  $\tilde{l}$  are vectors of length  $\tilde{N}$ , and  $\tilde{G}$  and  $\tilde{C}$  are  $\tilde{N} \times \tilde{N}$  matrices, can be computed efficiently with the PVL (Padé via Lanczos) algorithm [1]. At first glance, it appears that noise-type transfer functions (12) are very different from (14). However, we will show that there are vectors  $\tilde{r}$ ,  $\tilde{l}$  and matrices  $\tilde{G}$ ,  $\tilde{C}$  so that the functions (12) and (14) agree for all purely imaginary  $s$ , i.e., for all physically meaningful values of  $s$ .

First, consider the case when the noise sources are all white. Then  $S_{xx}$  is not a function of the frequency, and thus the function (12) reduces to

$$F(s) = l^T (G + sC)^{-1} B S_{xx} B^T (G + sC)^{-H} l. \quad (15)$$

Here we have used the new variable  $s = j\omega$ . We rewrite (15) by introducing two new vectors,  $u$  and  $v$ , as follows:

$$\begin{aligned} F(s) &= l^T u, \\ v &= (G + sC)^{-H} l, \\ u &= (G + sC)^{-1} B S_{xx} B^T (G + sC)^{-H} l \\ &= (G + sC)^{-1} B S_{xx} B^T v. \end{aligned} \quad (16)$$

The vectors  $u$  and  $v$  represent, therefore, the solution of a system of linear equations:

$$\begin{bmatrix} 0 & (G + sC)^H \\ G + sC & -B S_{xx} B^T \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} l \\ 0 \end{bmatrix}. \quad (17)$$

From (16) and (17) we obtain the expression

$$F(s) = [l^T \ 0] \left( \begin{bmatrix} 0 & G^T \\ G & -BS_{xx}B^T \end{bmatrix} + s \begin{bmatrix} 0 & -C^T \\ C & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} l \\ 0 \end{bmatrix}. \quad (18)$$

Note that (18) is exactly of the form (14), and thus amenable to PVL reduction.

Unfortunately, as shown in the previous section, not all noise sources are white. In order to be able to treat more general noise sources, we actually consider a more general class of noise-type transfer functions. More precisely, we study functions of the form

$$F(s) = l^T (G + sC)^{-1} B P^{-1}(s) B^T (G + sC)^{-H} l. \quad (19)$$

Here,  $l$  is a real vector of length  $N$ ,  $G$  and  $C$  are real  $N \times N$  matrices,  $B$  is a real  $N \times M$  matrix, and  $P(s)$  is a *matrix polynomial*,

$$P(s) = P_0 + P_1 s + P_2 s^2 + \cdots + P_L s^L, \quad (20)$$

whose coefficients  $P_i$ ,  $i = 0, 1, \dots, L$ , are  $M \times M$  matrices. We assume that  $P_L$  is not a zero matrix, so that  $L$  is the *degree* of the matrix polynomial  $P(s)$ . The form (19) can express practically all interesting noise power spectral densities. The degree  $L$  itself can be arbitrary; however, the cases of low degree such as  $L = 0, 1, 2$  are the most important ones. For example, for  $L = 0$  and  $P_0 = S_{xx}^{-1}$ , the function (19) reduces to the case (15) of white noise. The flicker noise frequency-dependent power spectral density (3) can also be well approximated by an expression of form (19) by expanding the denominator into a power series as follows:

$$S_{ff}(\omega) = K_1 I^a (c_0 + c_1 s + c_2 s^2 + \dots)^{-1}. \quad (21)$$

Rewriting the noise-type transfer function  $F(s)$  given by (19) in the form (14) then allows us to compute Padé-based reduced-order models for  $F(s)$  by simply applying the PVL algorithm to the representation (14) of  $F(s)$ .

Next, we show how to transform (19) to form (14). Consider the linear system

$$\begin{bmatrix} 0 & (G + sC)^H & 0 & 0 & \cdots & 0 \\ G + sC & 0 & B & 0 & \cdots & 0 \\ 0 & B^T & P_0 + sP_1 & sP_2 & \cdots & sP_L \\ 0 & 0 & sI & -I & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & sI & -I \end{bmatrix} \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \\ \vdots \\ z_L \end{bmatrix} = \begin{bmatrix} l \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad (22)$$

From the last  $L - 1$  blocks of equations in (22), it follows that

$$z_i = sz_{i-1} \quad \text{for all } i = 2, 3, \dots, L, \quad (23)$$

and thus

$$z_i = s^{i-1} z_1 \quad \text{for all } i = 2, 3, \dots, L. \quad (24)$$

Using the third block of equations in (22), together with (24) and (20), we get

$$\begin{aligned} B^T y &= -(P_0 + sP_1)z_1 + sP_2z_2 + \cdots + sP_Lz_L \\ &= - (P_0 + sP_1 + s^2P_2 + \cdots + s^LP_L)z_1 \\ &= -P(s)z_1. \end{aligned} \quad (25)$$

By the first two blocks of equations in (22), we have

$$\begin{aligned} y &= (G + sC)^{-H}l, \\ x &= -(G + sC)^{-1}Bz_1. \end{aligned} \quad (26)$$

Combining (25) and (26), we get

$$x = (G + sC)^{-1}B(P(s))^{-1}B^T(G + sC)^{-H}l. \quad (27)$$

Next, we observe that, for purely imaginary  $s$ , the linear system (22) can be rewritten in the form

$$(\tilde{G} + s\tilde{C})\tilde{x} = \tilde{l}. \quad (28)$$

Here  $\tilde{x}$  and  $\tilde{l}$  are vectors of length  $\tilde{N}$  defined by

$$\tilde{x} = \begin{bmatrix} x \\ y \\ z_1 \\ \vdots \\ z_L \end{bmatrix} \quad \text{and} \quad \tilde{l} = \begin{bmatrix} l \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad (29)$$

and  $\tilde{G}$  and  $\tilde{C}$  are  $\tilde{N} \times \tilde{N}$  matrices given by

$$\tilde{G} = \begin{bmatrix} 0 & G^T & 0 & 0 & \cdots & 0 \\ G & 0 & B & 0 & \ddots & 0 \\ 0 & B^T & P_0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & -I & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 0 & -I \end{bmatrix} \quad (30)$$

and

$$\tilde{C} = \begin{bmatrix} 0 & -C^T & 0 & 0 & \cdots & 0 \\ C & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & P_1 & P_2 & \ddots & P_L \\ 0 & 0 & I & 0 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & I & 0 \end{bmatrix}. \quad (31)$$

Using (19) and (28)–(31), it follows that

$$\begin{aligned} F(s) &= l^T(G + sC)^{-1}B(P(s))^{-1}B^T(G + sC)^{-H}l \\ &= l^T x = l^T \tilde{x} = l^T (\tilde{G} + s\tilde{C})^{-1} \tilde{l}. \end{aligned} \quad (32)$$

This shows that, for purely imaginary  $s$ , the noise-type transfer function (19) is indeed of the form (14) with  $\tilde{l} = \tilde{r}$ ,  $\tilde{G}$ , and  $\tilde{C}$  defined in (31).

Of particular interest are several special cases:

◇ The case  $L = 1$ .

Here  $\tilde{l}$ ,  $\tilde{G}$ , and  $\tilde{C}$  reduce to

$$\tilde{l} = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix}, \quad \tilde{G} = \begin{bmatrix} 0 & G^T & 0 \\ G & 0 & B \\ 0 & B^T & P_0 \end{bmatrix}, \quad \tilde{C} = \begin{bmatrix} 0 & -C^T & 0 \\ C & 0 & 0 \\ 0 & 0 & P_1 \end{bmatrix}. \quad (33)$$

◇ The case  $L = 0$ .

This is the case (15) that all noise sources are white. It is covered by (33) with  $P_0 = S_{xx}^{-1}$  and  $P_1 = 0$ . However, in this case, we eliminate the third block rows and columns in (33) and obtain

$$\tilde{l} = \begin{bmatrix} l \\ 0 \end{bmatrix}, \quad \tilde{G} = \begin{bmatrix} 0 & G^T \\ G & -B^T S_{xx}^{-1} B \end{bmatrix}, \quad \tilde{C} = \begin{bmatrix} 0 & -C^T \\ C & 0 \end{bmatrix}. \quad (34)$$

This is exactly the form arrived at in (18).

## 4. Application of PVL

Now that we have shown how to reformulate noise-type transfer functions  $F(s)$  given by (19) in the “PVL” form (14), it is straightforward to employ PVL to generate reduced-order models. Recall that, in our case,  $\tilde{l} = \tilde{r}$  in (14).

First, we choose a real expansion point  $s_0$  and compute  $\mathcal{L}$  and  $\mathcal{U}$  via the factorization

$$\tilde{G} + s_0 \tilde{C} = \mathcal{L} \cdot \mathcal{U}. \quad (35)$$

Then, setting  $s = s_0 + \sigma$ , we rewrite (14) as follows:

$$\begin{aligned} F(s_0 + \sigma) &= \tilde{l}^T (\tilde{G} + s_0 \tilde{C} + \sigma \tilde{C})^{-1} \tilde{l} \\ &= (\mathcal{U}^{-T} \tilde{l})^T (I + \sigma \mathcal{L}^{-1} \tilde{C} \mathcal{U}^{-1})^{-1} (\mathcal{L}^{-1} \tilde{l}). \end{aligned} \quad (36)$$

Next, we apply the Lanczos process to the matrix  $A = \mathcal{L}^{-1} \tilde{C} \mathcal{U}^{-1}$ , using  $b = \mathcal{L}^{-1} \tilde{l}$  and  $c = \mathcal{U}^{-T} \tilde{l}$  as the right, respectively left, starting vector. After running the Lanczos process for  $n$  iterations, we obtain an  $n \times n$  tridiagonal matrix,  $T_n$ , such that the function

$$F_n(s_0 + \sigma) = (c^T b) \cdot e_1^T (I + \sigma T_n)^{-1} e_1, \quad (37)$$

where  $e_1$  represents the first unit vector of length  $n$ , is just an  $n$ -th *Padé approximant* to  $F(s_0 + \sigma)$ . More precisely,  $F_n(s_0 + \sigma)$  is a rational function of  $\sigma$  with numerator polynomial of degree at most  $n - 1$  and denominator polynomial of degree at most  $n$  such that

$$F_n(s_0 + \sigma) = F(s_0 + \sigma) + O(n^{q(n)}), \quad (38)$$

where  $q(n)$  is maximal. In the generic case,  $q(n) = 2n$ . Note that (38) just states that the Taylor expansions of  $F_n$  and  $F$  about the expansion point  $s_0$  agree in as many leading Taylor coefficients as possible. We note that all quantities involved in the Lanczos process are real, as long as the coefficient matrices  $P_0, P_1, \dots, P_L$  of (20) are real, which is usually the case.

We observe that the reduced-order model for the noise spectral density of a circuit module will always have the form in (37), which results from the PVL algorithm. If the reduced-order models of circuit modules are used in higher-level simulations, expressions of the form in (37) appear in the  $S_{xx}(s)$  noise source spectral density matrix of the system simulation. The resulting output noise spectral density of the system will have the form

$$F(s) = l^T (G + sC)^{-1} B (P_0 + sP_1)^{-1} B^T (G + sC)^{-H} l. \quad (39)$$

This form is compatible with PVL, as shown for the special case  $L = 1$  in (33).

Finally, we make some comments regarding properties of the PVL algorithm specific to its application to “noise”-type problems. So far, we have made no assumptions on the matrix polynomial

$$P(s) = P_0 + P_1 s + P_2 s^2 + \cdots + P_L s^L. \quad (40)$$

If the function  $F(s)$  describes the noise power spectral density of a circuit, then  $P(s)$  needs to be such that

$$F(j\omega) > 0 \quad \text{for all } \omega > 0. \quad (41)$$

Ideally, the PVL algorithm is stopped as soon as the Padé approximant  $F_n$  has converged to  $F$  in the frequency range of interest, i.e., if

$$|F(j\omega) - F_n(j\omega)| \leq \text{tol} \quad \text{for all } \omega \in [\omega_{\min}, \omega_{\max}]. \quad (42)$$

Together with (41), this ensures that the Padé-based reduced-order model

$$F_n(s_0 + \sigma) = (c^T b) \cdot e_1^T (I + \sigma T_n)^{-1} e_1 \quad (43)$$

satisfies

$$F_n(j\omega) \gtrsim 0 \quad \text{for all } \omega \in [\omega_{\min}, \omega_{\max}]. \quad (44)$$

This observation is important if we want to use the reduced-order model as noise sources in a high-level simulation.

As a final note, we remark that the discussion in this and previous sections can be generalized for the computation of the cross-spectral density matrix of multiple outputs. In the multiple output case the  $l$  and  $\tilde{l}$  vectors become matrices and MPVL [2] is used instead of PVL.

## 5. Examples

We applied the noise computation algorithm to a number of circuits. The first example is the 741 operational amplifier. The size of the problem is 55 variables. Figure 1 shows the exact transfer function of the amplifier compared to the PVL reduced-order models of orders 16 and 20. The order 20 approximation captures the behavior of the amplifier almost exactly. Figure 2 shows the amplifier output noise power spectral density over the same frequency range. Here a Padé approximation of order 5 is already sufficient to capture the noise spectrum.

The next example is a 5-th order Cauer filter that uses ten 741 opamps as building blocks. The total size of the problem is 463 variables. Figures 3 and 4 show the transfer function and the output noise spectrum computed exactly and with PVL. We observe that we need roughly the same number of iterations to obtain an almost perfect match of both the transfer function and the noise spectrum.

The final example is a bandpass filter derived from a 3-rd order Chebyshev low-pass prototype, and implemented with single amplifier biquads. It also uses the 741 opamp as a building block. The problem size is 147. Figures 5 and 6 show the transfer function and the output noise spectrum computed exactly and with PVL. We observe that we need 18 iterations to match the transfer function and only 14 to match the noise spectrum.

## 6. Conclusions

In this paper we have introduced a new noise analysis method that computes the noise power spectral density of a circuit node or the cross-spectral density of

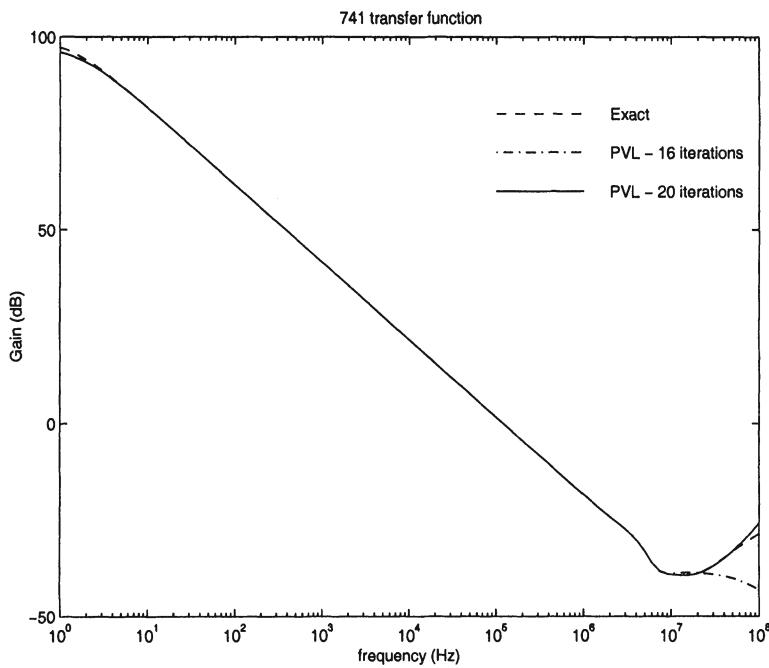


Figure 1. 741 gain.

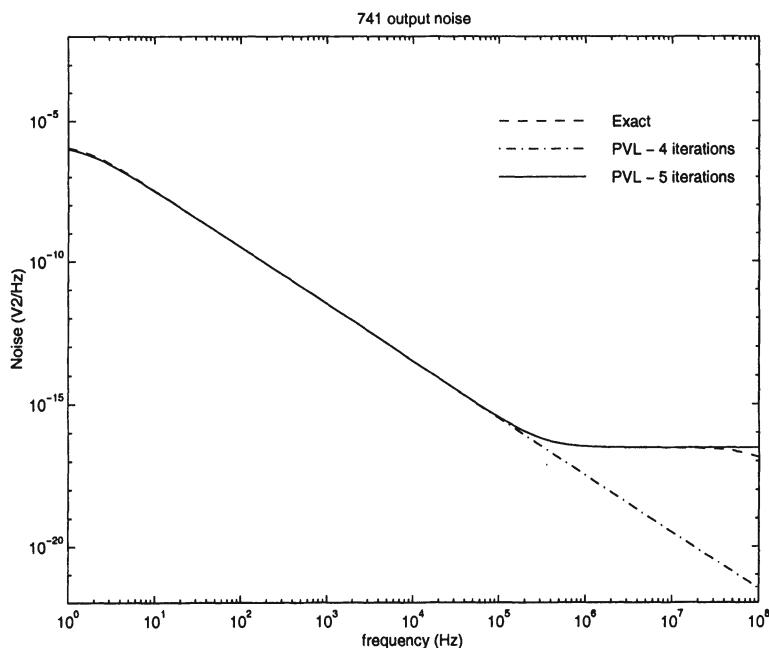


Figure 2. 741 noise.

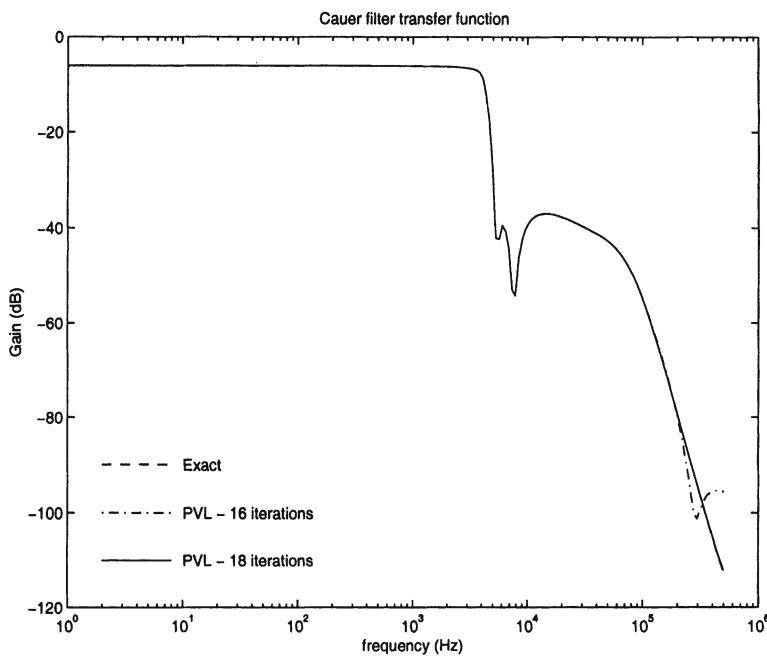


Figure 3. Cauer filter transfer characteristic.

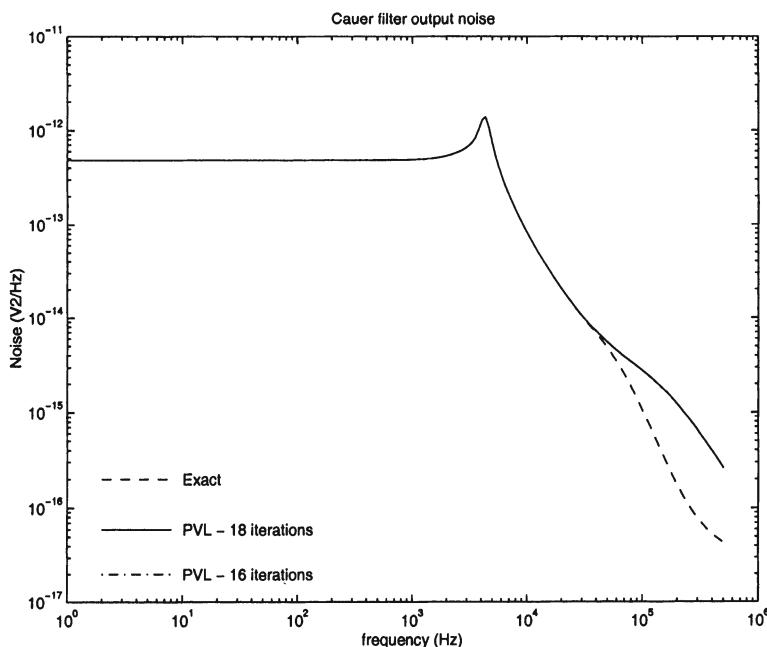


Figure 4. Cauer filter noise.

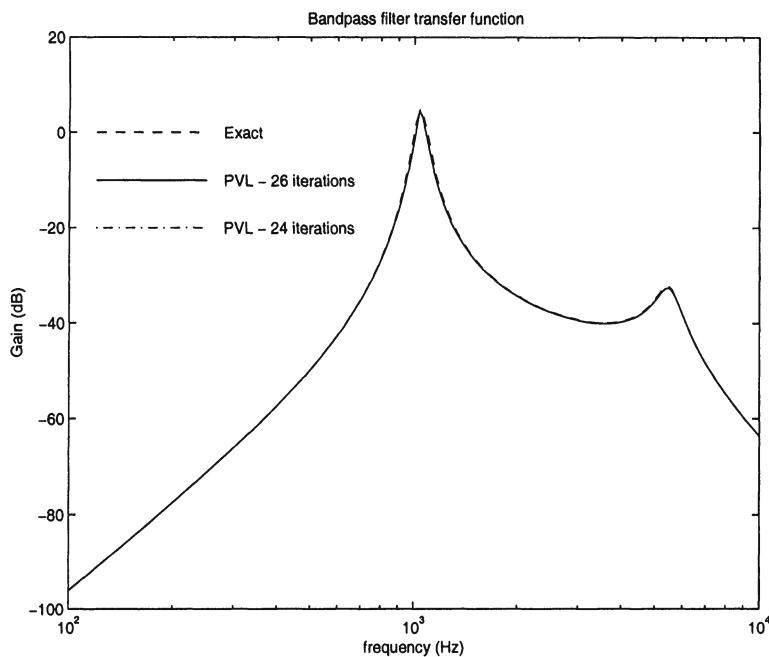


Figure 5. Bandpass filter transfer characteristic.

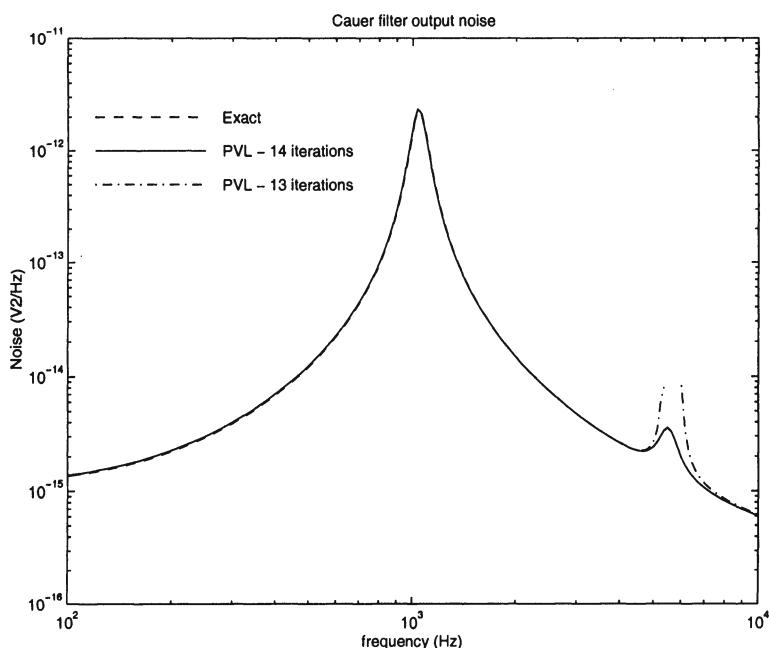


Figure 6. Bandpass filter noise.

a number of nodes. The results are presented in the form of a closed-form polynomial rational function of frequency, which represents a Padé approximation of the true noise spectral density. This method is significantly more efficient than the classical noise analysis method for predicting noise over a range of frequencies. The main advantage of the method, however, is the fact that it produces a reduced-order model of the noise generated by the circuit under analysis. This model can then be employed, using the same algorithm in a system-level analysis. The noise analysis algorithm accepts the noise source power spectral density in a rational polynomial form. This form covers practically all possible noise sources of interest.

## Acknowledgments

We would like to thank our colleagues: Peter Kinget, David Lee, Jaijeet Roychowdhury, and László Tóth, for useful discussions and help with examples.

## References

- [1] P. Feldmann and R. W. Freund, *Efficient linear circuit analysis by Padé approximation via the Lanczos process*, IEEE Trans. Computer-Aided Design **14** (1995), 639–649.
- [2] P. Feldmann and R. W. Freund, *Reduced-order modeling of large linear sub-circuits via a block Lanczos algorithm*, Proc. 32nd Design Automation Conference, ACM, New York, 1995, pp. 474–479.
- [3] P. R. Gray and R. G. Meyer, *Analysis and Design of Analog Integrated Circuits* Third edition, New York, N.Y.: John Wiley, 1993.
- [4] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards **45** (1950), 255–282.
- [5] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, Third edition, New York, N.Y.: McGraw-Hill, 1991.
- [6] R. A. Rohrer, L. Nagel, R. Meyer, and L. Weber, *Computationally efficient electronic-circuit noise calculations*, IEEE J. Solid State Circuits **SC-6** (1971), 204–213.
- [7] J. Roychowdhury and P. Feldmann, *A new linear-time harmonic balance algorithm for cyclostationary noise analysis in RF circuits*, Proc. Asia and South-Pacific Design Automation Conference, January 1997.

## Part VI

# PHYSICAL DESIGN

Physical Design Overview <i>Ernest S. Kuh and Chi-Ping Hsu</i>	467
Floorplan Design Using Annealing ( <i>ICCAD 1984</i> ) <i>Ralph H.J.M. Otten and Lukas P.P.P. van Ginneken</i>	479
GOALIE: A Space-Efficient System for VLSI Artwork Analysis ( <i>ICCAD 1983</i> ) <i>Thomas G. Szymanski and Christopher J. Van Wyk</i>	489
Gordian: A New Global Optimization/ Rectangle Dissection Method for Cell Placement ( <i>ICCAD 1988</i> ) <i>Jürgen M. Kleinhans, Georg Sigl and Frank M. Johannes</i>	499
Exact Zero Skew ( <i>ICCAD 1991</i> ) <i>Ren-Song Tsay</i>	509
Efficient Network Flow Based Min-Cut Balanced Partitioning ( <i>ICCAD 1994</i> ) <i>Honghua Hannah Yang and D. F. Wong</i>	521
Rectangle-Packng-Based Module Placement ( <i>ICCAD 1995</i> ) <i>Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake and Yoji Kajitani</i>	535

# **PHYSICAL DESIGN OVERVIEW**

**Ernest S. Kuh**

*University of California at Berkeley  
Berkeley, California  
U.S.A.*

**Chi-Ping Hsu**

*Get2Chip Inc.  
San Jose, California  
U.S.A.*

## **1. Introduction**

Since the early 70's, physical design has been a crucial part of chip design. Its origin came from the need for obtaining optimum electronic packaging, which involves the placement of components and modules on boards and interconnecting the pins on modules. Most early approaches were by and large empirical and ad. hoc. One major exception is perhaps Lee's maze router [8], which has proven to be a powerful computational tool for routing. In the late 70's more analytical tools for layout began to evolve both from industry and the universities. The field of physical design gradually attracted sizeable research interests.

The adoption of automated physical design tools in the semiconductor industry has accelerated since mid 1980's. This is mainly driven by the number of transistors available on a single chip. As we write this paper today in the year 2002, process technology is at 90 nanometers in critical process dimension. This is 2500 times improvement over 1985. In terms of layout objects the increase is from the low thousands to high millions for a real-world chip. As a result, physical design today is totally dependent on automated tools. The quality of physical design tools determines the competitiveness and cost of the final product. The problem sizes are so large that no one can tell whether a generated solution by the software is good or bad until a better solution is presented. The fact that the physical design problem is NP-Hard makes the research of physical design algorithms extremely interesting. This can be seen by the investment made by the electronics industry. During this time period, the electronics design automation industry (EDA) has grown from \$50M to over \$4B in size.

The ICCAD Conferences started in 1983, have published many outstanding papers in the field. The present chapter includes six such papers, which span some important sub-areas: partitioning, placement, floorplanning, and clock skew. It should be pointed out that they by no means cover all areas of interest in physical design. As an example, neither global routing nor detailed routing

is included. Also, some crucial issues of current interests in deep sub-micron design, i.e., timing-driven layout, the important consideration of interconnect, and the integration between physical design and logic synthesis are completely left out.

There exist previous review papers and edited books, for example, those contributed by Breuer [9] in 1979, Soukup [10] in 1981, Hu and Kuh [11] in 1985, Ohtsuki [12] in 1986, Kuh and Ohtsuki [13] in 1990, Camposano and Pedram [7] in 2000, and more recently excellent textbooks by Lengauer [14], Sherwani [15], and Sarrafzadeh and Wong [16], etc.

Our paper will first make brief comments on each of the six papers included in the Chapter. Furthermore, it will mention some of the related later work because much work has appeared since 1995 (the latest among the selected papers), especially with respect to the interest in deep sub-micron technology. Some practical aspects of various problems will also be mentioned. Finally, we will treat briefly an important subject on the integration of physical design and logic synthesis.

## 2. Topics on selected papers

### 2.1 Partitioning (with comments on [1])

Partitioning is an important part of chip and circuit design. It is crucial in the key problems in physical design, for example, in placement and floorplanning to be discussed in Sec. 2.2 and Sec. 2.3. The input of the problem is usually defined as a graph with vertex representing a device, a small cell, or a larger module in a circuit or chip. The edge of the graph gives the interconnect information, i.e., the number of connecting lines between two cells. The problem is to partition the graph into two or more parts so that the total edge cut is minimum. The partitioned parts are called partitioning subgraphs. When the partition is two way, we call it bipartition. Often it is required that the partitioned two subgraphs are balanced, i.e. each has about the same size. In [17], a ratio cut has been proposed to assign the partitioned sizes as variables and balance the partition with a rational function. Partition can also be discussed with respect to hypergraphs, i.e., vertex are connected by multi-terminal nets.

Because partitioning has broad applications in many fields, there exist well-known techniques and algorithms, for example, that of Kernighan and Lin [18] together with its efficient implementation by Fiduccia and Mattheyses [19], the spectral method [20], and the simulated annealing method. Paper [1] presents a method using network flow. The essence of the proposed method is as follows: From the given graph, a much larger associated graph called the flow network is introduced. In the flow network there is a source  $s$  and a sink  $t$ . We consider the flow from  $s$  to  $t$ . The well-known max-flow min-cut algorithm used in operations research is applied to the partitioning problem, which leads to an optimum partitioning with respect to  $s$  and  $t$ . This in turn gives a solution to the origi-

nal bipartitioning problem with minimum cut; however, the result is usually not balanced, i.e., the bipartition leads to two subgraphs of vastly different size.

The next step of the method is to introduce a balanced-bipartition in the flow network. This calls for collapsing one of the subgraphs to either s or t and repeating the min-net-cut. The process continues until the recombined parts reach a stage so that two subgraphs of the flow network are balanced. An efficient implementation of the iterative process is proposed in the paper. Computation complexity and illustrative examples are given. It is demonstrated that using the standard MCNC benchmark examples the method outperforms both the Kernighan-Lin method and the spectra method.

The partitioning algorithms developed so far have been used widely in real-world problems. In most commercial placement software, partitioning is one of the fundamental structures used to find a global solution. By its nature, it is a hierarchical divide and conquer technique. It suffers from the lack of details inside each partition. As a result, partitioning is always followed by other local optimization techniques to improve its results. It also uses a simple representation for the problem, i.e. minimize the signal crossing the partition boundaries. While this is an easy objective function, it is more complicated when one has to consider timing of the circuits, and, in addition, it is much more complicated when the distributed interconnect topology impacts timing as well. Recently, there are some interesting works on multilevel techniques [21, 22]. The field of partitioning will need more advancement to address these new issues.

## 2.2 Placement (with comments on [2])

When the placement problem was first introduced in physical design, it was formulated as placing cells of point dimension for a given connection specification in terms of an incidence matrix or connection matrix. The m point cells are placed on a rectangular array evenly spaced locations. This is often referred as the quadratic placement problem. In early days of design automation quadratic placement was needed for both the standard cell and gate array designs. Various methods were used, however it was not until the program Gordian [2] was introduced that the industry began to realize the virtue of a good placement method. The key to Gordian's success is the combination of bipartitioning and a global optimum placement, used top down until all modules are placed. Early approaches of using the same strategy includes the work on min-cut partitioning and resistance network solution [23], and the Proud Program [24]. Proud differs from Gordian in that each placement optimization imposes a boundary constraint instead of a centroid constraint, i.e., the center of the region is used to guide the optimization process. The implementation of the algorithm is superior and the quality of the result is excellent. Both Proud and Gordian have

been used in commercial products. In addition, Gordian has the capability of handling modules of finite area by means of an exhaustive slicing optimization.

Later it became obvious that in physical design some consideration must be given to timing as well. Several contributions appeared [25, 26, 27], among them Ritual [27] represents a combined continuous and discrete part. It introduces timing constraints on total path delays in global placement, followed by discrete space optimization such that one cell remains in a region. The former is accomplished by the method of Lagranging relaxation, and the latter by hierarchical partitioning.

Another problem, which has emerged but still unsolved is the consideration of the effects of interconnect delay on placement. In deep sub-micron design, circuit delay is dominated by interconnect delay. Interconnect wires also occupy sizeable part of the chip area. Thus, it is important to consider interconnect topology in addition to the gate placement. An attempt has been made in this direction [28] and may prove to be fruitful in the future.

Among all the papers, the Proud and Gordian papers have made most significant impact on the EDA industry. Today essentially all commercial placement tools are based on this technique. It represented a major breakthrough in standard cell placement technology in the last 15 years. More recent contributions include [29, 30, 31, 32]. As the complexity of chip permits the integration of over 50 million transistors on a single chip, the so called System-On-a-Chip (SoC) has become a reality. Along with the shrinking of process geometries are the added layers for interconnect. Today 8 to 12 layers of metal or copper on the chip for interconnect are very common. The placement problem becomes one that mixes many large pre-existing blocks with different shapes plus millions of small standard cells. Furthermore some of the blocks allow signals to route over them and some do not. This calls for new approaches for the placement problem.

### 2.3 Floorplanning (with comments on [3, 4])

Floorplanning can be considered as a generalization of point (cell) placement discussed in the last Section, in which point cells become modules of finite area and given aspect ratio. In early days, it is also referred to as building-block placement [33]. Given  $m$  rectangular modules of fixed height and width, a floorplan is a non-overlapping placement of the  $m$  modules into a rectangular region. Like point cell placement, an interconnect specification among modules is given in terms of incidence matrix. The objective is to find a floorplan with minimum total area among all possible floorplans. Some authors also consider the case that the cost function is a combination of area and the interconnect. Additional constraints may be imposed, for example, upper and lower bounds of the aspect ratio. Paper [3] deals with optimization using simulated annealing.

It is well known that simulated annealing is time consuming and, in general, cannot find the optimum solution. Paper [3] proposes a clever algorithm to speed up the process. It, however, does not give any example, neither does it discusses the computation complexity. Otten had done previous good work on floorplanning such as slicing floorplan, shape functions, and the polynomial time algorithms [34, 35].

In contrast, paper [4] introduces a brand new theory on floorplanning using sequence pairs. Given  $m$  modules and a floorplan, there exists a sequence pair which consists of two sequences, one represents a permutation of the other. Given a sequence pair and the shapes of all modules, there is a unique topology for the floorplan. The topology here refers to the fact that the blocks can be shifted locally without changing the relative positions between the blocks. Thus, the solution space of the optimum floorplan is finite, i.e.  $(m!)^2$ . Furthermore, each sequence pair can be mapped to the unique floorplan in  $O(m^2)$  time, and at least one of the  $(m!)^2$  floorplan represents the optimal solution. The proof is by means of two constraint acyclic digraphs defined by the floorplan. Several large examples are illustrated. The above can be generalized to include soft modules (arbitrary aspect ratio) and pre-placed fixed modules.

The publication of sequence pair representation inspired many new development in floorplanning. In [36], Nakateke et al. introduced a bounded slicing grid (BSG), a  $n$ -by- $n$  checkerboard grid, for a list of  $n$  blocks. More recently, Gao et al. [37] proposed an ordered tree structure (O-tree) for a packed floorplan (all blocks are packed toward one corner of the floorplan). The number of O-trees is smaller than the number of slicing trees and yet it guarantees to include all optimal packing of rectangles. This means floorplanners based on O-trees have the speed of slicing-tree based floorplanners and yet produce higher quality floorplans compared to those by sequence pair and BSG. In [38, 39], a Corner Block list together with an equivalent representation for a mosaic floorplan was proposed that covers all slicing and non-slicing floorplans with no empty room. In [40], Yao et al. demonstrated that a mosaic floorplan has a one-to-one correspondence with a twin binary tree.

In chip design, one starts with chip planning based on various specifications such as speed, power, etc. Often IP blocks are used as major components. These and other modules and components represent a floorplanning problem. The advance in floorplanning in physical design discussed above is of great value to the process.

Today, commercial floorplanning tools for a real-world design has to consider three additional elements: layout hierarchy planning, power distribution, and clock distribution. First, due to the high integration in a SoC design, more often than not, one has to do some form of hierarchical physical design, even though most designers avoid hierarchical design due to its inefficiency in die size and high complexity in generating and maintaining the hierarchical bound-

ary constraints. Although the hierarchical design problem has existed for many years, there has been very little research advancement in this area.

The second real-world concern is to minimize the total power consumption. Among several techniques, clock gating by turning off portions of a design during idle period and applying multiple voltage levels for different parts of the chip are common practices. With the large electrical current required, so is the area used for power distribution on a chip. The quality of power distribution is very critical for the reliable operation of the circuit. The large area required for power distribution, the different voltages and different power networks for different functions, and the need for quality power source at every circuit locations make power distribution task extremely tedious and challenging.

The third element of real-world floorplan design has to do with the clock distribution network. Clock distribution network consumes almost half of the chip power. A good clock distribution scheme, either zero skew technique as 2.4 or mesh distribution with pre-designed clock buffer rows, becomes another critical floorplan design task. In order to lower the risk of power supply noise created by these large clock distribution networks on a chip, many designs require a separate power distribution network for those clock circuits.

As a result of these real-world considerations of design hierarchy, power distribution, and clock distribution, to create a good floorplan tool has been a continuing challenge for electronics design automation industry. All available commercial floorplanning tools today offer highly interactive environments with functions for power and clock planning, timing budgeting capability for hierarchical design, block placement, soft block pin assignment, global routing and congestion analysis, and voltage drop analysis. This expanded definition of floorplanning covers a wide variety of issues and hard to abstract it to a simple problem definition for the researcher. In reality, before one sees a better floorplan, one does not know it exists. One can easily see great impact of a good floorplan versus a bad floorplan and this is an area worthy of significant development in the future.

## 2.4 Zero Skew (with comments on [5])

In VLSI design, cycle time is one of the key specifications because it affects the timing performance. Improper clock skew could cause system malfunction. The clock period depends on the worst-case path delay and, in a crucial way, the clock skew. The H-tree clock routing is perhaps the most commonly used. Early work on minimizing clock skew includes heuristics on balancing the wire lengths [41, 42]. In paper [5], the problem is attacked head on for the aim of obtaining zero skew. The author was successful in obtaining zero skew based on computing delay using the Elmore delay model, and a depth first search algorithm. Next, it uses a recursive bottom up algorithm to balance the delay at

each junction based on the delay calculations of sub-trees. The paper also includes a generalization to buffered RC trees, and provides numerical examples to compare with earlier results based on balancing the path length.

Assuming one has solved the zero skew problem, it is natural to extend this technique to control the exact time when each clocking element receives the clock signal. With many of the designs pushing the performance limits of the process technology, “Retiming” by moving logic function across flip flop or latch boundaries in order to meet clock cycle time has become a technique of interest. However, moving logic function across storage elements impacts formal verification and test processes. A different but equivalent technique is to skew the clock intentionally to accommodate the longer paths by borrowing the allocated cycle time from upstream or downstream neighboring logic across the storage elements. This is sometimes called “useful skew”. Useful skew technique removes the impact on formal verification and test process at layout stage in a design process. It has generated some interest in the design community. However, most designers view this technique as interesting but few have actually used it in real design. For those who have used, most apply it to very small portions of a design.

There are two main concerns from chip designers on zero skew or useful skew technique: (1) the transistor parametric variation across the chip; (2) the accuracy of delay models. Designers worry about the transistor behavior changes due to the supply voltage variation and process parameter changes across the chip. They also worry about the deep sub-micron effects on the accuracy of Elmore delay model. Using clock meshes and special clock buffers distributed in a regular pattern across the chip to make sure storage elements near by will not have large clock skew is still a common practice. With other consideration as described in 2.3, the clock distribution is no longer a standalone problem. It is important to consider clock design in both floorplanning and placement stages and not just a separate design task.

## 2.5 Goalie [6]

Physical verification has been one of the first commercial application available from EDA industry since early 1980's. It does the most important process design rule checks before a design is “taped out” to manufacture facility. As the process technology advances, physical verification tool has to address much larger design with hundreds of millions of geometry. Furthermore, the process rules are becoming very complex with many conditional rules. Goalie is focused on memory efficient algorithms to enhance the capacity of the tool while maintaining flexibility for complex process rules.

Commercial physical verification tools have gone through several generations of major development since Goalie. The most notable development is in the

area of hierarchical processing of a design instead of flattening all the design data first. Since most modern layout designs are structured hierarchically and made of limited set of pre-designed cells, new generation of physical verification solutions at 2002 have significantly improved the capacity and runtime by at least two orders of magnitude by exploiting the repetitive nature of the designs.

Physical verification tool is fundamentally different from physical design tool. Verification tool focuses on accuracy, capacity, and speed of execution. The correctness of verification tool is most important. Physical design tool, on the other hand, focuses on finding the optimal quality of result in terms of die size, timing, and power, in addition to software capacity and runtime. The most challenging aspect of physical verification has been “logic versus schematic” (LVS). LVS is a process to verify that the final layout has faithfully implemented the intended logic design by extracting schematic from layout and comparing it to the original logic description. Although much advancement has been made in geometry processing, it remains to be a challenging task for software to identify the source of the LVS problem when a design mismatch happens.

### 3. Physical design and logic synthesis

Traditionally, logic synthesis uses “wireload” model to consider interconnect delay during logic synthesis process. Wireload model is a statistical model extracted per technology process per cell library to estimate maximum wire length for each signal. It has been successfully used for many years until the early 1990’s when interconnect delay became dominant and causes major timing inconsistencies between logic synthesis tools and physical design tools. Since then, wireload model became a term everyone blames for timing closure problems.

In early 1990’s, as the world of physical design moves to consider timing impact due to interconnect dominated delays, most consider physical solution by moving cells or blocks closer and modifying routing topologies to accommodate timing critical paths. For those timing issues that cannot be fixed by physical design techniques, a feedback to logic synthesis tools is necessary. The feedback process from physical tool to logical tool takes very long and tedious manual work and, worst yet, in many cases it does not converge in timing, i.e. logic synthesis tool thought the timing problem is corrected and passes the design to the physical tool but the physical tool still cannot find a feasible solution. Some designs required over 30 iterations between physical design and logic synthesis tools and still cannot reach a solution after many weeks of trials.

A significant industry-wide development to address this issue took place since 1994 at both physical design and logic synthesis EDA companies. This is what many companies touted as “SinglePass” or “RTL-to-GDS” solution — merging the logic synthesis and physical design in one process. Although this is a noble

goal, in reality, the problem is far from being solved even at year 2002. Today, there are only some successful convergence techniques that use incremental buffering and cell sizing techniques from traditional logic synthesis tools combined with incremental placement and routing capabilities. This solution eliminates many of the iterations between logic synthesis and physical design tools. However, it does only local and incremental optimizations. It does not improve the quality of the original design structure. Its goal is to find a feasible physical solution that does not deviate too much from the original goals set by the logic design process. Furthermore, it is based solely on heuristics without much theoretical foundation and it suffers from severe capacity limitations and extreme long runtimes. With the world moving to higher complexity designs and smaller geometries, this is clearly an area for new research. While this initial solution is effective in local convergence, it is far from the original goal of SinglePass or RTL-to-GDS solution that one shall expect. The industry needs to move the physical effects to higher level of logic synthesis processes to generate best logic structures to match the best physical design at the same time. Up to this time, logic synthesis tools have not made much improvement in it technology. In the years ahead, logic synthesis tools need significant fundamental improvements in its capacity, quality of result, and runtime. Eventually logic synthesis and physical design processes will be combined in one.

It is interesting to observe that while the EDA industry is investing heavily to improve physical design tools to include incremental logic synthesis, there have been very little research publications on this topic.

## Acknowledgments

The authors wish to acknowledge the comments and suggestions of Prof. C.K. Cheng and help in Latex of Prof. Q.J. Yu. The authors also wish to acknowledge the support of the Semiconductor research Corporation under contract 99-HJ-656.

## References

- [1] Yang, H., and D.F. Wong, " Efficient network flow based min-cut balanced partitioning", ICCAD-94, pp. 50-55.
- [2] Kleinhans, J.M., G. Sigl, and F.M. Johannes, "GORDIAN: A New Global Optimization/Rectangle Dissection Method for Cell Placement", ICCAD-88, pp. 506-513.
- [3] Otten, R.H.J.M. and L.P.P.P. van Ginneken, "Floorplan Design Using Simulated Annealing", ICCAD-84, pp. 96-98.
- [4] Murata,H., K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packng -Based Module Placement", ICCAD-95, pp. 472-479.
- [5] Tsay, R.S., "Exact zero skew", ICCAD-91, pp. 336-339.
- [6] Szymanski, T.G., and C. J. Van Wak, "GOALIE: A Space-Efficient System for VLSI", ICCAD-84, pp. 278-280.

- [7] Camposano, R, and M. Pedram, "Electronic Design Automation at the Turn of the Century: Accomplishments and Vision of the Future," IEEE Trans. CAD, Vol.19, No.12, December, 2000, pp. 1401-1403.
- [8] Lee, C.Y., "An algorithm for path connection and its applications," IRE Trans. Electro. Comput., vol. EC-10, pp. 346-365, 1961.
- [9] Breuer, A.D., "Theory and Techniques," Design automation of digital systems, vol.1. Prentice-Hall, New Jersey, 1979.
- [10] Soukup, J., "Circuit layout," Proc. IEEE, vol. 69, no. 10, pp. 1281-1304, Oct. 1981.
- [11] Hu, T.C. and E. S. Kuh, "VLSI Circuit Layout: theory and Design," IEEE Press, New York, 1985.
- [12] Ohtsuki, T., "Layout Design and Verification," Advances in CAD for VLSI, vol. 4, North Holland, 1986.
- [13] Kuh, E.S. and T. Ohtsuki, "Recent advances in VLSI layout," Proc. IEEE, vol 78, no.2, Feb. 1990.
- [14] Lengauer, T., "Combinatorial algorithms for integrated circuit layout," John Wiley & sons, England, 1990.
- [15] Sherwani, Naveed, "Algorithms for VLSI physical design automation," Kluver Academic Publishers, The Netherlands, 1993.
- [16] Sarrafzadeh, M. and C.K. Wong, "An Introduction to VLSI Physical Design," McGraw Hill, N.Y., 1996.
- [17] Wei, Y.C. and C.K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," IEEE Trans. on Computer-Aided Design, vol. 10, pp. 911-921, July 1991.
- [18] Kernighan, B.W. and S. Lin, "An efficient procedure for partitioning graphs," BSTJ, vol. 49, no. 2, pp. 291-307, 1970.
- [19] Fiduccia, C.M. and R.M. Mattheyses, "A linear-time heuristics for improving network partitions," Proc. 19th Design automation Conference, pp. 175-181, 1982.
- [20] Hagen, L. and A. B. Kahng, "Fast spectral methods for ratio cut partitioning and clustering, "Proc. ICCAD-91, pp. 10-13, 1991.
- [21] Alpert, C. J., J. Huang, and Kahng, A., "Multilevel Circuit Partitioning," Proc. 34 Design Automation Conf. 1997, pp. 530-533.
- [22] Ouyang, M. et.al. "Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem," IEEE Trans. CAD, Vol.21, No. 6, June 2002, pp. 685-693.
- [23] Cheng, C.K. and E.S. Kuh, "Module placement based on resistive network optimization," IEEE Trans. CAD, pp. 218-225, 1984.
- [24] Tsay, R.S., E.S. Kuh and C.P. Hsu, "PROUD: A fast sea-of-gates placement algorithm," Proc. 25th Design automation Conference, pp. 318-323, 1988.
- [25] Burstein, M. and M.N. Youssef, "Timing influenced layout design," Proc. 22nd Design Automation Conf., pp. 124-130, 1985.
- [26] Marek-Sadowska, M and S.P. Lin, "Timing-driven placement," Proc. ICCAD-89, pp. 94-97, 1989.
- [27] Srinivasan, A., K. Chaudhary and E.S. Kuh, "RITUAL: A performance-driven placement algorithm," IEEE Trans. CAS-II, vol. 39, no.11, Nov., 1992.
- [28] Fan M, A. Tabbara, and R.K. Brayton, "A force-directed micro-cell placer," Proc. ICCAD-2000, pp. 177-180.

- [29] Wang, M., X. Yang, and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits," ICCAD 2000, pp. 260-263.
- [30] Chan, T.F., et al, "Multilevel Optimization for Large-Scale Circuit Placement," ICCAD 2000, pp. 171-176.
- [31] Hur, S-W, and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," ICCAD 2000, pp. 165-170.
- [32] Yang, X., R. Kastner, and M. Sarrafzadeh, "Congestion Estimation During Top-Down Placement," IEEE Trans. CAD, Vol.21, No.1, Jan. 2002, pp. 72-80.
- [33] Dai, W-M, et al. "BEAR: A new building- block layout system," Proc. ICCAD-87, pp. 34-37, 1987.
- [34] Szepieniec, A.A. and R. Otten, "The genealogical approach to the layout problem," Proc. 17th Design Automation Conf. pp535-542, 1980.
- [35] Otten, R., "Efficient floorplan optimization," Proc. ICCD 1983, pp. 499-502.
- [36] Nakatake, S., et al. "Module Packing Based on the BSG-Structure and IC Layout Applications," IEEE Trans on CAD, pp. 519-530, June 1998.
- [37] Gao, P.N., T. Takahashi, C.K. Cheng, and T. Yoshimura "Floorplanning using a Tree Representation," IEEE Trans. on CAD, pp. 281-289, Feb. 2001.
- [38] Hong, X., G. Huang, Y. Cai, J. Gu, S. Dong, C.K. Cheng, and J. Gu, "Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan," ICCAD-2000, pp. 8-12.
- [39] Sakanushi, K. and Y. Kajitani, "The Quarter-State Sequence (Q\_Sequence) to Represent The Floorplan and Applications to Layout Optimization, " IEEE APCCAS 2000, pp. 829-832.
- [40] Yao, B., H. Chen, C.K. Cheng, and R. Graham, "Revisiting Floorplan Representations," Int. Symp. on Physical Design, 2001, pp. 138-143.
- [41] Jackson, M.A.B., A. Srinivasan, and E.S. Kuh, "Clock routing for high-performance IC's," Proc. Design Automation Conference, pp. 573-579, 1990.
- [42] Kahng, A., J. Cong, and G. Robins, "High performance clock routing based on recursive geometric matching," Proc. Design Automation Conference, pp. 322-327, 1991.

# FLOORPLAN DESIGN USING ANNEALING

Ralph H.J.M. Otten<sup>1</sup>

*Thomas J. Watson Research Center*

*IBM Corporation*

*P.O. Box 218*

*Yorktown Heights NY 10598, The United States of North-America*

Lukas P.P.P. van Ginneken

*Eindhoven University of Technology*

*Department of Electrical Engineering*

*5600MB Eindhoven, The Netherlands*

## Abstract

An application of simulated annealing to floorplan design or macro placement is described. It uses a minimum size configuration space without invalid solutions, a constant object function, an adaptive control schedule, and an indicator for proper convergence. Fast convergence and improved flexibility are the salient features of this approach.

## 1. Introduction

Floorplan design and the context for which the algorithms of this paper were developed have recently been described [3]. Section 1.1. briefly summarizes that discussion, using the terminology of that paper. Simulated Annealing<sup>1</sup> was introduced into layout design by Kirkpatrick, Gelatt, and Vecchi [1]. In section 1.2 a slightly restricted version of that algorithm is described, together with some basic facts that can be found in literature, though somewhat scattered. The restrictions make an automatically adaptive schedule possible, and the described implementation can therefore be used in a silicon compiler.

### 1.1 Floorplan design

A floorplan is the topology (a set of neighbor relations) of a partition of a geometrical plane figure. In this paper the geometrical figure is a rectangle, and so are the elements of the partition. Such configurations are called rectangle dissections. In layout design floorplans are used to capture data at intermediate stages of the design. In that context the enclosing geometrical figure represents

---

<sup>1</sup>Author is currently with Eindhoven University of Technology.

a module, sometimes the whole system, and the elements of the partition represent its submodules. With the restriction to rectangle dissections floorplans can be represented by polar graphs. The task of a floorplan design procedure is therefore to produce a suitable polar graph.

The data available to a floorplan design procedure consists of data more or less directly derivable from the input for the layout design program, and environment data estimated in previous stages of the layout design. Among the first kind of data there is always a part that can be called 'proximity data'. It somehow indicates which modules are preferably kept close together. Very often this data is present in the form of an incidence structure  $(\mathcal{M}, \mathcal{P}, \mathcal{N})^2$ , called a net list.

Also in this first group of data is the information about the shapes of the submodules:  $\Gamma$  assigns a shape constraint [3] to each module. Usually there is quite some freedom in laying out these submodules. This is what makes floorplan design in some sense a generalization of placement, where fixed objects have to get a location and an orientation. Though fixed shaped objects must be acceptable to a floorplan design method, it differs from classical placement algorithms in its capability of handling flexible shape objects as well. A previously designed environment may imply data such as a good shape for the whole module, and indications concerning the position of entry points of certain global nets.

## 1.2 The annealing algorithm

An annealing algorithm works on a state space, i.e. a set  $\mathcal{S}$  of states, on which a topology is defined. Each state represents an encoding of a configuration. An object function  $\epsilon : \mathcal{S} \rightarrow \mathbb{R}$  assigns a real number to each state. This number is interpreted as a quality indicator, in the sense that the lower this number the better is the configuration that was encoded in that state. By defining a set  $\mu$  of neighbor relations over  $\mathcal{S}$  (i.e.  $\mu \subseteq \mathcal{S} \times \mathcal{S}$ ) a topology is endowed to the state set  $\mathcal{S}$ . This relation is required to be symmetric and antireflexive, and its transitive closure has to be  $\mathcal{S} \times \mathcal{S}$ . The elements of  $\mu$  are called moves.

A selection probability  $\beta : \mu \rightarrow (0, 1] \subset \mathbb{R}$  is assigned to each move. It satisfies

$$\forall_{s \in \mathcal{S}} \left[ \sum_{s' \in s\mu} \beta(s, s') = 1 \right] \quad (1)$$

and

$$\forall_{(s, s') \in \mu} [\beta(s, s') = \beta(s', s)] \quad (2)$$

Another probability function, called the acceptance probability, depends not only on the move, but also on a positive real number, the control parameter. This function  $\alpha : \mu \times \mathbb{R}^+ \rightarrow (0, 1]$  has the following properties:

$$\forall_{(s, s') \in \mu} [\alpha((s, s'), t) \text{ is monotonous in } t] \quad (3)$$

$$\forall_{(s,s') \in \mu} \forall_{t \in \mathbb{R}^+} [\varepsilon(s) \geq \varepsilon(s') \Rightarrow \alpha((s,s'), t) = 1] \quad (4)$$

$$\forall_{(s,s') \in \mu} \left[ \lim_{t \rightarrow \infty} \alpha((s,s'), t) \rightarrow 1 \right] \quad (5)$$

$$\forall_{(s,s') \in \mu} \left[ \varepsilon(s) < \varepsilon(s') \Rightarrow \lim_{t \rightarrow 0} \alpha((s,s'), t) \rightarrow 0 \right] \quad (6)$$

$$\begin{aligned} \forall_{(s,s') \in \mu} \forall_{(s'',s''') \in \mu} \forall_{t \in \mathbb{R}^+} [((s,s'') \in \mu \wedge \varepsilon(s) < \varepsilon(s') < \varepsilon(s'')) \Rightarrow \\ \Rightarrow \alpha((s,s'), t) \alpha((s',s'''), t) = \alpha((s,s'''), t)] \end{aligned} \quad (7)$$

The following (infinite) Markov process uses the above probability functions to move from one state to another.

```

select present△state
  from S
  at random;
repeat
  select next△state
    from present△stateμ
    according to β;
  if random([0, 1]) < α((present△state, next△state, t)
  then present△state := next△state
until false;
```

This process has a number of important properties:

**P1.** The relative frequency of steps with  $\text{present}\triangle\text{state} = s$  is

$$\delta(s, t) = \frac{\alpha((s_o, s), t)}{1 + \sum_{s' \in S \setminus \{s_o\}} \alpha((s_o, s'), t)} \quad (8)$$

where  $\varepsilon(s_o) = \min_{s \in S} \varepsilon(s)$ . So, these relative frequencies are independent of  $\beta$ ! The function  $\delta : S \times \mathbb{R}^+ \rightarrow [0, 1]$  is the equilibrium distribution.

**P2.** For a sufficiently high value of the control parameter  $t$  the relative frequency is the same for all states.

**P3.** For sufficiently low value of the control parameter  $t$   $\text{present}\triangle\text{state}$  is almost exclusively a state with  $\varepsilon \approx \varepsilon(s_o)$ . If  $s_o$  is the only state with  $\varepsilon(s_o)$  (i.e.  $\varepsilon$  has a unique global optimum over  $S$ ,  $\text{present}\triangle\text{state}$  will be  $s_o$  for an arbitrarily large proportion of the time for  $t$  low enough.

Two aggregate functions are useful in characterizing the process. The first one is the average value of  $\varepsilon$  during the process:

$$E(t) = \langle \varepsilon \rangle = \sum_{s \in S} \delta(s, t) \varepsilon(s) \quad (9)$$

The other parameter, called the entropy, is defined as

$$S(t) = - \sum_{s \in \mathcal{S}} \delta(s, t) \ln(\delta(s, t)) \quad (10)$$

It is very easy to calculate the entropy for high values of  $t$ , because of property P2. Its value will be

$$S_\infty = \ln |\mathcal{S}| \quad (11)$$

If the system has only one global minimum the entropy will come arbitrarily close to 0. If there are several global minima the entropy will approach

$$S_0 = \ln |\{s \in \mathcal{S} | \epsilon(s) = \epsilon(s_o)\}| \quad (12)$$

The goal of the annealing algorithm is to find a state  $s$  with  $\epsilon(s)$  close to  $\epsilon(s_o)$ . For a very low  $t$  the process would be almost all the time in such a state after reaching equilibrium, but it would take a huge number of steps to reach that situation. The algorithm therefore performs the above process for several, in general decreasing, values of  $t$ , each time of course with a limited number of steps. This number should be large enough to approach the equilibrium characteristics. It is relatively low for high values of  $t$ , and from an equilibrium situation for a certain value of  $t$  to one at a slightly different  $t$ . The decrements in  $t$  and the number of steps per value of  $t$  are the parameters that characterize the schedule. The schedule used in this application of annealing is described in section 4.

## 2. Problem formulation

In this section the relation between the original problem, floorplan design, and the annealing algorithm is established. First the correspondence between the states of the annealing algorithm and floorplans is discussed. Then, in section 2.2, the translation of floorplan 'quality' into an  $\epsilon$  function is illustrated.

### 2.1 The state set

Each state in  $\mathcal{S}$  represents a feasible solution to the floorplan problem. It is in some sense an encoding of the floorplan configuration. This implies the existence of an algorithm to derive an associated floorplan from each state. To keep the state space from being unnecessarily large, the encoding should preferably be bijective, i.e. each state represents exactly one floorplan. Also, the encoding should be easy to handle, in the sense that generating configurations from others (i.e. implementing the moves) and evaluating configurations can be done efficiently.

Previous approaches [2] to floorplan design have shown the usefulness of point configurations as intermediate structures to carry the relevant topological as well as geometrical information. The points represent the submodules. Algorithms transforming such a point configuration into a floorplan while quite

accurately preserving these relative positions in the axes' directions have been described. The distances between the points form geometrical data that can be used for evaluating the configuration.

The task of the annealing algorithm will be the generation of such a point configuration. Point configurations, however, are not the internal representation of the states. That would lead to a huge state space even if the points were required to take a limited number of positions (for example the vertices of a grid). Instead it uses orderings implied by the point configurations. It uses one sequence of modules for each dimension. The interpretation of  $s$  is therefore a function (or array) with  $s_{axis}(i)$  being the module in position  $i$  of the sequence of the indicated axis.

## 2.2 The object function

Many deterministic optimization algorithms depend heavily on the object function. The eigenvalue methods for floorplan design are examples of such algorithms. The annealing algorithm, however, is quite flexible with respect to the object function. To keep the discussion from becoming too abstract the implementation of one, rather specific, but frequently used, quality criterion, total wiring length, will be described in this section. Also, to avoid a confusing amount of detail, the description is restricted to its simplest form, i.e. all modules are flexible, no external nets, aspect ratio 1, no weights on the nets, the same weight on the directions, etc. The generalizations, however, are mostly straight forward.

Unlike the deterministic optimization algorithms annealing requires the object function to be a good approximation of the real objective, because the result will most likely be a state with an  $\epsilon$  close to  $\epsilon_o$ , but possibly quite different from  $s_o$ . To estimate the length of the wires geometrical rather than topological information is needed. A point configuration based on the size of the submodules is therefore derived from the sequences.

$$axis(m) = \frac{\frac{1}{2}\Gamma(m) + \sum_{i < s^{-1}(m)} \Gamma(s(i))}{\sum_{m \in \mathcal{M}} \Gamma(m)} \quad (13)$$

The length of a net is estimated by half the perimeter of the smallest rectangle enclosing all the points representing submodules connected to that net. This is a lower bound for the length of the Steiner tree in the plane with rectilinear distances. This object function has to be evaluated many times during the annealing process. An efficient computation of the estimates of the net lengths for each  $s$  is therefore highly desirable. This efficiency is obtained by redoing only part of the computation and retrieving the rest from a precomputed data structure. This data structure is updated, rather than recomputed, after each move. It contains the range of the nets in both directions. So, for each net the first and the last submodule in each sequence, sharing a pin with that net, are stored. The notation

is  $f_{axis}(n)$  and  $l_{axis}(n)$ . When a move is made only the ranges of the nets that are connected to one of the involved modules have to be updated.

If the sum of the net length estimates is chosen as the object function, a good solution for one axis is also a good solution for the other axis, and if there are only four global optima the corresponding point configurations have all the points on a diagonal. To avoid such solutions, the sequences are forced to be close to mutually orthogonal which means that the correlation between the positions in the two direction must be low.

$$\rho = \frac{12 \sum_{m \in \mathcal{M}} s_x^{-1}(m) s_y^{-1}(m) - 3|\mathcal{M}|(|\mathcal{M}|+1)^2}{|\mathcal{M}|^2 - |\mathcal{M}|} \quad (14)$$

This leads to the following objectfunction:

$$\varepsilon(s) = (1 + \rho) \sum_{n \in \mathcal{N}} [x(l_x(n)) - x(f_x(n)) + y(l_y(n)) - y(f_y(n))] \quad (15)$$

### 3. The moves

The topology of the state space is given by its moves. Although the topology does not influence the equilibrium distribution, it has a considerable effect on the convergence properties of the annealing process.

#### 3.1 Selection

The effects of a good set of moves must be easy to compute, because many moves will be tried in the annealing process. For a good convergence all states must be easy to reach, that is it must be possible to reach any state from any other state with a small number of moves. Yet the difference between the values of the objectfunction of two by a move connected states has to be relatively small.

The move used in this implementation is exchanging two modules. The number of positions between those two modules plus one is called the length of a move. The maximum length of a move is controlled by a parameter  $L$ . For object functions like the one in (15) the change in  $\varepsilon$  tends to grow with the move length. Decreasing the maximum length of the moves therefore mostly reduces the maximal difference in  $\varepsilon$  of the two connected states, and thus increases the proportion of accepted moves, but it increases the diameter of the state space. For any given  $L$  all states have the same number of possible moves. If these moves are chosen with equal probability the requirements of (1) and (2) are satisfied. The equilibrium is therefore not affected by a change in  $L$  (this not true for changing  $\varepsilon$  during the annealing). It also means that the entropy of the configuration for high  $t$  can be easily computed. For all states are equally probable when  $t$  is high enough.

$$S_\infty = 2 \ln |\mathcal{M}| \quad (16)$$

### 3.2 Acceptance

There are several functions that satisfy the requirements (3 - 6) and the preceding sections do not imply any specific choice. The commonly adopted function is

$$\alpha((s, s'), t) = \min \left\{ 1, \exp \left( -\frac{\varepsilon(s') - \varepsilon(s)}{t} \right) \right\} \quad (17)$$

This function has a number of properties that can be used advantageously in controlling the schedule. Firstly, it enforces a simple relation between  $E$  and  $S$ :

$$\frac{dE}{dt} = t \frac{dS}{dt} = \frac{\sigma^2}{t^2} \quad (18)$$

where  $\sigma^2 = |\langle \varepsilon^2 \rangle - \langle \varepsilon \rangle^2|$ . This makes it possible to monitor the entropy during the annealing if a quasi-equilibrium is maintained during the whole process. Another property that is a consequence of selecting (17) is that if  $\varepsilon$  is normally distributed over the states, then this normal distribution is not only realized in equilibria for  $t \rightarrow \infty$ , but for all values of  $t$ , with a sufficient number of states with  $\varepsilon$  close to  $E$ . Moreover,  $\sigma$  is the same for all these equilibria.

The average change in the objectfunction  $\bar{\Delta}\varepsilon(l)$  can be tabulated as a function of the length of the moves. The probability of accepting an  $\varepsilon$ -increasing move is kept more or less constant during the process by controlling  $L$ :

$$L^{-1} \sum_{l=1}^L \exp \left( \frac{-\bar{\Delta}\varepsilon(l)}{t} \right) \approx \zeta \quad (19)$$

## 4. The schedule

Information concerning the values of the control parameter for which the Markov-process of section 1.2 is simulated and for how long is called the schedule. It should specify the initial value of the control parameter, the decrements in that value, and when to stop the annealing.

### 4.1 Initialization

The initial probability of accepting the move with the biggest change in  $\varepsilon$  must be reasonably high. By approximating this maximum change in the object function, an initial  $t$  can be calculated for a given probability  $\xi$ :

$$t = \frac{\max(\Delta\varepsilon)}{\ln(1/\xi)} \quad (20)$$

Usually it is not difficult to approximate  $\max(\Delta\varepsilon)$ , but, if not done empirically, the method depends on the problem. In the case of floorplan design with total

wiring length as object function, the maximum change over a move certainly is smaller than

$$2 \times \text{length}(\text{longest axis}) \times \max_{\mathcal{M}}(\#\text{pins}). \quad (21)$$

Since initial values for  $E_\infty$  and  $\sigma^2$  have to be calculated, the obtained initial  $t$  can be checked against another condition, namely  $t \gg \sigma$ . This follows from the requirement that  $E(t)$  must initially be much closer to  $E_\infty$  than the standard deviation  $\sigma$  in order to be able to reach all states easily, and the fact that for high values of  $t$ , assuming a normal distribution for the  $\varepsilon(\text{present} \triangle \text{state})$ ,  $E$  depends on  $t$  according to

$$E(t) = E_\infty - \frac{\sigma^2}{t} \quad (22)$$

with  $\sigma^2$  independent of  $t$ . This  $\sigma^2$  is determined empirically by moving freely through the state space, calculating  $\varepsilon$  for each of the states visited, and at the end setting  $E_\infty$  to  $\langle \varepsilon \rangle$  and  $\sigma^2$  to  $|\langle \varepsilon^2 \rangle - E_\infty^2|$ . If the value of  $t$  calculated with (20) does not exceed  $\sigma$  by a considerable amount,  $t$  has to be increased.

## 4.2 A stop criterion

If the decrements in  $t$  are small enough to keep the process in quasi-equilibrium, a fairly general stop criterion can be used. It derives from the observation that the improvement possible by lowering  $t$  further must be much smaller than the improvement obtained by decreasing  $t$  stepwise from its initial value. The latter equals  $E_\infty - E(t)$ . An upper bound for the improvement still possible, if  $t$  is in the interval with a positive second derivative of  $E$ , is  $t(dE/dt)$ . Using (12) this gives

$$\frac{\sigma^2}{t(E_\infty - E(t))} < \theta, \quad (23)$$

$\theta$  being a very small positive number. Outside the interval the upper bound is not valid, but the left hand side of (23) is close to 1 for high values of  $t$ , and drops slowly until it enters that interval.

The above criterion is only reliable if the process was kept in quasi-equilibrium during the annealing process. An indication for this condition is the value of  $S$ , the entropy. The value of  $S$  at high values of  $t$  is close to  $S_\infty$ , which can be calculated with (16). The decrements in  $S$  can be calculated by using the relation in (18). The value of  $S$  when the stop criterion is satisfied, should be close to  $S_o$ . If the schedule is too fast the process is likely to get trapped in a local minimum, for a while or forever. In the latter case  $S$  will stay much too high. In the former case  $S$  will drop at too low a value for  $t$ , and consequently drops quickly and, finally, below  $S_o$ . Of course, both phenomena can occur in the same process, and the  $S$  may be close to  $S_o$  when the stop criterion is satisfied, without convergence to a global minimum.

## 4.3 Control

The schedule must be controlled such that the process stays in quasi-equilibrium and yet converges fast to a global optimum. This has to be achieved by determining the decrements in  $t$ , and the number of moves per value for  $t$ . Experience learned that the best results were obtained by keeping the number of acceptances proportional to the change in the entropy. So if the number of moves per selected  $t$ -value is constant, and big enough to obtain useful information about  $E(t)$  and  $\sigma^2(t)$  the decrement in  $t$  is completely determined up to a constant factor:

$$\Delta t \propto \frac{t^3}{\sigma^2} \quad (24)$$

The factor can be chosen such that the steps do not disturb the distribution function too much. For example,

$$\forall_{s \in S} \left[ \frac{\delta(s, t)}{\delta(s, t - \Delta t)} < \gamma \right] \quad (25)$$

which for the higher values of  $t$  is satisfied if

$$|\Delta t| < \frac{t^2 \ln \gamma}{\max_S \epsilon + t \ln \gamma} \quad (26)$$

From (24) and (26) an expression for the factor can be derived. Using the maximum value of the object function is, however, over-pessimistic and leads to unnecessary slow schedules. In the program it is therefore replaced by  $E_\infty$ .

## Notes

1. C.D.Gelatt jr. et.al.: "Optimization of an organization of many discrete elements"; Patent to issue.
2. This is the notation of [3]:  $\mathcal{M}$  is the set of modules,  $\mathcal{N}$  the set of nets, and  $\mathcal{P}$  the set of pins. In this paper  $\Gamma$  simply assigns an area to a module.

## References

- [1] Kirkpatrick, S., C.D. Gelatt and M.P. Vecchi "Optimization by Simulated Annealing", Science, Vol. 220, May 1983, pp.671-680. pp. 21-34.
- [2] R.H.J.M. Otten "Automatic floorplan design", IBM Research Report, RC 9656, Yorktown Heights, N.Y., 1982, (also: VLSI Technologies in Graphics, H.Fuchs (ed.), IEEE Computer Society, 1983.), Preliminary version in: Proceedings of the 19th Design Automation Conference, Las Vegas, 1982, pp. 261-267.)
- [3] R.H.J.M. Otten, L.P.P.P. van Ginneken: "Stepwise layout refinement", Proceedings of the International Conference on Computers and Design, Port Chester, 1984.

```

module ANNEAL;
import ( $\mathcal{M}, \mathcal{P}, \mathcal{N}$ ),  $\Gamma, \Lambda, E_\infty, \overline{\Delta\epsilon_x}, \overline{\Delta\epsilon_x}, \sigma^2, t, \hat{s}$ ;
const  $\gamma, \zeta, \eta, \theta$ ;
□
begin  $s_1 := \hat{s}; k := -1$ ;
 $S_\infty := 2|\mathcal{M}| \ln(|\mathcal{M}|) + (2\pi|\mathcal{M}|)$ ;
 $S_o := \ln(\# \text{ global minima })$ ;
 $M := \frac{E_\infty + t \ln \gamma}{\sigma^2 \ln \gamma} t$ ;
repeat  $k := k + 1; E := E_\infty; S := S_\infty$ ;
 $L_x := L_y := |\mathcal{M}| - 1$ ;
repeat  $e := 0; esq := 0$ ;
for  $h := 1$  to  $|\mathcal{M}|$  do
  axis := random( $x, y$ );
   $i := \text{random}(1, \dots, |\mathcal{M}|)$ ;
  repeat  $j := \text{random}(1, \dots, |\mathcal{M}|)$ 
    until  $(i \neq j) \wedge (|s_1[i] - s_1[j]| \leq L_{axis})$ );
   $s_2 := \text{swap}(s_1[i], s_1[j], axis)$ ;
  if  $\text{random}([0, 1]) < \exp\left(-\frac{\epsilon(s_2) - \epsilon(s_1)}{t}\right)$ 
    then  $s_1 := s_2$ ;
  if  $\epsilon(s_1) < \epsilon(\hat{s})$  then  $\hat{s} := s_1$  end
  end;
   $e := e + \frac{\epsilon(s_1)}{|\mathcal{M}|}; esq := esq + \frac{\epsilon(s_1)^2}{|\mathcal{M}|}$ 
end;
 $\Delta E := e - E; E := e; \sigma^2 := |esq - E^2|$ ;
 $t := t - \frac{1}{2^k M} \frac{t^3}{\sigma^2}$ ;
 $L_{axis} := \min\left(\{|\mathcal{M}| - 1\} \cup \left\{L \mid L^{-1} \sum_{r=1}^L \exp\left(-\frac{\overline{\Delta\epsilon_{axis}}(r)}{t}\right) < \zeta\right\}\right)$ ;
 $S := S + \frac{\Delta E}{t}$ ;
until  $\frac{\sigma^2}{t(E_\infty - E(t))} < \theta$ ;
until  $\left|\frac{S - S_o}{S_\infty - S_o}\right| < \eta$ 
end;

```

# GOALIE: A SPACE-EFFICIENT SYSTEM FOR VLSI ARTWORK ANALYSIS

Thomas G. Szymanski<sup>1</sup> and Christopher J. Van Wyk<sup>2</sup>

*AT&T Bell Laboratories*

*Murray Hill, NJ 07974*

## Abstract

This paper deals with the algorithmic foundations of the GOALIE artwork analysis system. GOALIE includes programs for boolean geometric operations, connectivity analysis, transistor extraction and measurement, circuit extraction, parasitic capacitance measurement, and design rule checking. One of our major results is showing how the expected main memory requirement for all these tasks can be limited to  $O(\sqrt{n})$  where  $n$  is the number of edges in the artwork, while still running at least as fast as previously published algorithms. GOALIE can therefore handle large layouts on small computers, or even on personal workstations.

## 1. Introduction

The first step in processing a layout with GOALIE is to flatten the layout's hierarchical description (given, for example, in CIF [8]) into a set of files, each of which represents the geometric regions on one mask layer. Boolean geometric operations are then used to remove overlaps between figures within a layer, find transistor channel regions, and derive the set of "wires" comprising the circuit. *Connectivity analysis* is then performed to assign electrical net numbers to each wire (or piece thereof) and *transistor extraction* is performed to determine the terminal nets and channel dimensions of each transistor. If desired, the parasitic capacitance of each net can also be determined and design rules can be checked.

Each geometric region is represented by its edges, where an edge is described by its endpoints,  $(x_1, y_1)$  and  $(x_2, y_2)$ , with either  $x_1 < x_2$  (non-vertical edges) or  $x_1 = x_2$  and  $y_1 < y_2$  (vertical edges), along with an *orientation field* that indicates on which side of the edge the region lies. An edge may also contain additional information such as the number of the electrical net to which it belongs. A set of regions is represented by an *edge file*, containing the edges of all regions in *canonical order*, viz., lexicographical order on  $(x_1, y_1)$  pairs, with ties broken by the edges' slopes. Edge files are the basic inputs and outputs of the various operations available in the GOALIE system.

---

Authors are currently with <sup>1</sup>Avaya Labs Research and <sup>2</sup>Drew University.

## 2. Geometric Operations

GOALIE performs boolean geometric operations such as union and intersection using edge-oriented scanline algorithms as described, for example, in [6, 12]. These algorithms can be implemented to run in  $O(n \log n)$  time and  $O(\sqrt{n})$  expected space. We use the method described by us in [14] to sort the output edges into canonical order, that is, the output edges are first written into a temporary file that is then read backwards and passed through a priority queue to produce the final output file. This strategy is significantly more efficient than applying a general purpose sorting procedure and has the added advantage of using only  $O(\sqrt{n})$  expected space.

Our implementation can produce any boolean combination of two or more input edge files; moreover, GOALIE can produce several different boolean combinations at once. For example, in an NMOS process one might need to find both the intersection and the difference between diffusion and polysilicon layers to obtain the transistors and diffusion wires, respectively. GOALIE does both of these operations during a single boolean operation on the diffusion and polysilicon edge files.

## 3. Connectivity Analysis

A key algorithm used in GOALIE and recently described by us in [14] is a space efficient algorithm for connectivity analysis. Previous algorithms for this task [2, 4, 5, 7, 9, 10, 11, 13], have either kept all the edges of each net in main memory until the net is complete, or else gathered some “global topology” information in a table and used this information to renumber the edges in a separate pass. Both of these approaches take  $O(n)$  space.

For expository purposes, let us first consider the simpler problem of *region analysis*, that is, the problem of assigning the same number to each edge of the same connected region of a single set of polygons. Our method begins with a standard scanline algorithm and augments it with a “union-find” data structure [1] to maintain sets of output edges that have been discovered to belong to the same connected region in the half plane to the left of the scanline. Associated with each set of edges is a temporary region number. Whenever an input edge is discovered to be an output edge, it is placed in a new singleton set with a new temporary number. Whenever the scanline passes beyond an output edge, the output edge is written to the temporary file, tagged with the temporary number of its set. Whenever two output edges are discovered to touch, their corresponding sets are merged, a record is written to the temporary file giving the temporary numbers of the two sets involved, and one of the temporary numbers is discarded. Finally, whenever the last remaining output edge in some set is written out, a special record indicating the temporary number of the set is written to the temporary file.

This temporary file has exactly enough information to allow final region numbers to be assigned to each output edge during the backward sorting pass described in Section 2. The union-find structure, the renaming table used during the sorting pass, and the priority queue used for sorting, all require  $O(\sqrt{n})$  expected space and  $O(n \log n)$  time.

It is straightforward to extend this method to perform connectivity analysis on several mask layers. We maintain a separate scanline for each layer while keeping all output edges in a common union-find structure. Edges are tagged with the name of their layer. For each contact window in canonical order, we advance all scanlines to the abscissa of the window, and then find an edge of the region surrounding the contact on each of the layers being connected. The sets containing these edges are then merged together and a record is written to the temporary file just as before. During the sorting pass the output edges are written to the output files that correspond to their layers.

## 4. Transistor and Parasitic Capacitance Extraction

In this section we shall describe how we extract and characterize transistors and parasitic capacitance from the artwork. We first need to introduce some terminology. A *level* is an edge file representing a set of geometric regions, each of which has an associated electrical net number. A level is said to be *present* at exactly those points in the plane that lie in the interior of some region in the level. Given  $k$  levels, the *color* of a point is the set of levels present at the point. We can therefore view the  $k$  levels as partitioning the plane into regions of  $2^k$  possible colors.

We have developed a structure called a *region-oriented scanline* that makes it convenient to process the regions generated by a set of levels. The structure makes it simple to find the net of each level that is present in a region. Similarly, the lengths of edges and the areas of regions are easily obtained. This information is associated with the bottommost edge of a region and is maintained by a finite automaton traversing the scanline. The automaton also makes it simple to gain access to the information associated with all abutting regions.

Region-oriented scanlines can be used to extract transistor connectivity as follows: pass the region-oriented scanline over edge files containing diffusion and polysilicon regions (with previously assigned net numbers), transistor channels, and any relevant implants. For each transistor region, look for the net numbers on its covering polysilicon and on abutting diffusion. The record produced for a transistor includes the gate, source, and drain signals, any implant layers that are present, its area and its perimeter with diffusion. (From the last two numbers the channel length and width can be computed for rectangular transistors.)

This version of transistor identification and interconnection is simpler and more robust than the method we described earlier [14].

The parasitic capacitance of an electrical net is approximated as a function of the net's area and its perimeter abutting regions of different colors. By passing a region-oriented scanline over edge files containing the conducting layers and any relevant implants, we can find and report for each region its area, what electrical nets are carried on each layer that is present in it, and what its perimeters are with regions of other colors. The detailed description of regions output from

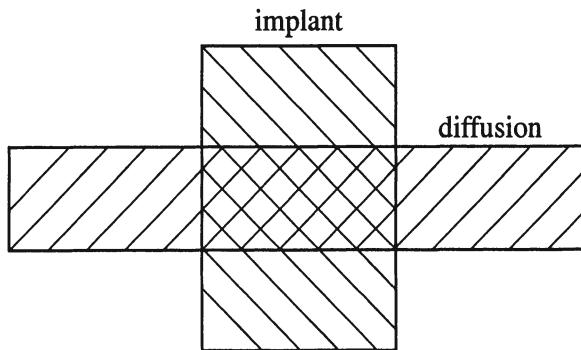


Figure 1. A diffusion runner under buried contact implant.

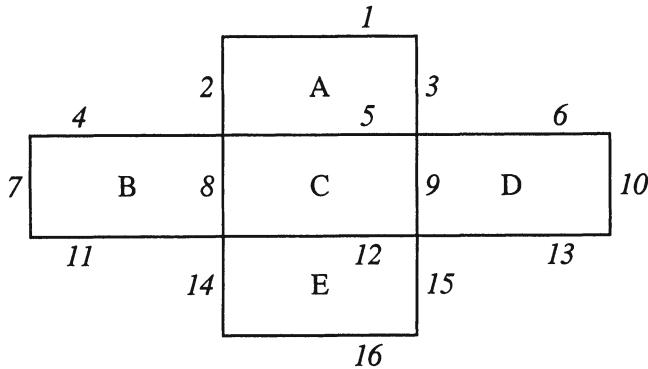


Figure 2. Areas and edges of the layout in Figure 1.

this step makes precise capacitance extraction possible. For example, consider a diffusion runner passing through a buried contact region, as shown in Figure 1. Our approach would gather the following statistics for the regions shown in Figure 2:

- ◇ Region B contains only diffusion; its perimeter with empty space is the sum of the lengths of edges 4, 7, and 11; its perimeter with both diffusion and buried contact implant is edge 8.

- ◊ Region C contains diffusion and buried contact implant; its perimeter with diffusion only is the sum of the lengths of edges 8 and 9; its perimeter with buried contact implant is the sum of the lengths of edges 5 and 12.
- ◊ Region D contains only diffusion; its perimeter with empty space is the sum of the lengths of edges 6, 10, and 13; its perimeter with both diffusion and buried contact implant is edge 9.

In calculating the capacitance of the diffusion runner, we could use a different coefficient for the areas of regions B and D than for C. We could also count B's perimeter at edge 8 differently than that of its other edges, and C's perimeter at edges 5 and 12 differently from that at 8 and 9.

Let us consider the space requirements for these operations. A transistor can be written out to disk as soon as the scanline sweeps beyond its channel region. Since the expected number of transistors that cross any scanline is  $O(\sqrt{n})$ , the working space for transistor extraction is  $O(\sqrt{n})$ . Similarly, the area and perimeter statistics for a region can be output when the region ends; the expected number of regions that cross any scanline is  $O(\sqrt{n})$ , so the expected working space for parasitic capacitance extraction is  $O(\sqrt{n})$ .

The output from parasitic capacitance extraction consists of a separate record for each region. We use caching and data compression techniques to reduce the size of this file which typically occupies the same amount of disk space as the edge files for the connectable levels of the layout. Various postprocessors are available for converting this file into whatever form is required for simulators and other programs.

## 5. Design Rule Checking

In this section we discuss the algorithm used in GOALIE for checking that the boundaries of geometric regions are separated by some minimum clearance. This algorithm is used for both clearance and enclosure checking. It supports both the Manhattan and Euclidean metrics, a property not shared by algorithms that check for clearance violations by growing regions and checking for intersections. Indeed, the algorithm even supports non-uniform metrics where the required clearance between two regions depends on the slopes of their facing edges.

For simplicity assume that the input to the algorithm consists of a set of disjoint polygonal regions represented by a single edge file with electrical net numbers attached to the edges. Without loss of generality, let the tolerance  $t$  be 1 and all edges lie within a bounding box of height  $H$  and width  $W$  whose lower left corner lies at the origin. Define *swath*  $i$  to consist of all points in the plane whose abscissa  $x$  satisfies  $i \leq x < i + 1$ .

Our approach is to check, for each endpoint  $p$  of every edge, whether another edge with a different net number lies within distance 1 of  $p$ . It suffices for any

endpoint in swath  $i$  to consider only those edges that cross swath  $i$ , or have an endpoint in swaths  $i - 1$  or  $i + 1$ . The separation checking algorithm appears in Figure 3.

```

for  $i \leftarrow 0$  step 1 until  $W$  do
    insert into  $S$  all edges whose left endpoint is in swath  $i + 1$ ;
    for each edge  $e$  with an endpoint  $p$  in swath  $i$  do
        check endpoint  $p$  of  $e$  against the edges in  $S$ ;
    endfor;
    remove from  $S$  all edges whose right end is in swath  $i - 1$ ;
endfor;

```

*Figure 3.* Clearance checking algorithm.

The structure  $S$  is a variant of the “segment tree” structure described in [3].  $S$  is a binary tree in which each node is associated with a subinterval of  $[0, H]$  and with a list of edges. All intervals are closed on the left and open on the right, and are of the form  $[2^j i, 2^j(i+1))$  for some non-negative integers  $i$  and  $j$ . If  $i$  is even, then the nodes associated with the intervals  $[2^j i, 2^j(i+1))$  and  $[2^j(i+1), 2^j(i+2))$  are the children of the node associated with  $[2^j i, 2^j(i+2))$ . For example, the interval  $[16, 32)$  is the parent of intervals  $[16, 24)$  and  $[24, 32)$ . An edge is stored in  $S$  by putting it on the list associated with the lowest node whose interval includes the ordinates of both of the endpoints of the edge. For example, the edge from  $(x_1, 18)$  to  $(x_2, 29)$  would be attached to the list on the node whose interval is  $[16, 32)$ .

To check an endpoint  $p = (x, y)$  of an edge  $e$  against the edges in  $S$ , it suffices to check only those edges on the lists of nodes whose intervals intersect the interval  $[y - 1, y + 1]$ . These nodes may be found either on or immediately adjacent to the path from the leaf containing  $y$  to the root. Some of the details are shown in Figure 4.

```

let  $l$  be the leaf of  $S$  whose interval contains  $y$ ;
while  $l$  is not the root of  $S$  do
    check  $p$  of  $e$  against all edges on the list of  $l$ ;
    check  $p$  of  $e$  against all edges on the list of  $l$ 's left neighbor;
    check  $p$  of  $e$  against all edges on the list of  $l$ 's right neighbor;
     $l \leftarrow$  parent of  $l$ ;
endwhile;

```

*Figure 4.* Algorithm for checking edge  $e$  against  $S$ .

At this point, let us consider a number of implementation issues. First of all, the nodes of  $S$  can be organized in a “heap” data structure [1] and indexing can be used instead of pointers for moving up, down, or sideways in the tree. Second, the algorithm supports a variety of metrics because all tests for clearance violations are explicitly performed on pairs of edges. If the required distance between edges is independent of their slopes, then the speed of the algorithm can be doubled by observing that any endpoint of an edge is actually the endpoint of at least two edges of the associated region. The process of checking endpoint  $p$  of edge  $e$  against  $S$  in Figure 4 need only be performed for the bottom endpoint of a left edge, the left endpoint of a top edge, the top endpoint of a right edge, and the right endpoint of a bottom edge. Third, the checks against the lists of neighbors mentioned in Figure 4 can be suppressed whenever the interval associated with that neighbor is further than 1 from  $y$ . Indeed, at the non-leaf nodes of  $S$ , this will always be the case with at least one of the neighbors.

Let us next consider the resource requirements of the algorithm. Let  $n$  be the number of edges in the file being checked. Certainly the expected number of edges cut by any line across the design is  $O(\sqrt{n})$ . Moreover, the expected number of edges cut by any swath of the plane is also  $O(\sqrt{n})$  where the constant of proportionality is a function of the tolerance  $t$ , the layout methodology, and the process technology. Thus the total number of edges held in  $S$  at any instant is  $O(\sqrt{n})$ . Finally, we argue that both  $H$  and  $W$  have an expected value that is  $O(\sqrt{n})$ . Since there are exactly  $2H - 1$  nodes in  $S$ , the total storage required by the algorithm is  $O(\sqrt{n})$ .

It should be clear that the height of  $S$  is  $O(\log n)$  and that an upper bound on the running time of the algorithm is  $O(bn \log n)$  where  $b$  is the maximum length of any edge list associated with a node of  $S$ . We claim that the expected value of  $b$  is  $O(1)$ . We shall assume at this point that the design is free of any “minimum feature size” violations. Since each leaf  $l$  of  $S$  corresponds to a  $3 \times 1$  rectangle in the plane, and all edges of  $l$  pass through this rectangle, it is clear that a leaf’s edge list can contain at most some constant number of edges without generating a small feature. Similarly, at a given instant, all the vertical edges associated with an interval  $[2^j i, 2^j(i+1))$  intersect the line  $y = 2^{j-1}(2i+1)$  along some horizontal interval of width 3. Thus the number of vertical edges in a node is also bounded by a constant. We have therefore accounted for all edges except diagonal ones. As it turns out, the number of diagonal lines in a single node can grow arbitrarily large if the geometry contains a large number of long diagonal edges. Although we have never seen this happen in practice, the problem can be fixed by simply storing long diagonal edges in a separate data structure. Further details will appear in a subsequent paper.

## 6. Implementation and Performance

GOALIE is technology-independent and handles geometry at all angles, although our implementation treats manhattan edges more efficiently than non-manhattan edges. GOALIE consists of about 8000 lines of C code and has been used in the verification of several dozen chips at Bell Laboratories.

In Figure 5 we present some typical performance data obtained on a DEC VAX 11/780. Time used is shown in hours, minutes, and seconds, while the maximum main memory used is shown in kilobytes. Because the four sample chips use four different technologies, the number of operations needed to extract the circuit varied from chip to chip. Both NMOS and CMOS technologies are represented, and the design methodologies ranged from hand-layout to automatically routed standard cells. The same chips have also been processed by GOALIE on an IBM 3081, a DEC VAX/750, and a SUN workstation. The space requirements remain about the same and the running times change by factors of about 0.1, 1.6, and 1.5 respectively.

chip	number of transistors	number nonvertical edges	percentage nonmanhattan edges	number of metal edges
1	7,666	177,407	0.4%	56,683
2	17,036	627,153	0.3%	174,251
3	28,792	726,297	3.7%	186,580
4	27,223	929,446	6.8%	287,433

chip	determine transistor list			compute parasitic capacitance		check metal spacing	
	ops	time	space	time	space	time	space
1	3	11:25	191	4:21	309	0:57	99
2	7	58:37	255	17:07	331	3:01	223
3	5	1:38:22	442	36:51	543	3:11	179
4	5	2:53:15	400	1:22:48	623	4:56	342

Figure 5. Goalie Performance Data.

The largest chip of which we are aware that GOALIE has extracted has 137,000 transistors. The extraction took under ten hours on a DEC VAX 11/780 with three megabytes of main memory and an operating system that does not page; the amount of memory actually used is not available to us.

## References

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

- [2] H. S. Baird. Fast algorithms for LSI artwork analysis. *Journal of Design Automation and Fault-Tolerant Computing*, 2(2):179–209, 1978.
- [3] J. L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, C-29(7):571–577, 1980.
- [4] A. Gupta. ACE: a circuit extractor. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 20, pages 721–725, 1983.
- [5] M. Hofmann and U. Lauther. HEX: an instruction-driven approach to feature extraction. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 20, pages 331–336, 1983.
- [6] U. Lauther. An  $O(N \log N)$  algorithm for boolean mask operations. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 18, pages 555–562, 1981.
- [7] P. Losleben and K. Thompson. Topological analysis for VLSI circuits. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 16, pages 461–473, 1979.
- [8] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.
- [9] T. Mitsuhashi, T. Chiba, M. Takashima, and K. Yoshida. An integrated mask artwork analysis system. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 17, pages 277–284, 1980.
- [10] D. Noice, J. Newkirk, and R. Mathews. A polygon package for analyzing integrated circuit designs. *VLSI Design*, 2(3):33–36, 1981.
- [11] B. T. Preas, B. W. Lindsay, and C. W. Gwyn. Automatic circuit analysis based on mask information. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 13, pages 309–317, 1976.
- [12] F. Preparata and J. Nievergelt. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25(10):739–747, 1982.
- [13] P. Swartz and H. Gummel. HCAP - a topological analysis program for IC mask artwork. In *Proceedings of the IEEE International Conference Computer Design*, pages 298–301, 1983.
- [14] T. G. Szymanski and C. J. Van Wyk. Space efficient algorithms for VLSI artwork analysis. In *Proceedings of the IEEE/ACM Design Automation Conference*, volume 20, pages 734–739, 1983.

# GORDIAN: A NEW GLOBAL OPTIMIZATION/ RECTANGLE DISSECTION METHOD FOR CELL PLACEMENT\*

Jürgen M. Kleinhans<sup>1</sup>, Georg Sigl<sup>1</sup> and Frank M. Johannes<sup>2</sup>

*Institute of Computer-Aided Design, Department of Electrical Engineering,  
Technical University of Munich, D-8000 Munich 2, West Germany*

## Abstract

A new placement method for cell-based layout styles is presented. It is composed of alternating and interacting global optimization and partitioning phases. In contrast to other methods using the divide-and-conquer paradigm, it maintains the simultaneous treatment of all cells during optimization over *all* levels of partitioning. In the global optimization phases, constrained quadratic optimization problems are solved having unique global minima. Their solutions induce the assignment of cells to regions during the partitioning phases. For general-cell circuits, a highly efficient exhaustive slicing procedure is applied to small subsets of cells. The designer may choose a configuration from a menu to meet his requirements on chip area, chip aspect ratio and wire length. Placements with high area utilization are obtained within low computation times. The method has been applied to general-cell and standard-cell circuits with up to 3000 cells and nets.

## 1. Introduction

A good placement is the major prerequisite for successful routing and effective use of chip area.

The complexity of the VLSI placement problem has forced the use of algorithms based on the divide-and-conquer paradigm. An important representative is the min-cut method based on graph-partitioning (e.g. [9]). But since min-cut algorithms are iterative improvement heuristics [8, 6], they depend on the initial partition. To get a good solution it might be necessary to select one partition computed from many randomly generated starting partitions [12].

Recently, algorithms have been studied that prefer another approach. The idea is to model the placement problem as a linear or nonlinear optimization problem. Usually no starting solution is needed and all modules (cells) are treated simultaneously. Among these approaches are methods using physical (force or electrical network) analogies [1, 17, 3, 7, 16] and eigenvector methods [13, 2, 5].

---

\*This work was partially supported by *Deutsche Forschungsgemeinschaft (DFG)* under grant An 125/5-2

<sup>1,2</sup>Authors are currently with <sup>1</sup>Infineon Technologies AG and <sup>2</sup>Technical University of Munich, Institute of Electronic Design Automation.

Some of these methods apply partitioning to recursively create smaller subproblems. However, they leave the simultaneous optimization after the initial step. In the following, a new placement method called GORDIAN is introduced. It has the unique feature to maintain simultaneity over *all* optimization steps.

In section 2, we give an outline of the procedure. The ingredients of the proposed method are presented in sections 3 through 5. Space and time complexity is discussed in section 6. Results for standard-cell and general-cell circuits are presented in section 7.

## 2. Outline of the Procedure

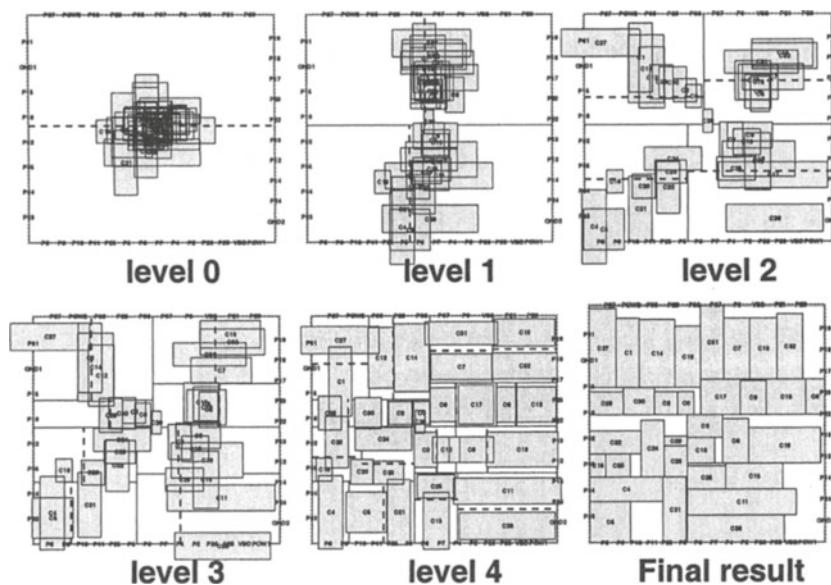
The placement procedure GORDIAN is composed of alternating global optimization and partitioning phases.

To illustrate the idea, suppose that all modules have been placed and a binary slicing tree has been created whose leaf regions contain  $m$  modules. A region represents a rectangular area of the chip together with the set of modules placed in this area. The position of each module can be described in terms of the coordinates of the leaf region containing it. The center of a region  $\rho$  is denoted by  $c_\rho = (x_\rho, y_\rho)$ . On the  $\ell^{th}$  level of partitioning there are  $q \leq 2^\ell$  regions. The regions of level  $\ell$  are the sons of  $q/2$  father regions on level  $\ell - 1$  each having a center  $c_\rho = \frac{f_{\rho'} \cdot c_{\rho'} + f_{\rho''} \cdot c_{\rho''}}{f_\rho + f_{\rho''}}$ , where  $f_\rho$  is the area of region  $\rho$ , and  $\rho'$  ( $\rho''$ ) is the left (right) son of  $\rho$ . Thus, the center of the root region of the slicing tree can be determined recursively from its leaf regions.

To create a placement from scratch, the centers of the regions have to be specified in reverse order from the root to the leaves together with the centroids<sup>1</sup> of their modules. For the root region, *one* constraint is imposed on *all* modules. A global optimization is performed with this constraint forcing the centroid of the modules to the chip's center. Then, the set of modules is bipartitioned and two subregions are created by dissecting the root rectangle. Their centers impose two new constraints on the modules replacing the single constraint of the root region. Then, the next global optimization is carried out. These global optimization and partitioning phases are repeated until each module is assigned to its own region, step by step refining the placement of the modules (see fig. 1).

---

<sup>1</sup>The centroid of a set of modules is the area-weighted mean of their center coordinates



*Figure 1.* Stepwise placement refinement.

More formally, the procedure can be stated as follows:

```

Procedure GORDIAN
level  $\ell := 0$ ; /* the root region */
while (level  $\ell$  contains a region  $\rho$  to be partitioned)
    globally optimize level  $\ell$ ;
    for (each region  $\rho$  to be partitioned)
        partition region  $\rho$  into subregions  $\rho'$  and  $\rho''$ ;
        generate new constraints for  $\rho'$  and  $\rho''$ ;
    endfor
     $\ell := \ell + 1$ ;
endwhile

```

### 3. Global Optimization

In each global optimization phase, a constrained quadratic optimization problem (CQOP) is derived from the circuit topology (the netlist) and the geometry of the rectangle dissection on the respective level. The solution of the CQOP is a global placement of all modules that induces the assignment of module subsets to subregions to be created in the next partitioning phase.

Let us first introduce some definitions: The topology is described by the bi-

nary relation  $T \subseteq \mathcal{N} \times \mathcal{M}$ , where  $\mathcal{N}$  and  $\mathcal{M}$  are the index sets of the nets and the modules, resp., with  $N = |\mathcal{N}|$  and  $M = |\mathcal{M}|$ . A connection of net  $v$  to module  $\mu$  is represented by  $(v, \mu) \in T$ . By  $\mathbf{T}$  we denote the topology matrix  $\mathbf{T}_{\langle N \times M \rangle} = [t_{v\mu}]$ ,  $t_{v\mu} = \begin{cases} 1, & \text{if } (v, \mu) \in T \\ 0, & \text{otherwise} \end{cases}$ . The coordinates of the modules and the nets are  $(x_\mu, y_\mu)$  and  $(x_v, y_v)$ , respectively.

The objective function  $\phi$  of the CQOP is the weighted sum of the squared distances between modules and nets:

$$\phi(x, y) = \frac{1}{2} \sum_{v \in \mathcal{N}} \sum_{\mu \in \mathcal{M}} [(x_\mu + \xi_{v\mu} - x_v)^2 + (y_\mu + \eta_{v\mu} - y_v)^2] \cdot t_{v\mu} \cdot w_v, \quad (1)$$

where  $(\xi_{v\mu}, \eta_{v\mu})$  are the coordinates of a pin relative to the center of its module  $\mu$ , and  $w_v$  is the weight of net  $v$  specified by the designer. Since

$$\phi(x, y) = \phi(x) + \phi(y), \quad (2)$$

we consider  $\phi(x)$  synonymous for either term of (2). Writing (1) in matrix form, we get

$$\phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}' \mathbf{C} \mathbf{x} + \mathbf{d}' \mathbf{x} + \text{const.} \quad (3)$$

with the coordinate vector  $\mathbf{x}' = [\mathbf{x}'_N, \mathbf{x}'_m]$  of the  $N$  nets and the  $m$  movable modules, and  $\mathbf{C}$  the positive definite connectivity matrix. The vector  $\mathbf{d}$  originates from the contributions of the  $M - m$  fixed modules (the pad cells) and the relative pin coordinates.

The centers of the  $q$  regions on the  $\ell^{th}$  level of partitioning form the constraints on the  $m$  movable modules

$$\mathbf{A}^{(\ell)} \mathbf{x}_m = \mathbf{b}^{(\ell)}, \quad (4)$$

with the vector  $\mathbf{b}^{(\ell)}$  of the center coordinates of the regions. The entries  $a_{\rho\mu}$  of  $\mathbf{A}^{(\ell)}_{\langle q \times m \rangle}$  describe which module (occupying  $f_\mu$  units of area) belongs to which region  $\rho$ :

$$a_{\rho\mu} = \begin{cases} f_\mu / \sum_\mu f_\mu & , \text{ if } \mu \in \mathcal{M}_\rho \\ 0 & , \text{ otherwise.} \end{cases} \quad (5)$$

Putting together the topological (3) and the geometric (4) demands, the problem to be solved is

$$\min_{\mathbf{x}} \{\phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}' \mathbf{C} \mathbf{x} + \mathbf{d}' \mathbf{x} \mid \mathbf{A}^{(\ell)} \mathbf{x}_m = \mathbf{b}^{(\ell)}\}. \quad (6)$$

Since  $\phi(\mathbf{x})$  is a convex function ( $\mathbf{C}$  is positive definite) and the linear equality constraints (4) are convex too, the problem (6) has the unique global minimum  $\phi(\mathbf{x}^*)$ .

The advantage of simultaneous optimization over the optimization in local module subsets is motivated by the fact that modules influence each other, even if

they belong to distant regions. Thus, the placement of the whole circuit is optimized in its entirety.

To solve problem (6), various methods can be applied like a fixed point iteration scheme similar to the one described in [7] or other optimization algorithms [11]. In this context, iterative solution methods outperform direct solvers, since the system matrix  $\mathbf{C}$  has to be set up only once for all CQOP's and sparsity is fully exploited. But most important, the result of the CQOP of level  $\ell$  is a good starting solution for the CQOP of level  $\ell + 1$ . Furthermore, the increasing number  $q$  of constraints in (4) more and more determines the result of the optimization. Therefore, the number of iterations needed to solve (6) decreases rapidly with higher levels  $\ell$ .

## 4. Partitioning

After each global optimization a new partitioning phase is started. For each region  $\rho$  with  $|\mathcal{M}_\rho| \geq 2$ ,  $\mathcal{M}_\rho$  is sorted according to the  $x$ - ( $y$ -) coordinates of its modules, if the region will be cut vertically (horizontally). Then,  $\mathcal{M}_\rho$  is divided into  $\mathcal{M}'_\rho$  and  $\mathcal{M}''_\rho$  such that the sums of the module areas of both subsets are approximately the same. The rectangular area of region  $\rho$  is dissected correspondingly.

For the column-oriented layout style of standard-cells, the chip area is first dissected by vertical cuts until all modules are assigned to columns. Then, the placement is refined within the columns by making horizontal cuts.

For the general-cell layout style with rectangular modules of different widths and heights, horizontal and vertical cuts alternate from one level to the next, until  $|\mathcal{M}_\rho| \leq k$ , where  $k \geq 2$  is a predefined constant. For these regions an exhaustive slicing optimization is performed.

## 5. Exhaustive Slicing Optimization

The area utilization of slicing structures can be optimized efficiently using the concept of *shape functions* [14, 10, 18].

For general-cell circuits, an exhaustive slicing optimization procedure (ESO) is proposed, combining enumeration of slicing subtrees with global optimization and evaluation of shape functions.

For small subsets of modules, it is not too expensive to enumerate all possible subtrees. For  $k = |\mathcal{M}_\rho| = 2$  modules, there are  $t(2) = 2$  subtrees;  $t(3) = 6$ ,  $t(4) = 22, \dots$  However, considering the  $k!$  possible permutations of modules for each of the  $t(k)$  subtrees,  $t(k) \cdot k!$  placements have to be evaluated for each subset  $\mathcal{M}_\rho$ . Therefore,  $k = 4$  represents the practicable limit in most implementations of this approach.

By exploiting the result of the global optimization of section 3 the factor  $k!$  of computation time can be saved. This allows us to apply ESO to larger subsets

of modules (approximately 7). The allocation of modules to regions is derived from the coordinates of the modules determined in the last global optimization phase. This results in short wire length, since it preserves the relative positions of the global placement.

Combining ESO with the evaluation of shape functions means to calculate the combined shape function of all  $t(m_\rho)$  subtrees for each region with  $m_\rho = |\mathcal{M}_\rho| \leq k$  modules and recursively that of the root. Thus, all possible shapes of all exhaustively optimized regions simultaneously contribute to the final result. This allows the designer to choose from a menu the best shape of the whole chip with the shortest estimated wire length.

## 6. Complexity of the Method

**Space complexity:** The connectivity matrix  $C$  for both the  $x$ - and  $y$ -coordinates is stored in a list structure with  $O(m + N + P)$  memory space, where  $m, N, P$  are the numbers of movable modules, nets and pins, resp. For larger circuits,  $P$  and  $N$  grow proportionally to  $m$ . The constraint matrix  $A$  can be stored in a vector of length  $m$  (cf. (5)). The slicing tree has  $2m - 1$  nodes. Thus, the space complexity of the CQOP and the partitioning phases is  $O(m)$ .

**Time complexity:** Each iteration step in the global optimization takes time proportional to  $(N + m + P)$ , which is  $O(m)$ . The number of iterations to solve the CQOP can be limited by a constant much smaller than  $m$ . The partitioning of  $q$  regions takes time proportional to  $q \cdot \frac{m}{q} \cdot \log \frac{m}{q}$  which is  $O(m \cdot \log m)$ . A balanced slicing tree has  $\log m$  levels. Thus, the total time complexity of global optimization and partitioning is  $O(m \cdot \log^2 m)$ .

The space and time complexity of ESO depends on the number of different module shapes and therefore varies from one circuit to the other.

## 7. Experimental Results

In this section, we present experimental placement results in terms of estimated wire length, chip area and computation time. The cpu times were obtained on a DEC MicroVAX II computer running ULTRIX-32. GORDIAN is written in C.

Characteristics of benchmark circuits from [15] are listed in table 1. In tables 2 and 3, three methods are compared for the standard-cell circuits *Primary1* and *Primary2*: (a) min-cut with terminal propagation [6, 4], where the best result obtained from several randomly created starting solutions is shown, (b) RT: the relative placement / transportation method from [7], and (c) GORDIAN. The modules were placed in 17 and 26 columns, respectively, with estimated inter-column channel widths of  $220\mu\text{m}$  and  $270\mu\text{m}$ . Wire length is measured as half perimeters and minimum spanning trees with squared Euclidian metric ( $E^2$ -MST), summed over all nets. The  $E^2$ -MST measure is highly correlated

with our CQOP objective (1).

circuit	layout style	modules	nets	pins
<i>Primary1</i>	standard-cell	752	904	2941
<i>Primary2</i>	standard-cell	2907	3029	11226
<i>AMI33</i>	general-cell	33	123	522

Table 1. Characteristics of the benchmarks.

algorithm	half perimeter [m]	E <sup>2</sup> -MST ( $\times 10^9$ )
min-cut	1.739	2.263
RT	2.177	3.626
GORDIAN	1.503	1.706

Table 2. Standard-cell benchmark *Primary1*.

algorithm	half perimeter [m]	E <sup>2</sup> -MST ( $\times 10^9$ )
min-cut	9.823	2.104
RT	8.685	1.739
GORDIAN	8.142	1.502

Table 3. Standard-cell benchmark *Primary2*.

ESO parameter $k$	wire length [mm]	chip area [ $mm^2$ ]	cpu time [s]
1	76.37	1.787	38.0
2	73.40	1.594	43.6
3	72.88	1.499	40.6
4	72.40	1.475	53.7
4 <sup>a</sup>	82.22	1.449	104.1
5	72.40	1.475	53.7
6	76.26	1.449	57.9
7	76.26	1.449	58.1

Table 4. General-cell benchmark *AMI 33*.

The general-cell circuit *AMI33* was placed with varying values of the ESO parameter  $k$  (cf. section 5) without extra wiring space. The results with minimal area from the GORDIAN menu are presented in table 4 (for  $k = 5$ , see also figure 1). The table shows the effect of exhaustive slicing on both chip area and

wire length scored as half perimeters, summed over all nets. The values of entry 4<sup>a</sup> were obtained by permuting the modules for subsets with  $|\mathcal{M}_p| \leq 4$  rather than exploiting the global optimization result. The ESO procedure of section 5 clearly performs superior when considering all factors.

Figure 2 shows the computation time of GORDIAN versus circuit size for several circuits. Evidently, the experiments conform to the expected time behaviour, allowing the solution of much larger placement problems even on a workstation.

## 8. Conclusions

An important aim of current research on layout techniques is the development of placement strategies that emphasize a global view, since this is difficult to obtain for a human designer particularly when designing large chips.

GORDIAN, a new placement method preserving the simultaneous treatment of all modules (cells) through all phases of the algorithm has been presented. For general-cell circuits, it provides the designer with a menu of near-optimal placements in terms of the probably conflicting parameters chip area, aspect ratio and estimated wire length, from which he can make his choice. Excellent results have been obtained with respect to placement quality and computation time.

First experiments with sea-of-gates circuits have shown that GORDIAN is well suited for this layout style, too.

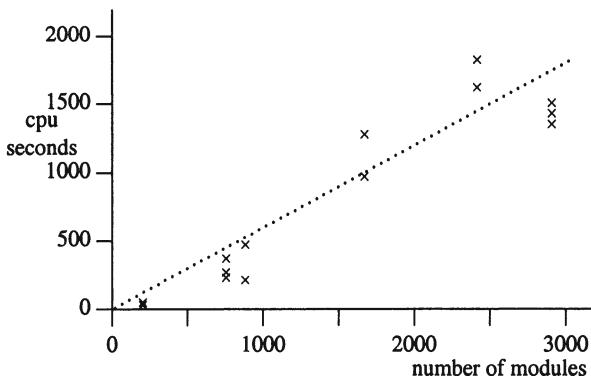


Figure 2. GORDIAN: computation time vs. circuit size.

## References

- [1] K. J. Antreich, F. M. Johannes and F. H. Kirsch. (1982). *A new approach for solving the placement problem using force models*. IEEE Proc. ISCAS, 481–486.
- [2] J. P. Blanks. (1985). *Near-optimal placement using a quadratic objective function*. Proc. 22nd DAC, 609–615.

- [3] C.-K. Cheng and E. S. Kuh. (1984). *Module placement based on resistive network optimization*. IEEE Trans. on CAD, 218–225.
- [4] A. E. Dunlop and B. W. Kernighan. (1985). *A procedure for placement of standard-cell VLSI circuits*. IEEE Trans. on CAD, 92–98.
- [5] J. Frankle and R. M. Karp. (1986). *Circuit placements and cost bounds by eigenvector decomposition*. IEEE Proc. ICCAD, 414–417.
- [6] C. M. Fiduccia and R. M. Mattheyses. (1982). *A linear-time heuristic for improving network partitions*. Proc. 19th DAC, 175–181.
- [7] K. M. Just, J. M. Kleinhans, and F. M. Johannes. (1986) *On the relative placement and the transportation problem for standard-cell layout*. Proc. 23rd DAC, 308–313.
- [8] B. W. Kernighan and S. Lin. (1970). *An efficient heuristic procedure for partitioning graphs*. Bell Systems Technical Journal, 291–307.
- [9] U. Lauther. (1979). *A min-cut placement algorithm for general cell assemblies based on a graph representation*. Proc. 16th DAC, 1–10.
- [10] D. P. LaPotin and S. W. Director. (1986). *Mason: A global floorplanning approach for VLSI design*. IEEE Trans. on CAD, 477–489.
- [11] D. G. Luenberger. (1973). *Introduction to Linear and Nonlinear Programming*. Addison-Wesley.
- [12] T.-K. Ng, J. Oldfield, and V. Pitchumani. (1987). *Improvements of a mincut partition algorithm*. IEEE Proc. ICCAD, 470–473.
- [13] R.H.J.M. Otten. (1982). *Eigensolutions in top-down layout design*. IEEE Proc. ISCAS, 1017–1020.
- [14] R.H.J.M. Otten. (1983). *Efficient floorplan optimization*. Int. Conf. on Computers and Design, 499–501.
- [15] B. Preas and K. Roberts. (1987). Physical Design Workshop, Hilton Head, South Carolina.
- [16] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu. (1988). *PROUD: A fast sea-of-gates placement algorithm*. Proc. 25th DAC, 318–323.
- [17] G. J. Wipfler, M. Wiesel, and D. A. Mlynski. (1982). *A combined force and cut algorithm for hierarchical VLSI layout*. Proc. 19th DAC, 671–676.
- [18] G. Zimmermann. (1988). *A new area and shape function estimation technique for VLSI layouts*. Proc. 25th DAC, 60–65.

# EXACT ZERO SKEW

Ren-Song Tsay<sup>1</sup>

*IBM, T. J. Watson Research Center*

## Abstract

An exact zero skew clock routing algorithm using Elmore delay model is presented. Recursively in a bottom-up fashion, two zero-skewed subtrees are merged into a new tree with zero skew. The algorithm can be applied to single-staged clock trees, multi-staged clock trees, and multi-chip system clock trees. It is ideal for hierarchical methods of constructing large systems. All subsystems can be constructed in parallel and independently, then interconnected with exact zero skew method.

## 1. Introduction

We propose in this paper an exact zero skew clock routing algorithm for timing performance optimization of synchronous digital systems. Clock skew is defined as the maximum difference of the delays from the clock source to the clock pins on latches. Optimization of the clock skew can dramatically reduce system's cycle time, and hence the timing performance. In contrast, improper clock skew may sometimes cause clock hazard and system malfunction [2]. The following equation summarizes the relationship of the clock period  $P$ , clock skew  $s$ , worst-case data path delay  $d_{max}$  and other offset constant  $P_o$ , for the condition of proper timing.

$$P = s + d_{max} + P_o$$

Note that  $P_o$  is a constant that includes data set up time, latch active time, and other possible offset factors such as timing safe margins. The latch active time is the lag time for the data to be latched in after the latch is triggered by a clock signal.

It is clear from the equation that in order to reduce the cycle time,  $P$ , it is necessary to minimize the skew  $s$ , besides the minimization of the worst-case data delay  $d_{max}$  on the combinational logics. As interconnection delay is becoming more dominating and design size is becoming larger, the clock skew is also becoming more significant in terms of performance optimization.

Many heuristics have been proposed in the past for clock routing. H-tree structures are most widely used, especially in systolic array designs. A generalization of H-tree that hierarchically divides at median and connects the mean

---

<sup>1</sup>Author is currently an independent consultant.

points is proposed in [3]. A further improvement is done by bottom-up pair wise connections, which construct a perfect length balanced tree [4]. However, all these heuristics focus only on wire length balancing, rather than the real objective as balancing clock delay. In contrast, what we propose is an exact algorithm that balances the clock delays and takes into account uneven loading and buffering effects.

The outline of this paper is as follows. We first study how to compute signal delays efficiently on an RC tree. An RC tree is a connected, acyclic, undirected graph with each branch associated with a resistance value and each node associated with a capacitance value.

Next, we discuss how a clock tree is modeled as an RC tree for delay analyses. In general, clock trees are classified into two types. The first type is a *single-staged* clock tree that all clock pins are driven directly from a clock source. In order to reduce phase delays (the maximum delay from the clock source to a clock pin) and to supply sufficient driving currents, usually several level of buffers are added to create a multi-staged clock tree. Thus the second type is called *multi-staged* clock trees that the clock pins are driven from intermediate buffers, and the buffers are driven by either other buffers or the clock source. A multi-chip system clock tree is basically a multi-staged clock tree except that the clock pins are scattered on many chips (or cards).

Then the zero skew algorithm is presented. Based on a lumped delay model and the delay computation method, we found that any two zero-skewed subtrees can be merged into a tree with zero skew by tapping the connection to a specific location of each subtree. Basically, it is a recursive bottom-up algorithm.

Finally, we present experimental results of the zero skew algorithm, and comparisons with a wire length balancing heuristics [4].

## 2. Linear Time Hierarchical Delay Computation

We adopt the commonly used Elmore delay model [5] to calculate the signal traveling time from a clock source to each clock pin. We modify the hierarchical method proposed in [5] and have a hierarchical method for computing delays in a bottom-up fashion, which is the key to our zero skew algorithm.

We first define some terms. Let  $T$  represent an RC tree with every node associated with an index. We assume the index of the root is always 0. A *predecessor* of node  $i$  is a node resides on the unique path between the root and node  $i$ , but excluding node  $i$  itself. An immediate predecessor of node  $i$  is a predecessor of node  $i$  with no other nodes between them. Similarly, *successors* of node  $i$  is the set of nodes with node  $i$  as one of their predecessors. An immediate *successor* of node  $i$  is a successor of node  $i$  with no other nodes in between. The root is a node with no predecessor. Leaf nodes are nodes with no successors. A subtree  $T_i$  is defined as a subtree of  $T$  formed by the node  $i$  and its successors. Since  $T$  is

a tree, there is only one unique edge between a node and its predecessor. So we simply define branch  $i$  as the edge between node  $i$  and its immediate predecessor.

Let  $c_i$  be the node capacitance of node  $i$  and  $r_i$  be the resistance of branch  $i$ . To simplify discussion, we set  $r_i = 0$  for root node  $i$ . We define  $IS(i)$  as the set of all immediate successors of node  $i$ . Then the total subtree capacitance  $C_i$  of  $T_i$  is defined recursively as

$$C_i = c_i + \sum_{k \in IS(i)} C_k \quad (1)$$

The above equation suggests that the subtree capacitance can be computed in a depth first search manner. The capacitance of the subtree rooted from a node can be computed from its own node capacitance and the summation of the subtree capacitance of its immediate successors. Hence a recursive bottom-up algorithm can be used to compute the subtree capacitance of each node.

To calculate the delay, we first define  $N$  as the collection of all nodes on tree and  $N(i, j)$  as the collection of nodes on the path between node  $i$  and node  $j$ , excluding node  $i$  but including node  $j$ . The delay to a leaf node  $i$  can be calculated by the following formula

$$t_{0i} = \sum_{n \in N(0, i)} r_n C_n$$

As a generalization, we can compute the “delay time” between any two nodes  $i$  and  $j$  by the following formula, assuming  $i$  is a predecessor of  $j$ .

$$t_{ij} = \sum_{n \in N(i, j)} r_n C_n$$

It can be shown easily that if  $i$  is an intermediate node between node  $k$  and node  $j$ , then

$$t_{kj} = t_{ki} + t_{ij} \quad (2)$$

In case that node  $k$  is the root (i.e.  $k = 0$ ) and node  $i$  is the immediate predecessor of node  $j$ , then we have

$$t_{0j} = t_{0i} + r_j C_j \quad (3)$$

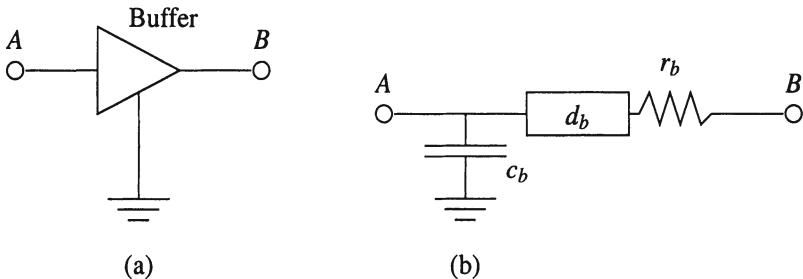
since there is only one edge between nodes  $i$  and  $j$ , and  $t_{ij} = r_j C_j$ . This equation suggests that we can easily calculate the delay from the root to any leaf node in one depth first search. The delay time to each node can be derived from its immediate predecessor, the branch resistance and the subtree capacitance. Recursively, in a top-down fashion we compute the delay time to each node.

Since the time complexity of the depth first search algorithm is linear in number of edges [1] and the number of edges is the number of nodes minus one for a tree, we easily have the following theorem.

**Theorem 1** *The delay time from the root to each node on an RC tree can be computed in linear time.*

## Generalization to Buffered RC Trees

To handle multi-staged clock trees (or buffered clock trees), we generalize the previous delay computation method for a *buffered RC tree*. Before we define what is a buffered RC tree, we first discuss the equivalent circuit model of a clock buffer as shown in Fig. 1b. We specifically designate the input node of



- $d_b$ : buffer internal delay
- $c_b$ : buffer input capacitance
- $r_b$ : buffer output driving capacitance
- A: buffer input node
- B: buffer output node

Figure 1. (a) A clock buffer. (b) An equivalent model.

a buffer as a *buffer input node*, which is important for delay calculation. The box in Fig. 1 represents a delay element with  $d_b$  as the buffer internal delay and is connected to the buffer input node on one end and the buffer output-driving resistor  $r_b$  on the other end. The buffer input capacitor  $c_b$  is on the buffer input node, and the buffer output-driving resistor  $r_b$  is connected to the delay element and buffer output node. One function of buffers is to supply enough currents for driving latches. The other function of buffers is for creating stages such that the subtree capacitance of the buffer output node will not be *carried over*, i.e. the equivalent total subtree capacitance as seen at the buffer input node is only  $c_b$ . Usually, the buffer driving resistance and input capacitance are designed to be small values. This is why buffering usually reduces delay time.

To account for the buffering effects, we define a buffered RC tree just like a normal RC tree except that each branch  $i$  is now also associated with a branch delay  $d_i$  besides the branch resistance  $r_i$ . The branch delay is always equal to zero except the case that it stands for a buffer. The basic delay calculation presented previously is modified as the following for buffered RC trees.

The calculation of the equivalent subtree capacitance at node  $i$  is now depending on whether node  $i$  is a buffer input node or not. Eq. (1) has to be modified

as the following for computing the subtree capacitance of a buffered RC tree.

$$C_i = \begin{cases} c_i & \text{if node } i \text{ is a buffer input node} \\ c_i + \sum_{k \in IS(i)} C_k & \text{otherwise} \end{cases}$$

We also extend the delay computation for a node  $i$  and its successor  $j$  as the following equation in order to accommodate the now branch delay situation, i.e.

$$t_{ij} = \sum_{n \in N(i,j)} (r_n C_n + d_n)$$

Thus Eq. (3) is modified to be

$$t_{0j} = t_{0i} + r_j C_j + d_j$$

for delay calculation of a buffered RC tree.

### 3. Delay Computations of Clock trees

We shall discuss in this section how to model a clock tree as a buffered RC tree so that we can perform delay computation efficiently. Each clock tree realization is consisted of wiring segments, clock pins, and clock buffers. Hence it is necessary to know the equivalent RC model of each component.

#### Equivalent $\pi$ -model for a Distributed RC line

Distributed RC lines are more accurate for characterizing circuit performance of wiring segments. A distributed RC line is usually represented as the symbol shown in Fig. 2a. Either a  $\pi$ -model (Fig. 2b) or a T-model (Fig. 2c) is used to represent the equivalent circuit of the distributed RC line.

Throughout this paper, we will use the equivalent  $\pi$ -model for analysis. An input node, an output node and a branch between both nodes are used to represent the equivalent  $\pi$ -model of a wire segment. Let  $R$  be the total wire resistance and  $C$  the total wire capacitance. Then the equivalent input and output node capacitances are all equal to  $C/2$ , and the equivalent branch resistance is  $R$ .

#### Equivalent Buffered RC Tree of a Clock Tree

We use a generic example as shown in Fig. 3a. to illustrate how to construct an equivalent buffered RC tree from a multi-staged clock tree. For this particular example, we assume a clock source is driving a buffer through wire 1, and the buffer is connected to the clock pin on a latch through wire 2. The driving resistance of the clock source is assumed to be  $r_s$ . Both wire segments 1 and 2 are represented by equivalent  $\pi$ -models as discussed earlier. The buffer is transformed to an equivalent circuit with buffer input capacitance  $c_b$ , buffer delay  $d_b$  and buffer output driving resistance  $r_b$ . The end clock pin of the latch

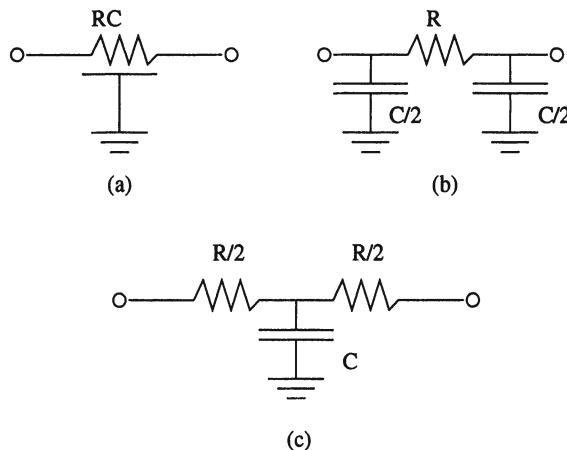


Figure 2. (a) A distributed RC line. (b) The equivalent  $\pi$ -model. (c) The equivalent  $T$ -model.

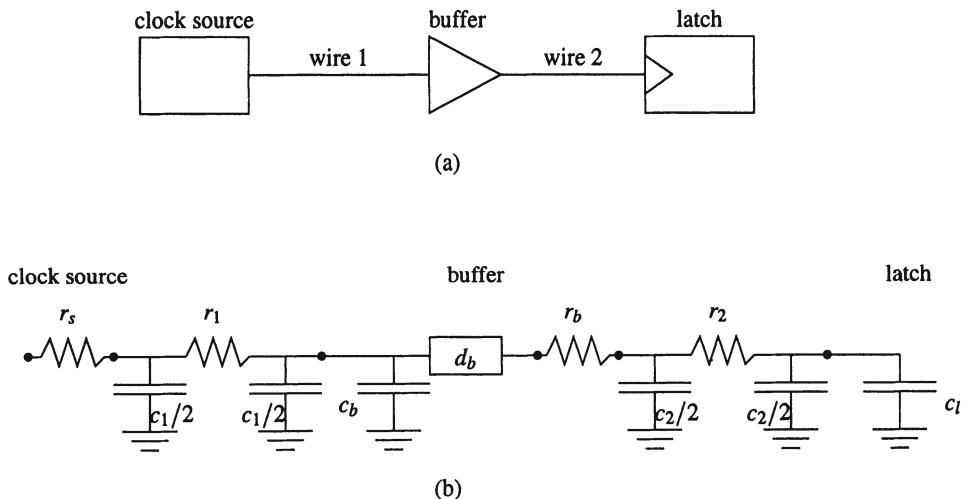


Figure 3. (a) A generic multi-staged clock tree. (b) The equivalent buffered RC tree.

is associated with a loading capacitance  $c_l$ . The equivalent buffered RC tree is as shown in Fig 3b.

## Lumped Delay Model

To make the presentation of the zero skew algorithm easier, we shall introduce a *lumped delay model* of a subtree. Recall Eq. (2)  $t_{kj} = t_{ki} + t_{ij}$ . Suppose  $i$  is an immediate successor of  $k$ , and  $j$  is a leaf node. Then

$$t_{kj} = d_i + r_i C_i + t_{ij} \quad (4)$$

Consider node  $i$  as the root of the subtree  $T_i$ . To compute the delay time one level up, from node  $k$  to node  $j$ , we need only to know the branch resistance  $r_i$ , the branch delay  $d_i$ , the subtree capacitance  $C_i$  and the delay time  $t_{ij}$  from the root of  $T_i$  to the node  $j$  according to Eq. (4).

Thus we propose an equivalent lumped delay model of the subtree  $T_i$  for simplifying the delay computation. In the equivalent circuit, the subtree  $T_i$  is replaced by an input capacitance  $C_i$  and a branch delay  $t_{ij}$  from input node  $i$  to leaf node  $j$ . We will use this lumped delay model for developing the algorithm in the next section.

#### 4. Zero Skew Algorithm

The zero skew algorithm is a recursive bottom-up process. We describe only one recursive step. Repeat the process in a bottom-up fashion will construct a complete zero skew clock tree.

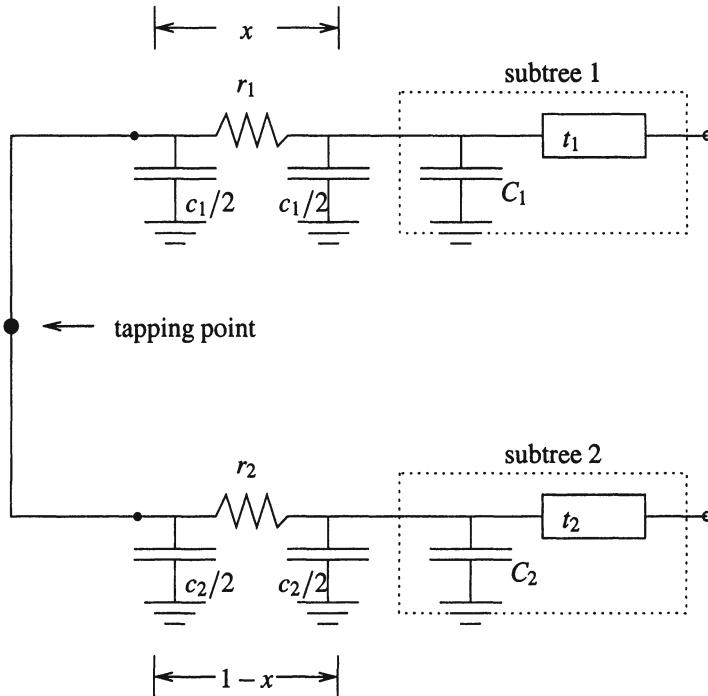


Figure 4. Zero-Skew-Merge of two subtrees.

We assume every subtree has achieved zero skew, which means the signal delays from the root of the subtree to its leaf nodes are equal. This is obvious if the subtree contains only one leaf node, and it serves as our starting point of the algorithm.

To interconnect two zero-skewed subtrees with a wire and ensure zero skew of the merged tree, the problem to be solved is the decision of where on the wire will be the new root of the merged tree, such that the delay time from this new root to all leaf nodes are equal, i.e. zero skew. We will call this new root point on the wire as a *tapping* point, and this process as the zero-skew-merge process.

Let us discuss the example shown in Fig. 4 with subtree 1 and subtree 2. First, assume the lumped delay model of each subtree is as shown in Fig. 4. The tapping point separate the interconnection wire of the two subtrees into two halves (which may not be equal). Each half-wire segment is represented as a  $\pi$ -model as shown. To ensure the delay from the tapping point to leaf nodes of both subtrees being equal, it requires that

$$r_1(c_1/2 + C_1) + t_1 = r_2(c_2/2 + C_2) + t_2 \quad (5)$$

according to Eq. (4). Note that  $r_1$ , and  $c_1$  are for wire segment 2. There are no branch delays.

We assume that the total wire length of this interconnection wire segment is  $l$ . The wire length from the tapping point to the root of subtree 1 is  $x \times l$  and hence the wire length from the tapping point to the root of subtree 2 will be  $(1 - x) \times l$ .

Let  $\alpha$  be the resistance per unit length of wire and  $\beta$  be the capacitance per unit length of wire. Then we have  $r = \alpha l$ ,  $r_1 = \alpha xl$ ,  $r_2 = \alpha(1 - x)l$ . Also,  $c = \beta l$ ,  $c_1 = \beta xl$ ,  $c_2 = \beta(1 - x)l$ .

Hence, after solving Eq. (5), we find that the zero skew condition requires that

$$x = \frac{(t_2 - t_1) + \alpha l(C_2 + \beta l/2)}{\alpha l(\beta l + C_1 + C_2)}$$

If  $0 \leq x \leq 1$ , the tapping point is somewhere along the segment interconnecting the two subtrees and is legal. In case that  $x < 0$  or  $x > 1$ , it indicates the two subtrees are out of balance. The interconnection wire has to be elongated. For simplicity, we discuss only the case that  $x < 0$ . For this case, the tapping point has to be exactly on the root of subtree 1 in order to minimize total interconnection length. Assume the elongated wire length is  $l'$ . The distributed resistance value is  $\alpha l'$  and the distributed capacitance value is  $\beta l'$ . To determine a minimum elongated wire length  $l'$ , it requires

$$t_1 = t_2 + \alpha l'(C_2 + \beta l'/2)$$

or

$$l' = \frac{[\sqrt{(\alpha C_2)^2 + 2\alpha\beta(t_1 - t_2)}] - \alpha C_2}{\alpha\beta}$$

Similar results can be obtained for the case  $x > 1$ . It is worthwhile noting that the uneven loading effect is naturally taken care of by this approach.

Wire “snaking” is a common practice for wire elongation. Since it is the nature of a clock-wiring algorithm to *balance* two subtrees, the *snaking* should not occur often.

In case that two subtrees are out of balance and the elongation severely affect the wirability then addition of buffers, delay lines, or capacitive terminators should be considered based on the same balancing principle. For instance the capacitance value, say  $C_t$ , of a capacitive terminator to be attached on the root of subtree 2 for the case  $x < 0$ , can be determined by solving the equation  $t_1 = t_2 + \alpha l(C_2 + \beta l/2 + C_t)$ , or we have  $C_t = (t_1 - t_2)/(\alpha l) - (C_2 + \beta l/2)$ .

Before presenting the algorithm formally, we define a few more related terms. The number of stages of a clock tree is defined as the maximum number of clock buffers on a path from the clock source to a clock pin, with the clock source counted as a buffer. A cluster is the collection of a clock buffer and its associated clock pins. Each cluster is tagged with a *stage number*, which is exactly the number of buffers on the path between the clock source and the clock buffer of the cluster. The number includes the clock source and clock buffers of the cluster. In summary, we have the following linear time zero skew clock routing algorithm.

### Algorithm 4.1 (Zero Skew Algorithm)

*S1: Let  $s = \text{number of clock tree stages.}$*

*S2: If  $s = 0$ , report results and exit; continue, otherwise.*

*S3: For each cluster in stage  $s$ , do*

*S3.1: Treat each clock pin in the cluster as a tapping point. Repeat steps S3.2 and S3.3 until there is only one tapping point left.*

*S3.2: Pair-up tapping points.*

*S3.3: For each pair, perform zero-skew-merge of two subtrees and determine new tapping point, using the algorithm discussed in this section. If only one point left in the group, then do nothing.*

*S3.4: Connect the last tapping point directly to the clock buffer output node.*

*S4: Let  $s = s - 1$ . Continue from S2.*

The zero skew algorithm does not depend on the algorithm used for grouping the clock pins or tapping points into pairs. For any pairing algorithm, the zero skew algorithm will work fine. However, to optimize wirability, minimum weighted matching algorithm maybe better. Or a more efficient algorithm that alternately partitions the clock pins into two equal numbered groups can be used.

For implementation in real environment, we have to consider blockages and the difference of electric constants on different layers. The connection between any two tapping points can be done by any existing wiring algorithm that handles wiring blockages. The tapping point is then found by searching through each wiring segment of different electric constants.

To minimize the total wire length, we may construct a few possible wiring patterns (ex. two one-bend connections) between each pair of tapping points, and pick up the one that gives shorter length at the next higher level pairing process.

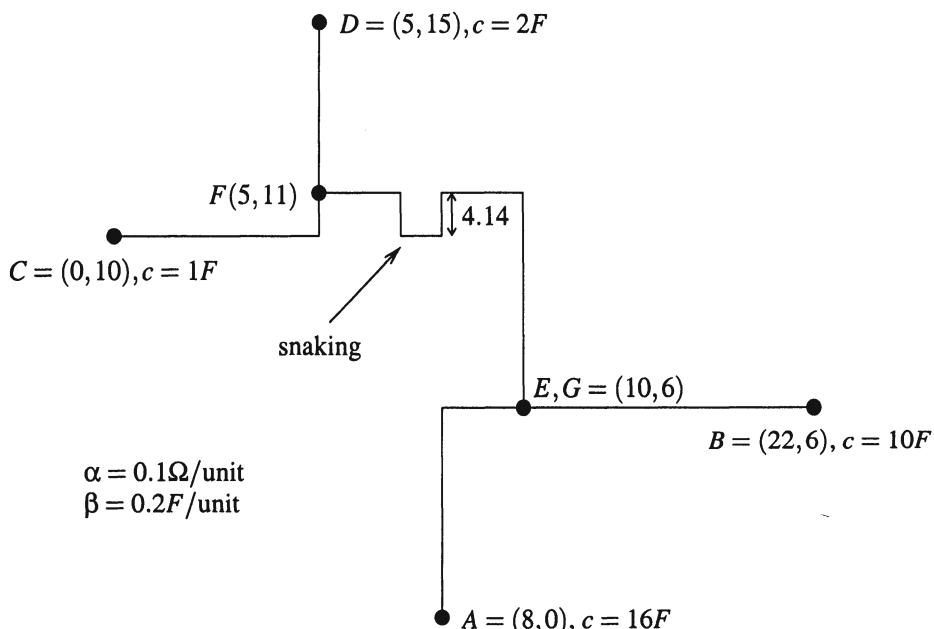


Figure 5. A zero skew wiring result of a simple example.

**Example:** An example with four clock pins (Fig. 5) is used to illustrate the algorithm. Pin A is at (8,0) with 16F loading capacitance. Pin B is at (22,6) with 10F capacitance. Pin C is at (0,10) with 1F. Pin D is at (5,15) with 2F. Per unit resistance is  $0.1\Omega$ , and per unit capacitance is  $0.2F$ . Pin A and B are in one pair and C, D in other pair. According to the algorithm, a tapping point E is decided to be on (10,6) so that the delays to both A and B are all equal to 13.44ns. Similarly, a tapping point F is located at (5,11) for connection of pins C and D, with equal delay 0.96ns. The two subtrees rooted by E and F are VERY unbalanced. We find that  $x = -0.175 < 0$ . The wire connecting E and F has to be elongated by 8.28 units, and the tapping point G has to coincide with E. The final wiring result is shown in Fig. 5. Note that the connection between

(A, B) and (C, D) are chosen from the two one-bend connections of each pair for shorter wire length between (E, F).

## 5. Experimental Results

We test our algorithm on five different sized examples. The statistics of the examples are shown in Table 1. The chip width and height units are both in 1/10 microns. We assume per unit resistance is  $3m\Omega$ , and per unit capacitance is  $0.02fF$ . The loading capacitances of clock pins are ranging from  $30fF$  to  $80fF$ . For simplicity, we assume all are one-stage clock trees, i.e. no intermediate clock buffers. All experiments are conducted on an IBM 3090 machine.

Examples	r1	r2	r3	r4	r5
No. Pins	267	598	862	1903	3101
Chip width	69984	94016	97000	126970	142920
Chip height	70000	93134	98500	126988	145224

Table 1. The Statistics of Testing Examples.

Algorithm	Zero Skew			Length Balancing	
	Examples	Phase delay (ns)	Skew (ns)	Runtime (s)	Phase delay (ns)
r1	1.799	0	0.1	1.798	0.132
r2	4.631	0	0.3	5.367	0.806
r3	7.055	0	0.5	7.655	0.702
r4	20.666	0	1.2	23.316	3.558
r5	35.918	0	2.0	38.958	1.931

Table 2. A Comparison Between the Zero Skew Algorithm and a Wire Length Balancing Heuristic.

We use a simple heuristic for pairing up clock pins in this experiment. We recursively partition the pins into two equal (or almost equal) halves by the median of the sorted pin list in alternate horizontal and vertical directions. This heuristic creates a binary tree for each example. Then the pins are connected based on the zero skew algorithm. For comparison, we also implement a wire length balancing heuristic [4] on the same binary tree. The results are shown in Table 2. It is needless to say that the zero skew algorithm is very important for eliminating the clock skew, especially as for large chips.

## 6. Conclusions

We have presented a novel zero skew clock routing algorithm based on Elmore delay calculation. The approach is ideal for constructing large systems. It

can be modified for customized skew (for cycle stealing) and multi-phase clock systems [6]. We expect this clock routing algorithm will be widely used for performance enhancement for synchronous VLSI digital systems.

## References

- [1] Corman, T. H., C. E. Leiserson, and R. L. Rivest (1990). *Introduction to Algorithms*. McGraw Hill, New York.
- [2] Fishburn, J. P. (1990). "Clock skew optimization," *IEEE Transactions on Computer* C-39(7), pp. 945–951.
- [3] Jackson, M. A. B., A. Srinivasan, and E. S. Kuh (1990). "Clock routing for high performance IC's". In *Proceedings of Design Automation Conference*, pages 573–579.
- [4] Kahng, Andrew, Jason Cong, and Gabriel Robins (1991). "High performance clock routing based on recursive geometric matching". In *Proceedings of Design Automation Conference*, pages 322–327.
- [5] Rubinstein, J., P. Penfield, and M. A. Horowitz (1983). "Signal delay in rc tree networks", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(3), pp. 202–211.
- [6] Tsay, Ren-Song (1991). Exact zero skew. Technical Report RC 16683, IBM Yorktown Research Center.

# EFFICIENT NETWORK FLOW BASED MIN-CUT BALANCED PARTITIONING

Honghua Hannah Yang<sup>1</sup> and D. F. Wong<sup>2</sup>

*Department of Computer Sciences*

*University of Texas at Austin*

## Abstract

We consider the problem of bipartitioning a circuit into two balanced components that minimizes the number of crossing nets. Previously, Kernighan and Lin type (K&L) heuristics, simulated annealing approach, and analytical methods were given to solve the problem. However, network flow techniques were overlooked as viable approaches to min-cut balanced bipartition due to their high complexity. In this paper we propose a balanced bipartition heuristic based on repeated max-flow min-cut techniques, and give an efficient implementation that has the same asymptotic time complexity as that of one max-flow computation. We implemented our heuristic algorithm in a package called FBB. The experimental results demonstrate that FBB outperforms K&L heuristics and analytical methods in terms of the number of crossing nets, and our efficient implementation makes it possible to partition large circuit netlists with reasonable runtime. For example, the average elapsed time for bipartitioning a circuit S35932 of almost 20K gates is less than 20 minutes on a SPARC10 with 32MB memory.

## 1. Introduction

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design, such as floorplanning, placement and multi-chip, multi-FPGA partitioning. *Min-cut balanced bipartition* is the problem of partitioning a circuit into two disjoint components with equal weights such that the number of nets connecting the two components is minimized. The min-cut balanced bipartition problem was shown to be NP-complete [11]. Because of its importance, many heuristic algorithms have been devised for its solution. Among the well-known heuristics are the following [6] Kernighan and Lin type (K&L) iterative improvement methods [20], [9], simulated annealing approaches [19], and analytical methods for the ratio-cut objective [25], see e.g., [15], [4], [23].

The well-known network max-flow min-cut theorem [8], [22], [7], [10], [16] is an important combinatorial optimization technique. It has many applications in VLSI design such as linear placement [3], min-cut replication [13], [14], and FPGA technology mapping [2], [27]. The network max-flow min-cut technique is in fact the most natural method for finding a min-cut in a graph. However,

---

Authors are currently with <sup>1</sup>Intel Corp. and <sup>2</sup>Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign.

it was overlooked as a viable approach for circuit partitioning due to the following reasons: (1) The two components obtained by the network max-flow min-cut technique are not necessarily balanced, (2) Although a balanced cut can be achieved by repeatedly applying min-cut to the larger component, this method can possibly incur  $n$  max-flow computations, where  $n$  is the size of flow network, making it impractical for large problem instances. (3) The traditional network flow technique works on graphs, but hypergraphs are more accurate models for circuit netlists than graphs.

In this paper we explore solutions to the above problems faced by the traditional network flow technique. We first propose a method for exactly modeling a netlist (or equivalently, a hypergraph) by a flow network, and a balanced bipartition heuristic based on a repeated max-flow min-cut technique. We then give an efficient implementation that has the same asymptotic time complexity as that of one max-flow computation.

We use a generalized notion of the balanced bipartition, the *r-balanced bipartition* (also used in [9]), which is a bipartition such that one component is of weight a fraction  $r$  of the total weight  $W$ . As a special case when  $r = 1/2$ , an  $r$ -balanced bipartition is a balanced bipartition. Since in practice there is little reason to strictly enforce the  $r$ -balanced criterion, we introduce a *deviation factor*  $\epsilon$  to allow the component weight to deviate from  $(1 - \epsilon)rW$  to  $(1 + \epsilon)rW$ . We show in Theorem 3.2 that both the runtime and the cut size produced by our algorithm are decreasing functions of  $\epsilon$ . This kind of direct relationship was not shown in previous partitioning heuristics.

The rest of this paper is organized as follows. In Section 2, we first present a method for exactly modeling a netlist by a flow network, and an optimal algorithm for finding a min-net-cut bipartition (not necessarily balanced) of a circuit with respect to a source and a sink. This algorithm serves as a basic procedure for our min-cut balanced bipartition heuristic. We also show that the *most r-balanced min-cut bipartition problem* is NP-complete. We then present our heuristic algorithm for finding a min-net-cut  $r$ -balanced bipartition based on the repeated network flow technique in Section 3 with an efficient implementation. We compare our balanced bipartition results with those of K&L heuristics and analytical methods in Section 4, and conclude the paper in Section 5.

## 2. Optimal Min-Net-Cut Bipartition

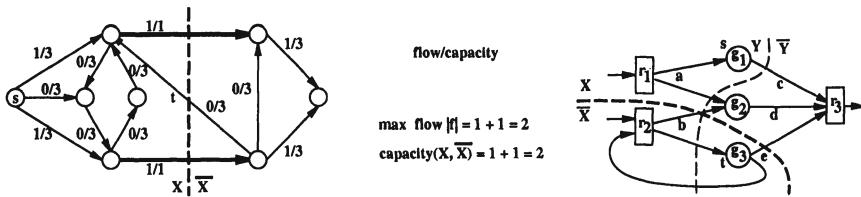
We first give some definitions of a flow network. A flow network  $G = (V, E)$  is a directed graph in which each edge  $e \in E$  has a capacity  $c(e) \geq 0$ . Two nodes  $s$  and  $t$  in  $V$  are specified:  $s$  is called the *source*,  $t$  is called the *sink*. Figure 1(a) shows an example of a flow network and a max-flow. The label  $x/y$  on an edge indicates that the flow and the capacity on the edge are  $x$  and  $y$  respectively. The dark edges are the forward edges of the corresponding min-cut ( $X, \bar{X}$ ). An  $s-t$

*flow* (or *flow* for short) in  $G$  is a real-valued function  $f : E \rightarrow R$  such that (1) for all  $e \in E$ ,  $0 \leq f(e) \leq c(e)$ , and (2) for all  $u \in V \setminus \{s, t\}$ , the sum of the incoming flow into  $u$  is equal to the sum of the outgoing flow from  $u$ . An edge  $e$  in  $E$  is *saturated* if  $f(e) = c(e)$ . The *value*  $|f|$  of a flow  $f$  is defined as the sum of the flow outgoing from  $s$ , which is equal to the sum of the flow incoming to  $t$ . A *maximum-flow* (or *max-flow* for short) in  $G$  is a flow of maximum value from  $s$  to  $t$ .

An *s-t cut* (or *cut* for short)  $(X, \bar{X})$  of a flow network  $G = (V, E)$  is a bipartition of  $V$  into  $X$  and  $\bar{X}$  such that  $s \in X$  and  $t \in \bar{X}$ . An edge whose starting node is in  $X$  and ending node is in  $\bar{X}$  is called a *forward edge*. An edge whose ending node is in  $X$  and starting node is in  $\bar{X}$  is called a *backward edge*. The *capacity of the cut*  $(X, \bar{X})$ , denoted by  $\text{cap}(X, \bar{X})$ , is the sum of the capacities on the *forward edges only* from  $X$  to  $\bar{X}$ . An *augmenting path* from  $u$  to  $v$  in  $G$  is a simple path from  $u$  to  $v$  in the undirected graph resulting from the network by ignoring edge directions, that can be used to push additional flow from  $u$  to  $v$ .

**Theorem 2.1** Max-flow min-cut theorem [8]

Given a max-flow  $f$  in  $G$ , let  $X = \{v \in V : \exists \text{ an augmenting path from } s \text{ to } v \text{ in } G\}$ , and let  $\bar{X} = V \setminus X$ . Then  $(X, \bar{X})$  is a cut of minimum capacity (which is equal to  $|f|$ ), and  $f$  saturates all forward edges from  $X$  to  $\bar{X}$ .



**Figure 1.** (a) A flow network  $G$ , and a max-flow  $f$  in  $G$ . (b) A digraph  $N$  representing a sequential circuit and its net-cuts.

## 2.1 Modeling a Net in a Flow Network

We represent a *sequential circuit* netlist as a digraph  $N = (V, E)$  where  $V$  is a set of nodes representing combinational gates and registers, and  $E$  is a set of edges representing interconnections between gates and registers. Each node  $v$  in  $V$  has an associated weight  $w(v) \in R^+$ . The total weight of a subset  $U \subseteq V$  is denoted by  $w(U)$ . Let  $W = w(V)$  denote the total weight of the circuit  $N$ .

A *net*  $n = (v; v_1, \dots, v_l)$  is a set of outgoing edges from node  $v$  in  $N$ . For example in Figure 1(b), net  $a$  consists of two edges  $(r_1, g_1)$  and  $(r_1, g_2)$ . Given two nodes  $s$  and  $t$  in  $N$ , an  $s$ - $t$  *cut* (or *cut* for short)  $(X, \bar{X})$  of  $N$  is a bipartition of the nodes in  $V$  such that  $s \in X$  and  $t \in \bar{X}$ . The *net-cut*  $\text{net}(X, \bar{X})$  of the cut is

the set of nets in  $N$  that are incident to nodes in both  $X$  and  $\bar{X}$ . A cut  $(X, \bar{X})$  is a *min-net-cut* if  $|net(X, \bar{X})|$  is minimum among all  $s-t$  cuts of  $N$ . In Figure 1(b),  $net(X, \bar{X}) = \{b, e\}$ ,  $net(Y, \bar{Y}) = \{c, a, b, e\}$ , and  $(X, \bar{X})$  is a min-net-cut.

In order to find a min-net-cut in  $N = (V, E)$ , we reduce it to the problem of finding a cut of minimum capacity, and then solve the latter problem by the max-flow min-cut theorem. If the cut edges all have unit capacity, then the problem is equivalent to finding a cut with the minimum number of forward edges from  $X$  to  $\bar{X}$ .

We construct a flow network  $N' = (V', E')$  from  $N = (V, E)$  as follows (see Figures 2 & 3):

1.  $V'$  contains all nodes in  $V$ .
2. For each net  $n = (v; v_1, \dots, v_l)$  in  $N$ , add two nodes  $n_1$  and  $n_2$  in  $V'$  and a *bridging edge*  $(n_1, n_2)$  in  $E'$ .
3. For each node  $u \in \{v, v_1, \dots, v_l\}$  incident on net  $n$ , add two edges  $(u, n_1)$  and  $(n_2, u)$  in  $E'$ .
4. Let  $s$  be the source of  $N'$  and  $t$  the sink of  $N'$ .
5. Assign unit capacity to all bridging edges and infinite capacity to all other edges in  $E'$ .
6. For a node  $v \in V'$  corresponding to a node in  $V$ ,  $w(v)$  is the weight of  $v$  in  $N$ . For a node  $u \in V'$  split from a net,  $w(u) = 0$ .

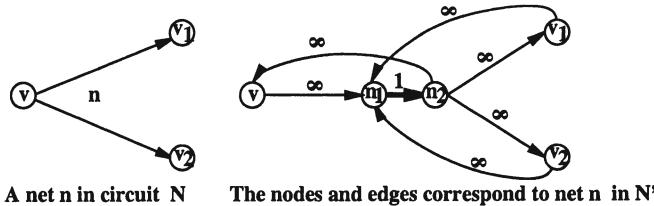


Figure 2. Modeling a net in  $N$  in the flow network  $N'$ .

Note that all nodes incident on net  $n$  are connected to  $n_1$  and are connected from  $n_2$  in  $N'$ . Hence the flow network construction is symmetric with respect to all nodes incident on a net. We show in Lemma 2.1 that the size of  $N'$  is only a constant factor larger than the size of  $N$ , for a connected graph  $N$ .

**Lemma 2.1** *Let  $N' = (V', E')$  be the flow network constructed from a digraph  $N = (V, E)$  using the above method. Then  $|V'| \leq 3|V|$  and  $|E'| \leq 2|E| + 3|V|$ .*

The above flow network construction for modeling net-cuts also works when the circuit  $N$  is represented by a hypergraph. We note that another optimal approach for finding a min-net-cut of a hypergraph was given in [16] by modeling a net (or a hyperedge) as a star node, and then transforming a node-capacitated

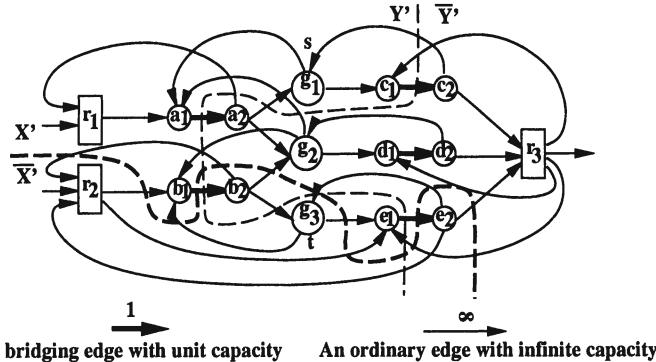


Figure 3. A flow network  $N'$  constructed from the circuit  $N$  in Figure 1(b).

flow network into an edge-capacitated network [22] by splitting *every* node. Our method is different from that of [16] in that we split the nodes corresponding to the nets *only*, and hence use fewer nodes and edges in the resulting flow network. For a huge input circuit, our method translates into less memory usage and faster runtime.

Another related result given in [18] shows that modeling hypergraphs by graphs (with positive weights) with the same min-cut properties is not possible. However, our method models a hypergraph for network flow based partition algorithms only. The differences between the model used in [18] and our model are the following: (1) The weight of a cut in [18] is computed by the sum of all cut edges with fixed weights, while the weight of a cut in our model is computed by the sum of the capacities of the forward edges only. (2) [18] tries to model hypergraphs for a wide range of existing partition algorithms developed for ordinary graphs only, while our method just models a hypergraph for network flow based partition algorithms. Hence our method is able to exactly model a hypergraph for our flow based partitioning algorithm without contradicting the result in [18].

It is easy to see that  $N'$  is a strongly connected digraph. The strong connectivity of  $N'$  is the key to reducing the bi-directional min-net-cut problem to the minimum capacity cut problem that counts the capacity of the forward edges only. We show in Theorem 2.2 and Corollary 2.2.1 that the problem of finding a min-net-cut in  $N$  can be reduced to the problem of finding a cut with minimum capacity in  $N'$ .

**Theorem 2.2**  $N$  has a cut of net-cut size at most  $C$  if and only if  $N'$  has a cut of capacity at most  $C$ .

**Corollary 2.2.1** Let  $(X', \bar{X}')$  be a cut of minimum capacity  $C$  in  $N'$ , and let  $(X, \bar{X})$  be the cut in  $N$  as constructed in Theorem 2.2 (2). Then  $(X, \bar{X})$  is a min-net-cut in  $N$ , and  $|net(X, \bar{X})| = C$ .

## 2.2 Optimal Network Flow Based Min-Net-Cut Bipartition

Based on Theorem 2.2 and Corollary 2.2.1, we give an optimal algorithm for finding a bipartition (not necessarily balanced) of a circuit  $N = (V, E)$  with respect to two nodes  $s$  and  $t$  that minimizes the number of crossing nets.

### Algorithm 1: Finding A Min-Net-Cut

0. Construct the flow network  $N' = (V', E')$  for  $N$  as described in Subsection 2.1;
1. Find a max-flow in  $N'$  from  $s$  to  $t$ ;
2. Find a cut  $(X', \bar{X}')$  of minimum capacity in  $N'$  as described in the max-flow min-cut theorem;
3. Find a min-net-cut  $(X, \bar{X})$  in  $N$  as described in Theorem 2.2 (2).

**Theorem 2.3** Algorithm 1 finds a min-net-cut in a circuit  $N = (V, E)$ , and terminates in  $O(|V||E|)$  time where  $N$  is a connected circuit.

**Proof:** The correctness of Algorithm 1 has been established in Theorem 2.2 and Corollary 2.2.1.

Steps 0, 2 and 3 take linear time in the size of  $N$ . We use the simple augmenting path algorithm [8] to implement step 1. Finding an augmenting path in  $N'$  takes  $O(|E'|)$  time. The number of augmenting paths in  $N'$  is equal to the number of bridging edges in a min-net-cut, which is at most  $|V|$  (usually much less). Hence Algorithm 1 takes  $O(|V||E'|) = O(|V|(|E| + |V|)) = O(|V||E|)$  time by Lemma 2.1.  $\square$

There are other asymptotically faster (worse-case) algorithms for finding a max-flow than the simple augmenting path algorithm based on the Ford and Fulkerson method. The fastest preflow method takes  $O(|E||V| \log(|V|^2/|E|))$  time [12] with a large constant factor. The Ford and Fulkerson method takes  $O(|E| * \text{max-flow-value})$  time. The latter method is efficient in our application, since the max-flow-value in the special flow network we construct is at most  $|V|$  (usually much smaller than  $|V|$ ).

## 2.3 Most $r$ -balanced Min-Net-Cut Bipartition is NP-complete

The min-cut bipartition may yield unbalanced components. The max-flow computation defines a set of min-cuts with the same cut size, but with varying weights in the two partitions. It is natural to ask the question of whether one can find a min-cut that is the *most  $r$ -balanced* among all the min-cuts defined by a max-flow, i.e., among all possible min-cuts  $(X, \bar{X})$  defined by a max-flow find the min-cut such that  $|w(X) - rw(V)|$  is as close to 0 as possible. One can

show that the decision version of the problem is NP-complete by reducing the weighted subset sum problem [11] to it.

### 3. Min-Cut Balanced Bipartition

It is not difficult to see that repeatedly applying the max-flow min-cut technique to cut the larger of the two partitions will eventually produce a balanced bipartition with a natural small net-cut. However, this approach was overlooked as a viable heuristic approach to circuit partitioning due to its high complexity (possibly  $|V|$  max-flow computations). In this section we first describe a repeated max-flow min-cut heuristic algorithm, Flow-Balanced-Bipartition (FBB), for finding an  $r$ -balanced bipartition that minimizes the number of crossing nets. We then give an efficient implementation of FBB that has the same asymptotic time complexity as one max-flow computation. For ease of presentation, we will describe our algorithm in terms of the original circuit and net-cuts, instead of the flow network constructed from the circuit (as shown in Section 2.1) and forward bridging edges, when there is no confusion. An illustration of Algorithm 2 is shown in Figure 4.

#### 3.1 Balanced Bipartition Heuristic

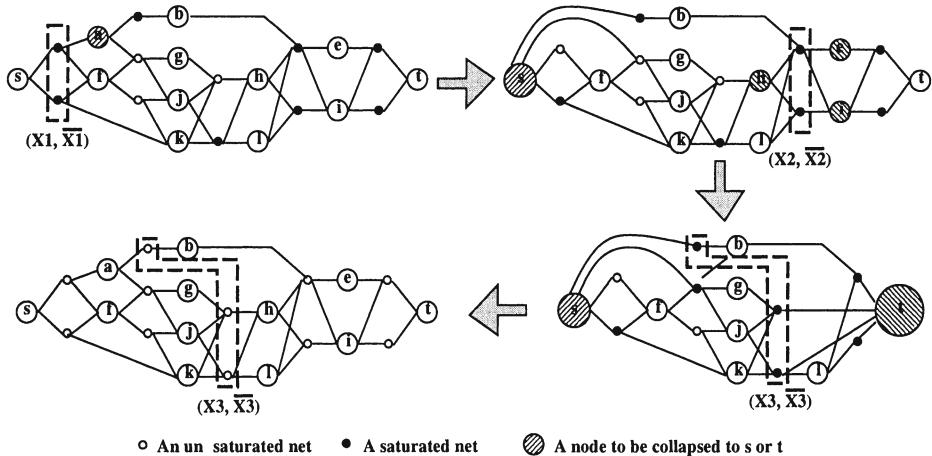
Given a circuit  $N = (V, E)$ , FBB randomly picks a pair of nodes  $s$  and  $t$  in  $N$ , and then tries to find an  $r$ -balanced bipartition that separates  $s$  and  $t$ , and that minimizes the number of crossing nets. Let  $W$  be the total weight of the circuit  $N$ . Since in practice there is little reason to strictly enforce the  $r$ -balanced criterion, we allow the component weights to deviate from  $(1 - \varepsilon)rW$  to  $(1 + \varepsilon)rW$ . Given a subcircuit  $X$  of  $N$ , let  $w(X)$  denote the total weight of nodes in  $X$ .

**Algorithm 2:** Flow-Balanced-Bipartition (FBB)

0. Randomly pick a pair of nodes  $s$  and  $t$  in  $N$ ;
1. Find a min-net-cut  $C$  in  $N$ ;
 

Let  $X$  be the subcircuit reachable from  $s$  through augmenting paths in the flow network, and  $\bar{X}$  the rest;

  2. If  $(1 - \varepsilon)rW \leq w(X) \leq (1 + \varepsilon)rW$   
then stop and return  $C$  as the answer.
  3. If  $w(X) < (1 - \varepsilon)rW$   
then
    - 3.1. collapse all nodes in  $X$  to  $s$ ;
    - 3.2. collapse to  $s$  a node  $v \in \bar{X}$  adjacent to  $C$ ;
    - 3.3. goto 1;
  4. If  $w(X) > (1 + \varepsilon)rW$   
then
    - 4.1. collapse all nodes in  $\bar{X}$  to  $t$ ;
    - 4.2. collapse to  $t$  a node  $v \in X$  adjacent to  $C$ ;
    - 4.3. goto 1.



*Figure 4.* An illustration of Algorithm 2 for  $r = 1/2$ ,  $\epsilon = 0.15$  and unit weight for each node. Hence a component can have weight between 5 and 7. If  $\epsilon = 0.45$ , then a component can have weight between 3 and 10, and Algorithm 2 would terminate after finding cut  $(X_2, \bar{X}_2)$ . A small solid node indicates that the bridging edge corresponding to the net is saturated with flow.

Step 1 can be implemented by Algorithm 1. In step 3.2, we need to collapse a node  $v \in \bar{X}$  incident on a cut net to  $s$  since otherwise the same set of nets in  $C$  will again be chosen as the min-net-cut in the next iteration in step 1. The reasons why we adopt this node collapsing method instead of a more gradual method (i.e., increasing the unit capacities of the bridging edges by a fixed amount as in [17] are (1) the capacity of the cut would no longer reflect the real net-cut size, and (2) the runtime would not be bounded by one flow computation. By collapsing  $v$  to  $s$ , FBB is able to explore a different net-cut with a *larger*  $X$  in the next iteration. Note that the size of the min-net-cut found in the next iteration will be the same as or larger than the size of the min-net-cut in the current iteration. A similar argument holds for step 4.2.

We now describe our strategy for picking a node in steps 3.2 and 4.2. To find an  $r$ -balanced bipartition that minimizes the net-cut size, our heuristic is to always focus on finding a min-net-cut at each iteration. But when the remaining circuit is very large, the current min-net-cut has less influence on what the final balanced min-net-cut would be. Therefore, we randomly pick a node in steps 3.2 and 4.2 in order to speed up the algorithm. When the remaining circuit becomes small enough, we need to be more careful about which node we pick, and we can afford to try out more than one node. We give a threshold value  $R$  for the number of nodes in the uncollapsed subcircuit. If the number of remaining nodes is larger than  $R$ , then we randomly pick one node from the nodes incident on the cut nets in  $C$ . Otherwise, we try all nodes incident on the cut nets in  $C$  and pick the node whose collapsing induces a min-net-cut with the smallest size.

We can also let the probability of choosing a node be inversely proportional to the number of nodes in the remaining (un-collapsed) circuit.

### 3.2 Efficient Implementation of Algorithm 2

A drawback of the repeated max-flow heuristic is that it has a relatively high time complexity. Iteratively applying Algorithm 1 in step 1 of Algorithm 2 to compute a max-flow and a min-net-cut from the zero flow can be very time-consuming. We show an efficient way to deal with this problem. In fact, it is not necessary to do the max-flow computation from the zero flow in every iteration. Instead, we can retain the flow value in the flow network, and only find additional flow to saturate the bridging edges of the min-net-cut from iteration to iteration.

In Procedure 1, we describe the incremental max-flow computation in step 1 of Algorithm 2. Initially, the flow network retains the flow function computed in the previous iteration.

**Procedure 1:** Incremental Flow Computation

0. While  $\exists$  an additional augmenting path from  $s$  to  $t$
1.     increase flow value along the augmenting path;  
      /\* There is no more augmenting path from  $s$  to  $t$ . \*/
2. Mark all nodes  $u$  such that  
       $\exists$  an augmenting path from  $s$  to  $u$ ;
3. Let  $C'$  be the set of bridging edges whose starting  
      nodes are marked and ending nodes are not marked;
4. Return the nets corresponding to the bridging edges in  
       $C'$  as the min-net-cut  $C$ , and the marked nodes as  $X$ .

Since the max-flow computation using the augmenting path method is insensitive to the initial flow values in the flow network and the order in which the augmenting paths are found, the above procedure correctly finds a max-flow with the same flow value as a max-flow computed in the collapsed flow network from scratch (i.e., the zero flow).

We show in Theorem 3.1 that if we fix the threshold  $R$  used in the node-picking strategy described in the previous subsection as a constant, then the total complexity of FBB is  $O(|V||E|)$ , which is the same as the complexity of one max-flow computation.

**Theorem 3.1** *If Procedure 1 is used to implement step 1 of Algorithm 2, then Algorithm 2 has time complexity  $O(|V||E|)$  for a connected circuit  $N = (V, E)$ .*

**Proof:** Since each augmenting path computation takes  $O(|E|)$  time, we prove that the total time complexity of step 1 of Algorithm 2 is  $O(|V||E|)$  by showing that there are at most  $2|V|$  augmenting path computations in the following iterations.

The total flow value  $|f|$  in the flow network  $N'$  constructed from  $N$  at the end of Algorithm 2 is the number of forward bridging edges in the final min-net-cut. Hence  $|f|$  is at most  $|V|$ . Since bridging edges have unit capacity, there are  $|f| \leq |V|$   $s-t$  augmenting paths found at the end of Algorithm 2. We now consider an augmenting path computation in step 0 of Procedure 1. Either an augmenting path is found, in which case the number of augmenting paths increases by 1, or at least one node will be collapsed to  $s$  or  $t$  in steps 3.1 and 3.2 or 4.1 and 4.2 of Algorithm 2. Hence the number of augmenting path computations in the following iterations is at most  $2|V|$ .

Note that step 3 of Procedure 1 can be accomplished during the searching for an augmenting path in step 2 of Procedure 1, and steps 4 and 5 of Procedure 1 takes  $O(|V|)$  time in the worst case.  $\square$

In practice, as shown in the experimental results, Algorithm 2 terminates much faster than the  $O(|V||E|)$  worst case time complexity. Because of the construction of the flow network in Section 2.1 where the bridging edges have unit capacity, the number of augmenting paths found in Algorithm 2 is the same as the size of the net-cut found in  $N$ , which is much less than  $|V|$ .

**Theorem 3.2** *The number of iterations and the final net-cut size of Algorithm 2 are non-increasing functions of  $\epsilon$ .*

**Proof:** Fewer iterations are needed in Algorithm 2 when  $\epsilon$  is larger, since the condition in step 2 of Algorithm 2 is satisfied in fewer iterations.

If an augmenting path from  $s$  to  $t$  is found in step 1 of Procedure 1, then the flow value is increased by at least 1 and hence the size of the min-net-cut is increased by at least 1. If an augmenting path from  $s$  to  $t$  is not found in step 1 of Procedure 1, then the size of the min-net-cut is equal to the flow value of the previous iteration, which is equal to the previous min-net-cut size. Hence the net-cut size found in each iteration is non-decreasing.  $\square$

Theorem 3.2 guarantees that with a larger  $\epsilon$  deviation factor we can improve the efficiency of Algorithm 2 and obtain a better partitioning solution. This property is not true for other partitioning approaches such as the K&L heuristics. Another interesting corollary of Theorem 3.2 is that the longer the execution time of Algorithm 2, the worse the net-cut size in the final solution. This property of Algorithm 2 can be utilized to further improve the efficiency of Algorithm 2.

## 4. Experimental Results

We implemented Algorithm 2 in a package called FBB using the C language, and integrated FBB in SIS/MISII [1]. Currently FBB runs on circuit formats accepted by SIS/MISII. We tested FBB on a set of large ISCAS and MCNC benchmark circuits using a SPARC 10 workstation with a 36Mhz SS10 and 32MB memory (the C code was compiled with gcc without the optimizer). For

name	circuit			ave. net-cut size			FBB bipart. ratio	impro. %	
	gates & latches	nets	ave deg	SN	PFM3	FBB		over SN	over PFM3
C1355	514	523	3.0	38.9	29.1	26.0	1:1.08	33.2	10.7
C2670	1161	1254	2.6	51.9	46.0	37.1	1:1.15	28.5	19.3
C3540	1667	1695	2.7	90.3	71.0	79.8	1:1.11	11.6	-12.4
C7552	3466	3565	2.7	44.3	81.8	42.9	1:1.08	3.2	47.6
S838	478	511	2.6	27.1	21.0	14.7	1:1.04	45.8	30.0
Ave							1:1.10	24.5	19.0

Table 1. Comparison of bipartition results of SN, PFM3, and FBB (with  $r = 1/2$  and  $\epsilon = 0.1$ ).

each circuit tested, the number of gates and latches, the number of nets, and the average net degree (i.e., the average number of nodes connected to a net) are given in Tables 4 and 1. Note that the actual number of nodes in a circuit includes PI nodes, and is therefore more than the the number of gates and latches.

Table 4 compares the average bipartition results of FBB with the results reported by Dasdan and Aykanat in [5]. The program SN is based on the K&L heuristic algorithm in Sanchis [24], which is a generalization of the Krishnamurthy [21] algorithm. The program PFM3 is based on a K&L heuristic with free moves as described in [5]. SN was run 20 times and PFM3 was run 10 times on each circuit starting from different randomly generated initial partitions, while FBB was run 10 times on each circuit from different randomly generated  $s$  and  $t$  as the source and the sink respectively. Table 4 shows that with only one exception, FBB outperforms both SN and PFM3 on the 5 circuits. On average, FBB finds a bipartition with 24.5% and 19.0% fewer crossing nets than SN and PFM3 respectively. This is not too surprising since max-flow min-cut techniques tends to find a natural small cut. The average actual ratios of the two partitions obtained by FBB are also shown in Table 4. Since we set  $\epsilon = 0.1$ , the actual ratios of the two partitions are roughly the same (1:1.10 on average).

We did not compare the runtime of SN, PFM3, and FBB since they were run on different workstations. SN and PFM3 were run on a SUN SPARC ELC, and FBB were run on a SUN SPARC 10. For example, for C3540, the average elapsed time (not CPU time) in seconds of SN, PFM3, and FBB for each run are 90.3, 71.0, and 13.6 respectively; and for C7552, the average elapsed time in seconds of SN, PFM3, and FBB for each run are 44.3, 81.8, and 18.8 respectively.

Table 1 compares the best bipartition net-cut size of EIG1 (Hagen and Kahng [15]), PARABOLI (Riess, Doll, and Frank [23]), and FBB. EIG1 and PARABOLI are two programs based on analytical methods and their results were obtained from [23]. The results produced by PARABOLI were the best previously known results reported on the benchmark circuits. The results for FBB were the best of 10 runs. The elapsed time of FBB for the run that generates the best result was

circuit				best net-cut size			improv % over		FBB elaps. sec.
name	gates & latches	nets	ave deg	EIG1	PB	FBB	EIG1	PB	
S1423	731	743	2.7	23	16	13	43.5	18.8	1.7
S9234	5808	5805	2.4	227	74	70	69.2	5.4	55.7
S13207	8696	8606	2.4	241	91	74	69.3	18.9	100.0
S15850	10310	10310	2.4	215	91	67	68.8	26.4	96.5
S35932	18081	17796	2.7	105	62	49	53.3	21.0	2808
S38584	20859	20593	2.7	76	55	47	38.2	14.5	1130
S38417	24033	23955	2.4	121	49	58	52.1	-18.4	2736
Average							58.5	11.3	

Table 2. Comparison of bipartition results of EIG1, PARABOLI (PB) and FBB (with  $r = 1/2$  and  $\epsilon = 0.1$ ). All results allow up to 10% deviation from bisection.

also recorded. All results in Table 1 allow up to 10% deviation from bisection. On average, FBB outperforms EIG1 and PARABOLI by 58.1% and 11.3% respectively. For circuit S38417, FBB produces a larger net-cut than PARABOLI does. We consider the following possible explanations: 1) If FBB is run more than 10 times, the best net-cut result is likely to be better. 2) In a huge circuit like S38417, the solution is sensitive to the selection of the initial  $s, t$  pair of nodes. Applying circuit clustering techniques based on the connectivity information before partitioning may improve the partitioning result of FBB.

Note that different programs using the same MCNC benchmark circuits reported different properties such as the number of cells and the number of nets for these circuits. This is because when a netlist format is translated to a hypergraph, some unnecessary details such as inverters are omitted. However, the underlying netlist structures are the same.

In the experiment, we have consistently observed that for the runs with longer-than-average runtime, FBB always generates exceptionally poor solutions. This can be explained by Theorem 3.2, since the net-cut size is non-decreasing with more iterations. This property of FBB is in contrast to both the K&L heuristics and the simulated annealing heuristics, where longer runtime means better solutions. This property of FBB provides another way of improving the efficiency of FBB. We can pick a reasonable upperbound for the runtime of FBB (for example, based on a few runs of FBB), stop FBB when the runtime exceeds the upperbound, and restart FBB using a new pair of nodes  $s$  and  $t$ . By doing so we are not likely to lose any good solutions, but we will further improve the efficiency of FBB.

## 5. Conclusions and Discussions

We have described a method for exactly modeling a netlist by a flow network, presented a balanced bipartition heuristic based on the repeated max-flow min-cut techniques, and given an efficient implementation of a good theoret-

ical method. We implemented our algorithm in a package called FBB. The experimental results demonstrate that the repeated max-flow min-cut heuristic outperforms K&L heuristics and analytical methods in terms of the number of crossing nets, and the efficient implementation enables our heuristic algorithm to partition large benchmark circuits with reasonable runtime.

FBB has predictable behavior in terms of the sizes of the two partitions, and the direct relationship between efficiency, solution quality of FBB, and relaxing the  $r$ -balanced criterion by using a larger  $\epsilon$ . Such a direct relationship was not shown in previous heuristics for circuit partition. We also believe that the choice of the pair of nodes  $s$  and  $t$  as the initial configuration of FBB has less influence on the solution than an initial bipartition would have. Hence the solution quality of FBB is less sensitive to the initial choice of  $s$  and  $t$ .

Our algorithm can be easily extended to handle that case where the nets in a circuit have different weights. We can simply assign the weight of a net to its corresponding bridging edge in the flow network, and FBB will find a net-cut with its weight minimized.  $K$ -way min-cut partitioning for  $K > 2$  can be accomplished by recursively applying FBB, or by setting  $r = 1/K$  and then using FBB to find one partition at a time. We are currently investigating more natural methods based on flow networks for the  $K$ -way min-cut partitioning problem.

Pre-partitioning circuit clustering according to the connectivity or the timing information of the circuit can be easily incorporated into FBB by treating a cluster as a node. A possible extension to FBB would be to combine FBB with the K&L heuristics and the simulated annealing heuristics. We can use FBB to find a natural small net-cut as a good initial partition, and then apply the K&L heuristics or the simulated annealing heuristics with low temperature to fine-tune the solution.

## References

- [1] R. K. Brayton, R. Rudell, and A. L. Sangiovanni-Vincentelli. MIS: A Multiple-Level Logic Optimization. *IEEE Trans. on CAD*, pp. 1061–1081, Nov. 1987.
- [2] J. Cong and Y. Ding. An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pp. 48–53, Nov. 1992.
- [3] C.-K. Cheng. Linear Placement Algorithms and Applications to VLSI Design. *Networks*, vol. 17, pp. 439–464, 1987.
- [4] J. Cong, L. Hagen, and A. Kahng. Net Partitions Yield Better Module Partitions. In *Proc. of the 29th ACM/IEEE Design Automation Conf.*, pp. 47–52, 1992.
- [5] A. Dasdan and C. Aykanat. Improved Multiple-Way Circuit Partitioning Algorithms. In *Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays*, Feb. 1994.
- [6] W. E. Donath. Logic Partitioning. In *Physical Design Automation of VLSI Systems*, pp. 65–86. Preas and Lorenzetti eds., Benjamin/Cummings, 1988.
- [7] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

- [8] J. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [9] C. M. Fiduccia and R. M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 175–181, 1982.
- [10] R. Gomory and T. C. Hu. Multi-Terminal Network Flows. *J. SIAM*, vol. 9, pp. 551–570, 1961.
- [11] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [12] A. W. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. SIAM*, vol. 35, pp. 921–940, 1988.
- [13] J. Hwang and A. El Gamal. Optimal Replication for Min-Cut Partitioning. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pp. 432–435, Nov. 1992.
- [14] J. Hwang and A. El Gamal. Min-Cut Replication in Partitioned Networks. *IEEE Trans. on CAD*, vol. 14(1), pp. 96–106, Jan. 1995.
- [15] L. Hagen and A. B. Kahng. Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pp. 10–13, Nov. 1991.
- [16] T. C. Hu and K. Moerder. Multiterminal Flows in a Hypergraph. In *VLSI Circuit Layout: Theory and Design*, pp. 87–93. Hu and Kuh eds., IEEE Press, 1985.
- [17] S. Iman, M. Pedram, C. Fabian, and J. Cong. Finding Uni-Directional Cuts Based on Physical Partitioning and Logic Restructuring. In *the 4th ACM/SIGDA Physical Design Workshop*, April 1993.
- [18] E. Ihler, D. Wagner, and F. Wager. Modeling Hypergraphs by Graphs with the Same Min-Cut Properties. In *Info. Proc. Letters*, 45, pp. 171–175, 1993.
- [19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, pp. 671–680, May 1983.
- [20] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning of Electrical Circuits. *Bell System Technical Journal*, pp. 291–307, Feb. 1970.
- [21] B. Krishnamurthy. An Improved Min-Cut Algorithm for Partitioning VLSI networks. *IEEE Trans. on Computers*, pp. 438–446, May 1984.
- [22] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart & Winston, New York, 1976.
- [23] B. M. Riess, K. Doll, and M. J. Frank. Partitioning Very Large Circuits Using Analytical Placement Techniques. In *Proc. 31th ACM/IEEE Design Automation Conf.*, pp. 646–651, 1994.
- [24] L. A. Sanchis. Multiway Network Partitioning. *IEEE Trans. on Computers*, pp. 62–81, Jan. 1989.
- [25] Y. C. Wei and C. K. Cheng. Towards Efficient Hierarchical Designs by Ratio Cut Partitioning. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pp. 298–301, Nov. 1989.
- [26] H. Yang and D. F. Wong. Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pp. 150–155, Nov. 1994.

# RECTANGLE-PACKING-BASED MODULE PLACEMENT

Hiroshi Murata<sup>1</sup>, Kunihiro Fujiyoshi<sup>2</sup> and Shigetoshi Nakatake<sup>3</sup>

*School of Information Science,*

*Japan Advanced Institute of Science and Technology, Japan*

Yoji Kajitani<sup>4</sup>

*Department of Electrical and Electronic Engineering,*

*Tokyo Institute of Technology, Japan*

## Abstract

The first and the most critical stage in VLSI layout design is the placement, the background of which is the *rectangle packing problem* : Given many rectangular modules of arbitrary size, place them without overlapping on a layer in the smallest bounding rectangle. Since the variety of the packing is infinitely many (two-dimensionally continuous), the key issue for successful optimization is in the introduction of a *P-admissible solution space*, which is a finite set of solutions at least one of which is optimal. This paper proposes such a solution space where each packing is represented by a pair of module name sequences. Searching this space by simulated annealing, hundreds of modules could be successfully packed as demonstrated. Combining a conventional wiring method, the biggest MCNC benchmark ami49 is challenged.

## 1. Introduction

Layout in physical design of VLSI is, simply to say, to pack all the circuit elements in a chip without violating the design rules, so that the circuit performs well and the production yield is high. So much is the variety of targets in different stages, the problem defined as follows is the base of all of them.

### Rectangle Packing Problem: RP

Let  $\mathcal{M}$  be a set of  $m$  rectangular modules whose height and width are given by real numbers. (Orientation is fixed.) A packing of  $\mathcal{M}$  is a non-overlapping placement of the modules. The minimal bounding rectangle of a packing is called the chip. Find a packing of  $\mathcal{M}$  in a chip of the minimum area.

A packing of six modules is shown in Fig.1.

---

Authors are currently with <sup>1</sup>MicroArk Co., Ltd., <sup>2</sup>Tokyo University of Agriculture and Technology, and  
<sup>3,4</sup>University of Kitakyushu.

**RP** can be shown to be NP-hard by reducing an NP-hard problem which is **RP** with a constraint that the width of the chip is fixed [1].

Since the height and width of modules are continuous real numbers, **RP** is not simply a combinatorial optimization problem. Hence there have been several numerical approaches [2, 3].

An alternative approach is “combinatorial search”. Define a solution space which is a system of codes and a mapping from each code to a solution which is a packing in the case of **RP**. Each code represents a constraint imposed on packing. A code is said to be *feasible* if the constraint is consistent, i.e. there exists a packing that satisfies the constraint represented by the code. The mapping defines a consistent packing for a code if it is feasible. The evaluation of a code is the minimum area of the chip of the corresponding packings. The combinatorial search is to search for a better code in the solution space. If a trade-off to the computation time is observed, the heuristics will stop the search on the way and output the one best so far. Being effective this search, the minimum requirement to the solution space is

- (1) The size of the code set is finite,
- (2) Every code is feasible,
- (3) The mapping, as well as the evaluation in our case, for each code is possible in polynomial time, and
- (4) The best evaluated code in the solution space coincides with an optimal solution of the original problem, that is **RP** in our case.

The solution space that satisfies the above four requirements is called *P-admissible*.

The reasons for (1),(3) and (4) are obvious. That for (2) is: most heuristics pick up one solution after another along the neighboring structure defined on the space [4], consulting with the difference of evaluations (gain) to the previous solution. Therefore, if infeasible solutions are included, the continuity will be destroyed and the convergence to a feasible solution is not guaranteed.

A practically known solution space is the one derived from *slicing floorplan* proposed by Otten [5] and others. Since it satisfies (1), (2) and (3) several optimization heuristics are applied for the space, and one of the most successful approaches uses simulated annealing [6]. Since the optimal solution can be non-slicing, it lacks (4). Efforts have been paid to add non-slicing structures [7, 8].

On the other hand, Onodera et.al. [9] uses a solution space by assigning one out of four relations, “left-of”, “right-of”, “above”, “below”, to every pair of modules. This space satisfies (4) since any packing satisfies a combination of the relations. But there are many infeasible codes such as; module *a* is left-of module *b*, *b* is left-of module *c* and *c* is left-of *a*. As a consequence, the

space does not admit such a heuristics as simulated annealing. In their paper, an exhaustive search with a branch-and-bound technique is applied to find an exactly optimal solution, but the size of tractable problems is limited up to six modules. Thus these two are not P-admissible.

This paper provides a P-admissible solution space, in which each code is an ordered pair of module name sequences, which we call the *Sequence-Pair*. Searching this space, we have been able to pack hundreds of modules very efficiently, almost optimally at a look as in Fig.9 and in Fig.10.

Utilizing this solution space of RP for VLSI layout design, the evaluation of a packing has to be modified to consider wires. Many formulae have been proposed for this purpose [6, 9], and we follow [9]. The largest MCNC building-block benchmark is successfully placed by simulated annealing in about 30 minutes(Fig.11).

This paper is organized as follows. In Section 2, a mapping from a given packing to a pair of module name sequences is given, to show that an optimal solution is included. Section 3 provides a procedure of an inverse mapping from a sequence pair to a packing. Section 4 demonstrates how the space can be utilized in placement problems. Section 5 is for concluding remarks.

## 2. From A Packing to A Sequence-Pair

Let  $\Pi$  be a packing on chip  $C$ . See Fig.1 for an example.

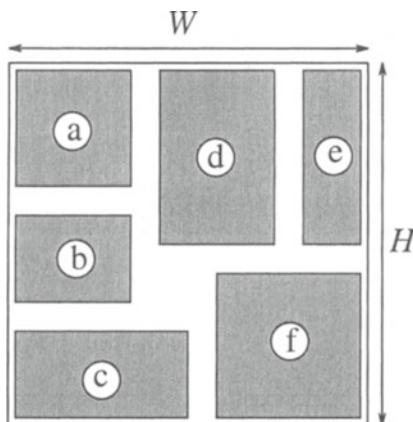


Figure 1. A packing in a chip whose area is  $H \times W$ .

## 2.1 Gridding

A *floorplan* is a partition of  $C$  into rectangles, called *rooms*, such that a room contains at most one module. A room which contains no module is said to be *empty*.

The line segments forming the room boundaries (including four sides of  $C$ ) are called the *cutting-segs*. We assume that a cutting-seg, except for four sides of  $C$ , terminates at a midpoint of an orthogonal cutting-seg (forming a T-intersection). It is trivial that such a floorplan is always possible, although not necessarily unique.

In the following, we describe a procedure to get a pair of module name sequences from a packing.

### procedure: Gridding( $\Pi$ )

Obtain one arbitrary floorplan and fix it. (See Fig.2 which is an example floorplan corresponding to  $\Pi$  in Fig.1.) Take a non-empty room. Put a pebble  $p$  at the center of the room. Move it rightward until it hits the cutting-seg which is the right side of the room. Then, move  $p$  upward until it hits an orthogonal cutting-seg. Then, move it rightward until it hits an orthogonal cutting-seg, and continue turning its direction as right, up, right, up,  $\dots$ , until reaching the upper right corner of the chip. The locus of pebble  $p$  is called the *right-up locus* of the module. Similarly, *up-left locus*, *left-down locus*, and *down-right locus* are defined. (Fig.3 shows these four loci of one module.)

The union of right-up locus of module  $x$  and left-down locus of module  $x$  is called the *positive locus* (since it passes inside the 1st and 3rd quadrants). Analogously, the union of the up-left locus of  $x$  and down-right locus of  $x$  is called the *negative locus*. For every module, one positive locus and one negative locus are uniquely defined. They are referred to by the corresponding module names. (An example with all loci is shown in Fig.4.)

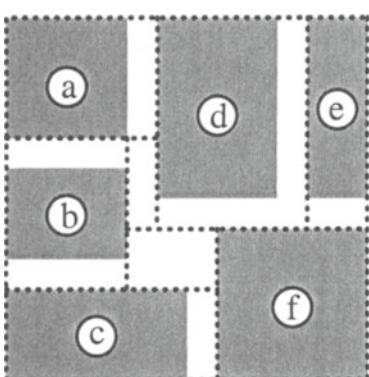


Figure 2. A floorplan of a packing.

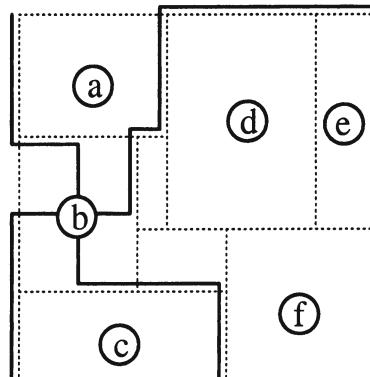


Figure 3. Loci of module  $b$ .

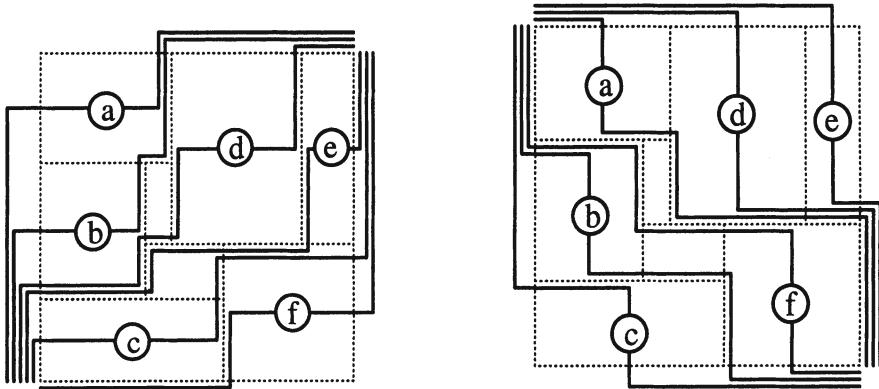


Figure 4. Positive loci(left) and negative loci(right), resulted in  $(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$ .

**Theorem 1** *No pair of positive loci cross each other. No pair of negative loci cross each other. (They may run along the same cutting-segs, but not cross each other.)*

**Proof:** Let two modules be  $a$  and  $b$ . Since positive loci of  $a$  and  $b$  cannot be inside the other room, a crossing, if any, would occur outside their rooms. Denote the right-up locus of module  $a$  by  $RU(a)$ . Similar notation is applied for the other three types of loci.

**Case 1:** Suppose that  $RU(b)$  comes from below and hits  $RU(a)$  at a point  $p_1$ . See Fig.5(case 1). Since  $RU(a)$  and  $RU(b)$  are along cutting-segs,  $RU(b)$  cannot cross  $RU(a)$  at  $p_1$  by definition of the cutting-seg. After  $p_1$ , the two may run along together for a while. Since they are following the same rule of right-up locus, they run together and never cross each other. Hence, right-up loci of  $a$  and  $b$  do not cross. By the same reason, left-down loci of  $a$  and  $b$  do not cross.

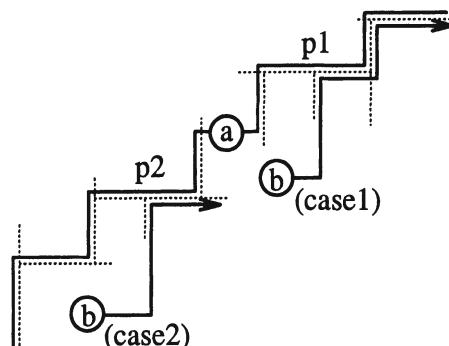


Figure 5. Loci used in the proof of Theorem 1.

*Case 2: Suppose that RU(b) comes from below and hits LD(a) at a point  $p_2$ . See Fig.5 (case 2). After  $p_2$ , RU(b) goes right upstream along LD(a) for a while. Then RU(b) reaches to the point where LD(a) comes from above. After that point, RU(b) continues to go right and thus goes below LD(a) again. Since RU(b) can not go inside the room of a, it goes below the room of a. Hence, left-down locus of a and right-up locus of b do not cross. By the same reason, right-up locus of a and left-down locus of b do not cross.*

*Then, the positive loci of a and b do not cross. Similarly, negative loci of a and b do not cross.*

The implication of the theorem is significant: all the  $m$  positive loci are linearly ordered, and so are the negative loci. Here we order the positive loci from the upper left, and order the negative loci from the lower left. Since each locus is uniquely referred to by the module name, we have obtained an ordered pair of module name sequences  $(\Gamma_+, \Gamma_-)$ , which we call the *Sequence-Pair*, where  $\Gamma_+$  (resp.  $\Gamma_-$ ) is a module name sequence which represents the order of positive (resp. negative) loci.

In Fig.4, positive loci are in order “*abdecf*” and negative loci are in order “*cbfade*”, then  $(\Gamma_+, \Gamma_-) = (\text{abdecf}, \text{cbfade})$  is obtained.

Given packing  $\Pi$ , the corresponding Sequence-Pair is not unique due to the arbitrariness in the fixing the floorplan. Let the one obtained by the procedure be  $(\Gamma_+, \Gamma_-)$ , which we denote as  $\text{Gridding}(\Pi)$ .

## 2.2 Geometrical Information of Sequence-Pair

Let  $\text{Gridding}(\Pi) = (\Gamma_+, \Gamma_-)$ . For a module  $x$ , any other module  $x'$  is uniquely one of four cases,  $x'$  is after/before  $x$  in  $\Gamma_+/\Gamma_-$ . Let us define four classes, accordingly.

$$\begin{aligned}\mathcal{M}^{aa}(x) &= \{x' \mid x' \text{ is after } x \text{ in both } \Gamma_+ \text{ and } \Gamma_-\}, \\ \mathcal{M}^{bb}(x) &= \{x' \mid x' \text{ is before } x \text{ in both } \Gamma_+ \text{ and } \Gamma_-\}, \\ \mathcal{M}^{ba}(x) &= \{x' \mid x' \text{ is before } x \text{ in } \Gamma_+ \text{ and after } x \text{ in } \Gamma_-\}, \\ \mathcal{M}^{ab}(x) &= \{x' \mid x' \text{ is after } x \text{ in } \Gamma_+ \text{ and before } x \text{ in } \Gamma_-\}.\end{aligned}$$

For example, with respect to  $(\Gamma_+, \Gamma_-) = (\text{abdecf}, \text{cbfade})$ , four subsets for module  $b$  are:  $\mathcal{M}^{aa}(b) = \{d, e, f\}$ ,  $\mathcal{M}^{bb}(b) = \emptyset$ ,  $\mathcal{M}^{ba}(b) = \{a\}$ , and  $\mathcal{M}^{ab}(b) = \{c\}$ . Any module other than  $x$  belongs to a unique subset, and it is trivial that two modules are in a dual relation through  $x \leftrightarrow x'$ , and  $a \leftrightarrow b$  as:

$$\begin{aligned}x' \in \mathcal{M}^{aa}(x) &\Leftrightarrow x \in \mathcal{M}^{bb}(x') \\ x' \in \mathcal{M}^{ba}(x) &\Leftrightarrow x \in \mathcal{M}^{ab}(x')\end{aligned}$$

Let us formally define the terminology “left-of”, “right-of”, “above” and “below”. In a packing, if the left side of module  $x$  is right-of the right side of module

$x'$ ,  $x$  is said to be *right-of*  $x'$ . Similarly, *left-of*, *above*, *below* relations between two modules are defined. These notations follow [9].

**Theorem 2** *Let  $\text{Gridding}(\Pi) = (\Gamma_+, \Gamma_-)$ . If  $x' \in \mathcal{M}^{aa}(x)$ , then  $x'$  is right-of  $x$  in  $\Pi$ .*

*The claim holds replacing the pair of words (“ $\mathcal{M}^{aa}$ ” and “right-of”) with any of (“ $\mathcal{M}^{bb}$ ” and “left-of”), (“ $\mathcal{M}^{ba}$ ” and “above”), and (“ $\mathcal{M}^{ab}$ ” and “below”).*

*An example is shown. In Fig.3, modules  $d, e, f$  are in  $\mathcal{M}^{aa}(b)$ , and they are right-of  $b$  in  $\Pi$ .*

**Proof:** We sketch the proof taking an example of Fig.3. Pick arbitrary two modules,  $b$  and  $f$ . The loci of  $b$  divide the chip into four regions. Among them, the region surrounded by the right-up locus of  $b$ , down-right locus of  $b$ , and the right side of the chip is called the right-cone of  $b$ . Similarly, the left-, above-, and below-cone denote other three regions.

Suppose  $f$  is in  $\mathcal{M}^{aa}(b)$ . This implies that the positive locus of  $f$  is in the union of the right-cone and below-cone of  $b$ . Also it is implied that the negative locus of  $f$  is in the union of the right-cone and above-cone of  $b$ . The cross point of the positive and negative locus of  $f$  is in their intersection, that is, in the right-cone of  $b$ . Then, in the floorplan generated in  $\text{Gridding}(\Pi)$ , the center point of the room of  $f$  is in the right-cone of  $b$ . Hence the module  $f$  is also in the right-cone of  $b$ . All the modules in the right-cone of  $b$  must be right-of module  $b$  by definition of right-up locus and down-right locus of  $b$ .

Similarly, the claim holds for the other cases.

### 3. From A Sequence-Pair to A Packing

In the previous section, we analyzed the packing, and fixed the way “gridding” to get one sequence-pair from a given packing. Now we synthesize one packing from an arbitrary sequence-pair.

#### 3.1 $(\Gamma_+, \Gamma_-)$ -Packing

Let  $(\Gamma_+, \Gamma_-)$  be an arbitrary sequence-pair. We define a geometrical constraint derived from  $(\Gamma_+, \Gamma_-)$ .

##### **GeomConst** $(\Gamma_+, \Gamma_-)$

For every two modules  $x$  and  $x'$ ,  $x'$  must be right-of  $x$  in  $\Pi$  if  $x' \in \mathcal{M}^{aa}(x)$ . This is also the constraint with replacing the pair of words (“ $\mathcal{M}^{aa}$ ” and “right-of”) with any of (“ $\mathcal{M}^{bb}$ ” and “left-of”), (“ $\mathcal{M}^{ba}$ ” and “above”), and (“ $\mathcal{M}^{ab}$ ” and “below”).

A packing  $\Pi$  is called  $(\Gamma_+, \Gamma_-)$ -packing if  $\Pi$  satisfies **GeomConst** $(\Gamma_+, \Gamma_-)$ .

**Corollary 1 :** There is a  $(\Gamma_+, \Gamma_-)$ -packing if  $(\Gamma_+, \Gamma_-)$  is the sequence-pair obtained by **Gridding**.  $\square$

However, we can prove the following fact.

**Theorem 3** *For every sequence-pair  $(\Gamma_+, \Gamma_-)$ , there is a  $(\Gamma_+, \Gamma_-)$ -packing.*

**Proof:** Consider an  $m \times m$  grid where  $m$  is the number of modules. Label the horizontal grid lines and vertical grid lines with module names along  $\Gamma_+$  and  $\Gamma_-$  from top and from left in order, respectively. A cross point of the horizontal grid line of label  $x$  and the vertical grid line of label  $x'$  is referred to by  $(x, x')$ . Then, draw the resultant grid on a plane rotating 45 degrees. (See Fig.6.) Put each module  $x$  with its center being on  $(x, x)$ . Expand the separation of grid lines enough to eliminate overlapping of modules. (Actually, the expansion is enough if it is  $\sqrt{2}$  times larger than the longest width/height over modules.) Trivially, the resultant packing satisfies the requirement implied by the given sequence-pair.

An example is shown in Fig.6 which corresponds to  $(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$ .

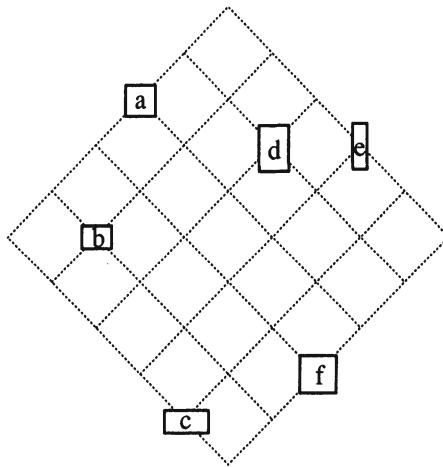


Figure 6. A  $(\Gamma_+, \Gamma_-)$ -packing  
 $(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$ .

### 3.2 $(\Gamma_+, \Gamma_-)$ -Optimal Packing

Given  $(\Gamma_+, \Gamma_-)$ , an optimal packing of  $(\Gamma_+, \Gamma_-)$ -packings is said  $(\Gamma_+, \Gamma_-)$ -optimal. A  $(\Gamma_+, \Gamma_-)$ -optimal packing can be obtained in  $O(m^2)$  time by applying the well-known *longest path algorithm* for vertex weighted directed acyclic graphs. The graphs can be made as follows.

Based on the “right-of” constraint of  $(\Gamma_+, \Gamma_-)$ , a directed and vertex-weighted graph, called *horizontal-constraint graph*  $G_H(V, E)$ , is constructed as follows.

$V$  : source  $s$ , sink  $t$ , and  $m$  vertices labeled with module names

$E : (s, x)$  and  $(x, t)$  for each module  $x$ , and  $(x, x')$  iff  $x$  and  $x'$  are in the constraint “ $x$  must be left-of  $x'$ ”.

Vertex-weight : zero for  $s$  and  $t$ , width of module  $x$  for the other vertices

Similarly the *vertical-constraint graph*  $G_V(V, E)$  is constructed using “below” constraint and the height of each module.

Both graphs do not contain any directed cycle. Furthermore, for every pair of modules, there is always an edge in  $G_H$  or in  $G_V$ , and not in both. This is because the order relation of any two modules in a Sequence-Pair uniquely defines one of the horizontal or vertical relation between them. From this fact, the X-coordinate and the Y-coordinate of the lower left corner of each module can be determined independently to satisfy the constraint in the direction, and the resultant placement is guaranteed not to contain any overlap. Then, the X and Y coordinates of each module are determined as the minimum by assigning the longest path lengths between the source and the node of the module in  $G_H$  and  $G_V$ , respectively. Similarly, the width and the height of the chip are determined as the longest path lengths between the source and the sink in  $G_H$  and  $G_V$ , respectively. Since the width and the height of the chip is independently minimum, the resultant packing is  $(\Gamma_+, \Gamma_-)$ -optimal. The longest path calculation can be done in  $O(m^2)$  time, proportional to the number of edges in the graph.

As an example,  $G_H$  and  $G_V$  are shown in Fig.7 for  $(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$ . The resultant placement after longest path length calculation is shown in Fig.8.

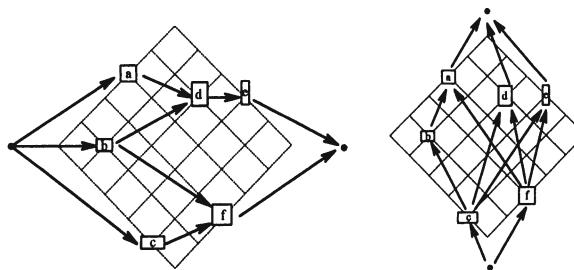


Figure 7. Constraint graphs  $G_H$ (left) and  $G_V$ (right) (transitive edges are not drawn for simplicity).

### 3.3 The P-admissible Solution Space

Previous discussions conclude:

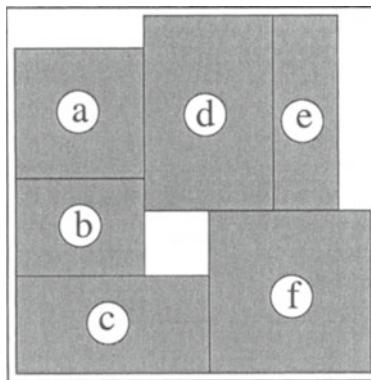


Figure 8. A  $(\Gamma_+, \Gamma_-)$ -optimal packing for  $(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$ .

**Theorem 4** *The set of all sequence-pairs is a P-admissible solution space of RP. More precisely, it consists of  $(m!)^2$  sequence-pairs, each of which can be mapped to one specific  $(\Gamma_+, \Gamma_-)$ -optimal packing in  $O(m^2)$  time. And at least one of these packings is an optimal solution of RP.*

Our discussion started for minimizing the area of the chip. However, all the theorems except Theorem 4 do not mention about the evaluation. While, Theorem 4 holds for any evaluating function as long as it is independently non-decreasing both for the width and height of the chip. Therefore we may assume instead, for example, the perimeter of the chip, area of chip of the pre-specified aspect ratio, and height of the chip when width of the chip is fixed. This fact will expand the applicability of our solution space.

It has been assumed that the orientation of each module (vertically laid or horizontally laid) is fixed. When the orientation is also requested to be optimized, we hold a  $\{0, 1\}$  sequence of length  $m$ , expressing the orientation of each module being horizontal or vertical. The solution space is enlarged to the size of  $(m!)^2 2^m$ . (Even the orientation optimization for a fixed floorplan is also known to be NP-hard [10].) This technique can be easily extended for so called “soft” modules, by preparing three or more candidates of (width, height) pairs for one module [11].

#### 4. Experiments

To show the usefulness of the proposed solution space, we first show its potential in packing. Then, an application example in VLSI layout will be given.

## 4.1 Rectangle Packing

We extracted the dimensions of 146 modules from a printed circuit board in an industry example. These modules are packed by a simulated annealing method by the move (transformation, perturbation) of a solution based on three operations of pair-interchanges of: (i) two module names in  $\Gamma_+$ , (ii) two module names both in  $\Gamma_+$  and  $\Gamma_-$ , and (iii) the width and the height of a module, where the last one is for orientation optimization. The initial sequence-pair is given by assuming  $\Gamma_+ = \Gamma_-$ , which corresponds to a one-horizontal-row packing. The temperature is decreased following a quite standard annealing schedule but from a heuristic point of view, operation (i) is selected with higher probability in higher temperature, and operation (iii) is selected with higher probability in lower temperature.

The result is shown in Fig.9. Computation on Sun SparcStationII stopped in 29.9 minutes reaching the terminating temperature. The algorithm has searched not greater than 606,192 distinct sequence-pairs out of the solution space of size  $(146!)^2 2^{146} \sim 1.23 \times 10^{552}$ . Notice that, only the search of a fraction about  $4.92 \times 10^{-547}$  of the solution space was enough to obtain Fig.9. As a challenge, we tried 500 pieces, using 18.83 hours to get the result shown in Fig.10.

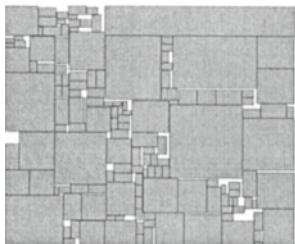


Figure 9. Packing of 146 modules.

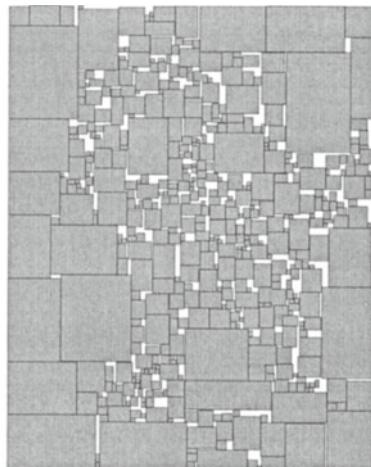


Figure 10. Packing of 500 modules.

## 4.2 Module Placement With Wire Consideration

For VLSI placement problems, we extend the evaluation to consider wires. Among various possible evaluations about wires, here we focus the final chip area. In the following, we demonstrate a method that minimizes the chip area including wiring spaces.

Assume  $(\Gamma_+, \Gamma_-)$  be a sequence-pair,  $\Pi$  be a  $(\Gamma_+, \Gamma_-)$ -optimal packing, and  $W$  and  $H$  be the width and height of  $\Pi$ . Terminals are given as fixed points on the boundaries of each module. A *net* is a set of terminals(multi-terminal net), which must be connected by wires, later in detailed routing phase. A set of nets is given as the netlist  $\mathcal{N}$ . For net  $i$ , the width and the height of the smallest bounding box of the terminals are denoted  $W_i$  and  $H_i$ , respectively.  $T$  is the sum of wire width and spaces between wires. We use the following formulae to estimate the final chip width  $W'$  and height  $H'$ , which are the ones proposed in [9].

$$\begin{aligned} W' &= W + T \frac{\sum_{i \in \mathcal{N}} H_i}{H} \\ H' &= H + T \frac{\sum_{i \in \mathcal{N}} W_i}{W} \end{aligned}$$

The second term in the right-hand side of each formula estimates the amount of increase in one direction owing to the wires, assuming all wires are uniformly distributed in the final chip. They experimentally showed that the resultant placement is acceptable for the later detailed routing stage.

As for the variety in self-symmetric placement of each module, there are totally eight choices per a module, which is the combination of four choices of  $\{0, 90, 180, 270\}$  degree rotations, and two choises of {yes,no} value of the reflection about Y axis. In our system, this code for orientation and a sequence-pair are put together into a simulated annealing process, which runs in a similar fashion as rectangle packing optimization. The process searches the solution space of size  $(m!)^2 8^m$ .

The point which is not mentioned in [9] is how the location of each individual module is calculated. After the best evaluated code is obtained, coordinates of each module is determined as follows. Assume  $(X_j, Y_j)$  is the coordinates of the lower left corner of module  $j$  in  $\Pi$ , which is the information we can use in this phase. Let  $\mathcal{N}_{X_j}$  be a set of nets such that the X coordinate of left side of bounding box of the net is less than or equal to  $X_j$ . Similarly,  $\mathcal{N}_{Y_j}$  is defined using  $Y_j$ . We determine the coordinates  $(X'_j, Y'_j)$  of the lower left corner of module  $j$  in the resultant chip by the following formulae.

$$\begin{aligned} X'_j &= X_j + T \frac{\sum_{i \in \mathcal{N}_{X_j}} H_i}{H} \\ Y'_j &= Y_j + T \frac{\sum_{i \in \mathcal{N}_{Y_j}} W_i}{W} \end{aligned}$$

The biggest building block layout data, called “ami49”, is taken from the MCNC benchmarks, and placed by the above described method with additional

constraint: aspect ratio = 1. For authors' knowledge, it has not been dealt without hierarchically dividing the problem to reduce the problem size. The result is shown in Fig.11. Computation time was 31.36 minutes on SunIPX.

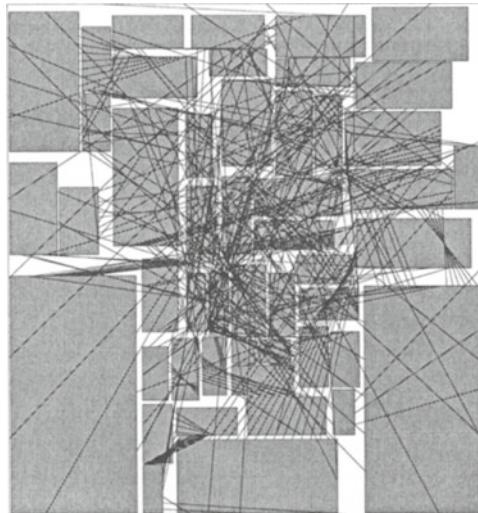


Figure 11. Placement of MCNC “ami49”.

## 5. Concluding Remarks

The motivation of this work was our experience that many VLSI designers are not satisfied with slicing structures. This paper has achieved a breakthrough by introducing a P-admissible solution space to the rectangle packing problem, which is fundamental to the layout design.

Experiments suggest that hundreds of rectangles can be packed very effectively in reasonable time. The biggest MCNC benchmark data, ami49, is now tractable without hierarchically dividing the problem, by utilizing the proposed solution space.

As experiments revealed, the search may cover only a fraction of the whole solution space. Though the results are of the quality high enough for practical use, the space is too vast. An interesting open question from a theoretical point of view would be about reducing the size of the space. However, we conjecture that the size of the space can not be decreased drastically.

## Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, and Culture of Japan (05452209) and Research Body CAD21 at JAIST.

## References

- [1] B.S.Baker, E.G.Coffman, and R.L.Rivest, "Orthogonal Packings in Two Dimensions," *SIAM J. Comput.*, vol. 9, no. 4, pp. 846–855, 1980.
- [2] L.Sha and R.W.Dutton, "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks," in *Proc. 22th ACM/IEEE Design Automation Conf.*, pp. 602–608, 1985.
- [3] A.Alon and U.Ascher, "Model and Solution Strategy for Placement of Rectangular Blocks in the Euclidean Plane," *IEEE Trans. on CAD*, vol. 7, no. 3, pp. 378–386, 1988.
- [4] Y.G.Saab and V.B.Rao, "Combinatorial Optimization by Stochastic Evolution," *IEEE Trans. on CAD*, vol. CAD-10, no. 4, pp. 525–535, 1991.
- [5] R.H.J.M.Otten, "Automatic Floorplan Design," in *Proc. 19th ACM/IEEE Design Automation Conf.*, pp. 261–267, 1982.
- [6] D.F.Wong and C.L.Liu, "A New Algorithm for Floorplan Designs," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, pp. 101–107, 1986.
- [7] W.M.Dai and E.Kuh, "Simaltaneous Floorplanning and Global Routing for Hierarchical Building Block Layout," *IEEE Trans. on CAD*, vol. CAD-6, no. 5, pp. 828–837, 1987.
- [8] T.C.Wang and D.F.Wong, "An Optimal Algorithm for Floorplan Area Optimization," in *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 180–186, 1990.
- [9] H.Onodera, Y.Taniguchi, and K.Tamaru, "Branch-and-Bound Placement for Building Block Layout," in *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 433–439, 1991.
- [10] L.Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control*, vol. 59, pp. 91–101, 1983.
- [11] P.Pan, W.Shi, and C.L.Liu, "Area Minimization for Hierarchical Floorplans," in *IEEE International Conf. on Computer Aided Design*, pp. 436–440, 1994.

## Part VII

# TIMING, TEST AND MANUFACTURING

Timing, Test and Manufacturing Overview <i>Karem A. Sakallah, Duncan M. (Hank) Walker and Sani R. Nassif</i>	551
A Methodology for Worst Case Design of Integrated Circuits (ICCAD 1983) <i>A. J. Strojwas, S. R. Nassif and S. W. Director</i>	563
Timing Analysis using Functional Relationships (ICCAD 1986) <i>Daniel Brand and Vijay S. Iyengar</i>	567
On the Design of Robust Multiple Fault Testable CMOS Combinational Logic Circuits (ICCAD 1988) <i>Sandip Kundu, Sudhakar M. Reddy and Niraj K. Jha</i>	575
Circuit Optimization Driven by Worst-Case Distances (ICCAD 1991) <i>Kurt J. Antreich and Helmut E. Graeb</i>	585
Verifying Clock Schedules (ICCAD 1992) <i>Thomas G. Szymanski and Narendra Shenoy</i>	597
Efficient Implementation of Retiming (ICCAD 1994) <i>Narendra Shenoy and Richard Rudell</i>	615

# TIMING, TEST AND MANUFACTURING OVERVIEW

Karem A. Sakallah

*Electrical Engineering and Computer Science Department*

*University of Michigan*

*Ann Arbor, MI*

Duncan M. (Hank) Walker

*Department of Computer Science*

*Texas A&M University*

*College Station, TX*

Sani R. Nassif

*IBM Research*

*Austin, TX*

## 1. Timing

### 1.1 Introduction

Timing analysis is concerned with estimating and optimizing the performance of integrated circuits. It encompasses a wide range of activities including physical modeling of transistors and interconnect wires, derivation of analytical and empirical gate and wire delay models, accurate estimation of long- and short-path delays through combinational logic, detection of setup and hold violations in sequential circuits, as well as a variety of combinational and sequential circuit transformations aimed at maximizing operation speed. The three papers reviewed here represent particularly significant contributions to the field of timing analysis and optimization: Brand and Iyengar [4] built the foundation for the field of false-path analysis; Szymanski and Shenoy [33] had the last word on the field of timing verification of latch-based circuits; and Shenoy and Rudell [29] are credited with making retiming viable for industrial-sized circuits.

### 1.2 Functional Timing Analysis

The origin of modern timing analysis algorithms can be traced back to the work of Kirkpatrick and Clark [17] who showed how the PERT (Project Evaluation and Review Technique) method of Operations Research can be adapted to compute the delay and identify the critical paths of combinational logic circuits. While the approach was fundamentally premised on ignoring the func-

tionality of a circuit's logic gates, it is interesting to note that Kirkpatrick and Clark recognized the need to account for some functionality (different rise and fall delays, and distinction between AND and OR gates) in order to reduce the inherent pessimism in a function-less analysis. Despite the appeal of this approach, its realization in timing analysis programs did not come quickly. One of the earliest reported implementations of these ideas is that of Hitchcock, Smith, and Cheng [11] ; they developed the Timing Analysis program and used it to detect timing bugs in the Processor Unit of the IBM 3081.

Brand and Iyengar are credited with being the first to demonstrate the need for a fuller accounting of functional relationships during "block-oriented" timing analysis. Their succinct 1986 ICCAD paper [4] (and its later journal version [5]) laid the foundation for the field of "false path analysis" and led, over the following decade and a half, to a proliferation of publications by many authors on various aspects of this topic [10, 21, 27, 2, 9, 7, 6, 30, 31, 36]. The impact that their contribution had can be attributed to two factors. First, they proposed an efficient procedure, based on their earlier work on logic synthesis [3], for collecting and checking for consistency the conditions for signal propagation along a circuit's paths. This showed that such a functional analysis, which theoretically increases the computational effort from linear to exponential, is still quite feasible in many cases. And second, they empirically demonstrated the existence of false paths (which they dubbed "non-functional" paths) in a variety of benchmark circuits; furthermore, they showed that ignoring such paths in timing analysis can lead to significant over-estimations of circuit delay.

### 1.3 Modeling and Verification of Clock Schedules

Ensuring proper operation of synchronous sequential circuits amounts to checking that the applied clock schedule does not lead to hold violations (due to short signal paths) or setup violations (due to long signal paths) at the circuit's state devices. For circuits that employ edge-triggered flip-flops, such checks are localized to the combinational logic between the flip-flops and can be carried out quite efficiently. The checks become significantly more involved, however, for circuits that use level-sensitive latches because such latches allow signals to "flow through" when the latches are enabled. The search for correct and efficient timing verification procedures of latch-based circuits began in the early eighties when level-sensitive latches started to replace the larger and slower edge-triggered flip-flops as the state device of choice in integrated circuits.

While the "latch problem" was recognized in the early timing verifiers of McWilliams [22] and Agrawal [1], it was Jouppi [15, 16] who pointed out that the flow-through property of latches allowed a latch-based circuit to be operated at a higher clock frequency than that predicted by the maximum combinational delay between latches. He referred to this as "time borrowing" and presented a

slack propagation procedure to implement it. At about the same time, Ousterhout [23] observed that ad-hoc solutions for the timing verification problem of latch-based circuits that employ multi-phase clocking could result in undetected timing errors and suggested that it is “the single greatest problem yet to be solved in timing verification.” From the mid-to-late eighties several authors proposed models and algorithms to solve this problem, most notably Unger and Tan [35], Szymanski [32], and Dagenais [8].

The first comprehensive mathematical model for latch-based multi-phase synchronous circuits was proposed by Sakallah, Mudge and Olukotun [28]. They introduced the notion of a “phase shift” operator that resulted in a significant simplification of the signal propagation equations between latches controlled by different clock phases. They also proposed and implemented algorithms for timing verification ( $\text{checkTc}$ ) and clock schedule optimization ( $\text{minTc}$ ). The significance of this development was quickly recognized by Szymanski and Shenoy who dubbed this the SMO model and proceeded to propose more efficient implementations of the  $\text{minTc}$  [34] and  $\text{checkTc}$  [33] algorithms. In particular, their paper on verifying clock schedules [33] provided a thorough theoretical analysis of the SMO model and highlighted several subtleties (such as solution non-uniqueness and the need for careful initialization) in the  $\text{checkTc}$  verification algorithm that could lead to incorrect analysis results or to non-convergence. They also proved that the (correctly-implemented) verification algorithm can check an n-latch circuit in time in the worst case; practically, the algorithm runs in almost linear time for typical circuits with sparse connections between latches.

## 1.4 Efficient Retiming

Retiming transformations were first proposed by Leiserson and Saxe in 1983 [19]. Using a simple model of sequential logic circuits, they presented several polynomial-time algorithms that can transform an initial circuit to an equivalent faster circuit. This was achieved by a sequence of register moves across the combinational logic that sought to balance delays between register stages. Many additional enhancements and extensions to this basic algorithm were proposed since its initial introduction. These include, among many others, retiming edge-triggered circuits under realistic delay models [18], retiming while accounting for hold constraints [26], and retiming of level-clocked circuits [20, 24, 13, 14].

The paper by Shenoy and Rudell [29] observed that polynomial-time algorithms are too expensive in practice and suggested several algorithmic enhancements that enabled retiming to be viable for industrial-sized circuits. Specifically, they proposed an early-termination condition for the procedure that checks if a given clock period is a feasible solution to a given retiming of the circuit. This condition is based on an efficient scheme for identifying negative-weight

cycles in a graph (similar to that used by Szymanski for computing optimal clock schedules [34]), and typically leads to a drastic reduction in runtime. Shenoy and Rudell also addressed the scalability of retiming by proposing and implementing an algorithm for minimum-area retiming whose worst-case complexity is worse than Leiserson and Saxe's original algorithm, but whose memory complexity is linear instead of quadratic.

## 2. Test

### 2.1 Introduction

Integrated circuit test is concerned with the problem of verifying that a manufactured integrated circuit meets its specifications and will operate reliably in a system. The most common specifications that are tested are function and performance over the operating voltage and temperature range. Two major long-term thrusts in test research are making chip designs easier to test, and relying increasingly on automatically-generated structural tests to screen for function, performance and reliability. Since chip complexity is increasing faster than pin count, design-for-test (DFT) hardware has had to be placed on chip to provide the necessary controllability and observability to achieve high test coverage of manufacturing defects. At the same time, falling transistor cost and rising performance has made it necessary and economically attractive to perform more of the testing with on-chip hardware, such as built-in self-test (BIST) and embedded deterministic test. The general problem is now one of test resource partitioning, that is, dividing up the test resources between the chip and the automatic test equipment (ATE), and test synthesis, that is, automatically synthesizing the on-chip test hardware.

As chip complexity has increased, it has become too expensive to rely solely on manually written functional tests, particularly for ASICs. Increasingly, automatic test pattern generation (ATPG) is performed for digital circuits, using knowledge of circuit structure and the defects likely to occur in manufacturing. The many possible defect behaviors and locations are abstracted to logical fault models. The increasing performance of integrated circuits has shifted the focus of structural test for digital circuits to delay test, that is, testing for defects that cause a circuit to be too slow.

These two long-term test research thrusts come together in the paper selected for the Test section. It is commonly the case that ATPG cannot achieve 100% coverage of all the targeted circuit faults. In their ICCAD88 paper [37] and their follow-on journal article [38], Kundu, Reddy and Jha showed that for CMOS combinational logic circuits, such limitations are due to the circuit structure, rather than the logic function. They described an algorithm to synthesize combinational logic functions so as to be robustly testable for multiple occurrences of stuck-at, stuck-open and path delay fault models. This paper sparked sev-

eral years of research on synthesis for testability, much of which appeared at ICCAD.

## 2.2 Synthesis for Testability

There is a large body of prior research on synthesizing circuits to increase their testability [39]. Much of this research has focused on enhancing stuck-at fault testability in combinational and sequential logic blocks, much of which appeared at ICCAD [40, 41, 42, 43]. There has also been extensive research on synthesis for path delay testability [44, 45, 46] and random pattern testability [47], with some important papers on these topics at ICCAD [48, 49]. If testability is not considered during timing optimization, it can make path delay test more difficult [50]. The synthesis method of Kundu, Reddy and Jha [37] produced circuits that are testable under multiple faults for both stuck and path delay fault models. The primary challenge of their approach is the restriction to unate circuits. For many functions, forming unate circuits results in an unacceptable increase in circuit size. The ICCAD papers that followed were able to achieve testability without this restriction. In today's large scale designs, testability synthesis research has shifted to focus on high-level design descriptions [51, 52, 53, 54], including work appearing at ICCAD [55, 56].

Industrial experience on large designs is that achieving high stuck-at fault coverage is a struggle, even with extensive DFT features. This is due to embedded arrays, buses, and structures that cannot have test circuitry included, and the difficulties they cause for ATPG. Most current delay test synthesis research is focused on improving test access via scan chains [58, 59], building upon the pioneering work at IBM [57], and applying the two-pattern tests via low-overhead test logic and constrained tester resources [60]. System-on-a-chip (SoC) designs present additional challenges and standardization requirements in providing test access to on-chip modules which internally have their own test methods [61, 62, 63, 64], but little of this has appeared at ICCAD.

## 2.3 Automatic Test Pattern Generation

The primary industrial impact of test research has been the development of automatic test pattern generation (ATPG) tools. The primary enabler was the wide adoption of scan chains. Most ATPG research appeared in other forums, but ICCAD contributed with pioneering research on targeting realistic

faults [65], improved search algorithms [66], and IDDQ testing [67]. Even “full” scan circuits contain sequential logic, and ICCAD served as a forum for some early sequential circuit ATPG research [68, 69].

As circuit performance increases, delay fault ATPG becomes increasingly important. There are two widely used delay fault models: transition and path delay. The transition fault model assumes that a slow transition on a single line is so large that any path through it will be slow. Smaller delay defects can be detected by testing the longest paths through each gate, with an early paper on this topic at ICCAD [71]. The path delay fault model assumes that a path may be slow, and targets distributed delay due to manufacturing process variations. Robust path delay tests are valid even in the presence of arbitrary circuit delays, while nonrobust tests assume a single slow path. Combinational CMOS circuits synthesized with Kundu, Reddy and Jha’s technique [37] are robustly testable. A large number of path delay fault ATPG tools have been developed [72, 73, 74], with many important papers appearing at ICCAD [75, 76, 77, 78]. Transition fault tests are widely used since they can be generated with a simple modification to a stuck-at ATPG. The traditional fault models are being extended to more directly target defects, termed defect-based test. These include delay variation due to capacitive effects and noise [79, 80], resistive bridges and opens [81, 82], or their combination [83, 84]. In addition, it has been shown that since a fault model cannot include all possible defect behaviors, non-target tests may be better for achieving high quality [85]. Only a small amount of defect-based testing research has appeared at ICCAD [86].

### 3. Manufacturing

#### 3.1 Introduction

Manufacturing research is usually associated with silicon technology development and is the major focus of such conferences as the IEDM. As the interface between the design and manufacturing of integrated circuits grew more complicated, however, there was a need for CAD tools and CAD developers to get involved in order to solve some of the pressing issues that were causing problems. Thus a small group of people, by the standards of the overall CAD community, crusaded for improved design/technology coupling through automation. This section chronicles this segment of CAD, as relating to ICCAD, over the last twenty years.

#### 3.2 Spice and Beyond

The wide availability and popularity of Spice [87] made it the de-facto standard interface between design and manufacturing. Much of the early work focused on methods that would either generate the statistically varying Spice pa-

rameters (e.g. [88, 90] ), or methods that would summarize them in a manner conducive to reducing the number of simulations requires, i.e. the paper selected for this book [89] and the follow-on journal article [92]. and book chapter [91] These early works set the stage for the current pervasive industrial practice of creating *corner cases* to model the inherent variability in the IC manufacturing process.

Throughout the eighties and early nineties, a number of other important papers appeared in ICCAD focusing on variability modeling and on circuit optimization to improve yield, e.g. [95, 97, 98, 99]. The second selected paper in this area, [101], represents the formal integration of worst case analysis and circuit yield maximization. As technology scaling continued and circuits became too large to analyze in detail using Spice, other works appeared, such as [102] where an early attempt at performing statistical timing was presented, and [103] where an early attempt at dealing with within-die variations was proposed.

### 3.3 Defects and Yield

Another lively area of early manufacturing research in ICCAD was that of modeling catastrophic defects (short, opens and other layout deformations). From the early work in [93, 94] to the seminal work [96] and the reporting of one the early CAD tools to deal with this problem [100], ICCAD provided an early forum for the design/manufacturing interface in this practically important area.

In spite of the excellent tutorial [104] in 1996, little has happened in ICCAD since the late nineties. Other conferences, particularly ISQED, have become the target of papers that focus on the design/manufacturing interface. Nevertheless, much of the work originally reported in ICCAD remains in active use today.

### Acknowledgments

We would like to acknowledge Chandramouli Kashyap for his assistance in the preparation of this review.

### References

- [1] V. D. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," *DAC*, pp. 629-635, 1982.
- [2] J. Benkoski et al, "Timing Verification Using Statically Sensitizable Paths," *IEEE Trans. CAD*, vol. 9, no. 10, pp. 1073-1084, 1990.
- [3] D. Brand, "Redundancy and Don't Cares in Logic Synthesis," *IEEE Trans. Computers*, vol. C-32, pp. 947-952, 1983.
- [4] D. Brand and V. Iyengar, "Timing Analysis Using Functional Relationships," *ICCAD*, pp. 126-129, 1986.
- [5] D. Brand and V. S. Iyengar, "Timing Analysis Using Functional Analysis," *IEEE Trans. Computers*, vol. 37, pp. 1309-1314, 1988.

- [6] H. Chang and J. A. Abraham, "VIPER: An Efficient Vigorously Sensitizable Path Extractor," *DAC*, pp. 112-117, 1993.
- [7] H. Chen and D. H.-C. Du, "Path Sensitization in Critical Path Problem," *IEEE Trans. CAD*, vol. 12, no. 2, pp. 196-207, 1993.
- [8] M. R. Dagenais and N. C. Rumin, "On the Calculation of Optimal Clocking Parameters in Synchronous Circuits with Level-Sensitive Latches," *IEEE Trans. CAD*, vol. 8, no. 3, pp. 268-278, March, 1989.
- [9] S. Devadas, K. Keutzer and S. Malik, "Delay Computation in Combinational Logic Circuits: Theory and Algorithms," *ICCAD*, pp. 176-179, 1991.
- [10] D. H. Du, H. C. Yen and S. Ghanta, "On the General False Path Problem in Timing Analysis," *DAC*, pp. 555-560, 1989.
- [11] R. B. Hitchcock, et al, "Timing Analysis of Computer Hardware," *IBM Journal of Research and Development*, vol. 26, no. 1, pp. 100-105, 1982.
- [12] V. M. Hrapcenko, "Depth and Delay in a Network," *Soviet Math. Dokl. (Dokl. Akad. Nauk SSSR)*, vol. 19, pp. 1006-1009, 1978.
- [13] A.T. Ishii and M.C. Papaefthymiou, "Efficient Pipelining of Level-Clocked Circuits with Min-Max Propagation Delays," *ACM/IEEE TAU Workshop*, 1995.
- [14] A.T. Ishii, C.E. Leiserson and M.C. Papaefthymiou, "Optimizing Two-Phase, Level-Clocked Circuitry," *Journal of the ACM*, vol. 44, no. 1, pp. 148-199, Jan. 1997.
- [15] N. P. Jouppi, "Timing Analysis for nMOS VLSI," *DAC*, pp. 411-418, 1983.
- [16] N. P. Jouppi, "Timing Analysis and Performance Improvement of MOS VLSI Designs," *IEEE Trans. CAD*, vol. CAD-6, pp. 650-665, 1987.
- [17] T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design," *IBM Journal of Research and Development*, vol. 10, pp. 135-141, 1966.
- [18] K.N. Lalgudi and M.C. Papaefthymiou, "Delay: An Efficient Tool for Retiming with Realistic Delay Modeling," *DAC*, pp. 304-309, 1995.
- [19] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *Journal of VLSI and Computer Systems*, vol. 1, pp. 41-67, 1983.
- [20] B. Lockyear and C. Ebeling, "Optimal Retiming of Multi-Phase Level-Clocked Circuits," *Brown/MIT Advanced Research in VLSI and Parallel Systems Conference*, pp. 265-280, 1992.
- [21] P. C. McGeer and R. K. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," *DAC*, pp. 561-567, 1989.
- [22] T. M. McWilliams, "Verification of Timing Constraints on Large Digital Systems," *DAC*, pp. 139-147, 1980.
- [23] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI," *IEEE Trans. CAD*, vol. CAD-4, pp. 336-349, 1985.
- [24] M.C. Papaefthymiou and K.H. Randall, "TIM: A Timing Package for Two-Phase, Level-Clocked Circuitry," *DAC*, pp. 497-502, 1993.
- [25] M.C. Papaefthymiou, "Understanding Retiming Through Maximum Average-Delay Cycles," *Mathematical Systems Theory*, no. 27, pp. 65-84, 1994.
- [26] M.C. Papaefthymiou, "Asymptotically Efficient Retiming Under Setup and Hold Constraints," *ICCAD*, pp. 396-401, 1998.
- [27] S. Perremans, L. Claesen and H. DeMan, "Static Timing Analysis of Dynamically Sensitizable Paths," *DAC*, pp. 568-573, 1989.

- [28] K. A. Sakallah, T. N. Mudge and O. A. Olukotun, "checkTc and minTc: Timing Verification and Optimal Clocking of Synchronous Digital Circuits," *ICCAD*, pp. 552-555, 1990.
- [29] N. Shenoy and R. Rudell, "Efficient Implementation of Retiming," *ICCAD*, pp. 226-233, 1994.
- [30] J. P. M. Silva and K. A. Sakallah, "An Analysis of Path Sensitization Criteria," *ICCAD*, pp. 68-72, 1993.
- [31] S. Z. Sun, D. H. C. Du and H. C. Chen, "Efficient Timing Analysis for CMOS Circuits Considering Data Dependent Delays," *ICCD*, pp. 156-159, 1994.
- [32] T. G. Szymanski, "LEADOUT: A Static Timing Analyzer for MOS Circuits," *ICCAD*, pp. 130-133, 1986.
- [33] T. G. Szymanski and N. Shenoy, "Verifying Clock Schedules," *ICCAD*, pp. 124-131, 1992.
- [34] T. G. Szymanski, "Computing Optimal Clock Schedules," *DAC*, pp. 399-404, 1992.
- [35] S. Unger and C. J. Tan, "Clocking Schemes for High Speed Digital Systems," *IEEE Trans. Computers*, vol. C-35, no. 10, pp. 880-895, Oct, 1986.
- [36] H. Yalcin et al, "Fast and Accurate Timing Characterization Using Functional Information," *IEEE Trans. CAD*, vol. 20, pp. 315-331, 2001.
- [37] S. Kundu, S. M. Reddy and N. K. Jha, "On The Design of Robust Multiple Fault Testable CMOS Combinational Logic Circuits," *ICCAD*, pp. 240-243, 1988.
- [38] S. Kundu, S. M. Reddy and N. K. Jha, "Design of Robustly Testable Combinational Logic Circuits," *IEEE Trans. CAD*, vol. 10, no. 8, pp. 1036-1048, Aug. 1991.
- [39] R. C. Aitken, "An Overview of Test Synthesis Tools," *IEEE Design & Test of Computers*, vol. 12, no. 2, pp. 8-15, Summer 1995.
- [40] S. Devadas and K. Keutzer, "Boolean Minimization and Algebraic Factorization Procedures for Fully Testable Sequential Machines," *ICCAD*, pp. 208-211, 1989.
- [41] G. Hachtel, R. Jacoby, K. Keutzer and C. Morrison, "On Properties of Algebraic Transformation and the Multifault Testability of Multilevel Logic," *ICCAD*, pp. 422-425, 1989.
- [42] M. J. Bryan, S. Devadas and K. Keutzer, "Testability-Preserving Circuit Transformations," *ICCAD*, pp. 456-459, 1990.
- [43] A. Saldanha, R. K. Brayton, A. L. Sangiovanni-Vincentelli and K. T. Cheng, "Timing Optimization With Testability Considerations," *ICCAD*, pp. 460-463, 1990.
- [44] S. Devadas and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Theory," *IEEE Trans. CAD*, vol. 11, no. 1, pp. 87-101, Jan. 1992.
- [45] S. Devadas and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Practice," *IEEE Trans. CAD*, vol. 11, no. 3, pp. 277-300, Mar. 1992.
- [46] T. J. Chakraborty, V. D. Agrawal and M. L. Bushnell, "Improving Path Delay Testability of Sequential Circuits," *IEEE Trans. VLSI*, vol. 8, no. 6, pp. 736-741, Dec. 2000.
- [47] A. S. Touba and E. J. McCluskey, "Automatic Logic Synthesis of Random Pattern Testable Circuits," *ITC*, pp. 174-183, 1994.
- [48] K. Roy, J. A. Abraham and K. De, "Synthesis of Delay Fault Testable Combinational Logic," *ICCAD*, pp. 418-421, 1989.
- [49] M. Chatterjee, D. K. Pradhan and W. Kunz, "LOT: Logic Optimization With Testability - New Transformations Using Recursive Learning," *ICCAD*, pp. 318-325, 1995.
- [50] T. W. Williams, B. Underwood and M. R. Mercer, "The Interdependence Between Delay Optimization of Synthesized Networks and Testing," *DAC*, pp. 87-92, 1991.

- [51] A. Majumdar, R. Jain and K. Saluja, "Incorporating Testability Considerations in High Level Synthesis," *Kluwer Journal of Electronic Testing, Theory and Applications*, vol. 5, pp. 43-55, 1992.
- [52] M. Potkonjak, S. Dey and R. K. Roy, "Behavioral Synthesis Of Area-Efficient Testable Designs Using Interaction Between Hardware Sharing and Partial Scan," *IEEE Trans. CAD*, vol. 14, no. 9, pp. 1141-1154, Sept. 1995.
- [53] K. Lai, C. Papachristou and M. Baklashov, "BIST Testability Enhancement Using High Level Test Synthesis Technique," *IEEE Asian Test Symp.*, pp. 338-343, 1997.
- [54] F. Fummi, D. Sciuto and M. Serra, "Synthesis For Testability of Highly Complex Controllers By Functional Redundancy Removal," *IEEE Trans. Computers*, vol. 48, no. 12, pp. 1305-1323, Dec. 1999.
- [55] F. F. Hsu, E. M. Rudnick and J. H. Patel, "Enhancing High-Level Control-Flow For Improved Testability," *ICCAD*, pp. 322-328, 1996.
- [56] S. Ravi, G. Lakshminarayana and N. K. Jha, "A Framework for Testing Core-Based Systems-on-a-Chip," *ICCAD*, pp. 385-390, 1999.
- [57] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Journal of Design Automation and Fault-Tolerant Computing*, vol. 2, pp. 165-178, 1978.
- [58] S. Narayanan, R. Gupta and M. Breuer, "Configuring Multiple Scan Chains for Minimum Test Time," *ICCAD*, pp. 4-8, 1992.
- [59] C. C. Lin, M. T. C. Lee, M. Marek-Sadowska and K. C. Chen, "Cost-Free Scan: A Low-Overhead Scan Path Design Methodology," *ICCAD*, pp. 528-533, 1995.
- [60] J. Saxena et al, "Scan-Based Transition Fault Testing - Implementation and Low Cost Test Challenges," *ITC*, pp. 1120-1129, 2000.
- [61] L. Whetsel, "Core Test Connectivity, Communication and Control," *ITC*, pp. 303-312, 1998.
- [62] Y. Zorian, E. J. Marinissen and S. Dey, "Testing Embedded-Core-Based System Chips," *IEEE Computer*, vol. 32, no. 6 , pp. 52-60, June 1999.
- [63] S. Ozev, I. Bayraktaroglu and A. Orailoglu, "Test Synthesis For Mixed-Signal SOC Paths," *DATE*, pp. 128-133, 2000.
- [64] K. Zarrieh, S. J. Upadhyaya and V. Chickermane, "System-On-Chip Testability Using LSSD Scan Structures," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 83-97, May-June 2001.
- [65] P. Nigh and W. Maly, "Layout-Driven Test Generation," *ICCAD*, pp. 154-157, 1989.
- [66] J. P. M. Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability," *ICCAD*, pp. 220-227, 1996.
- [67] W. Mao, R. K. Gulati, D. K. Goel and M. D. Celetti, "QUIETEST: A Quiescent Current Testing Methodology for Detecting Leakage Faults," *ICCAD*, pp. 280-283, 1990.
- [68] H. K. T. Ma, S. Devadas, A. R. Newton and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Finite State Machines," *ICCAD*, pp. 288-291, 1987.
- [69] T. P. Kelsey and K. K. Saluja, "Fast Test Generation for Sequential Circuits," *ICCAD*, pp. 354-361, 1989.
- [70] A. Ghosh, S. Devadas and A. R. Newton, "Test Generation for Highly Sequential Circuits," *ICCAD*, pp. 362-366, 1989.
- [71] M. Geilert, J. Alt and M. Zimmermann, "On the Efficiency of the Transition Fault Model for Delay Faults," *ICCAD*, pp. 272-275, 1990.

- [72] K. Fuchs, F. Fink and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Trans. CAD*, vol. CAD-10, no. 10, pp. 1323-1335, Oct. 1994.
- [73] I. Pomeranz, S. M. Reddy and P. Uppaluri, "NEST: A Non-Enumerative Test Generation Method for Path Delay Faults in Combinational Circuits," *DAC*, pp. 439-445, 1993.
- [74] K. Fuchs, M. Pabst and T. Roessel, "RESIST: A Recursive Test Pattern Generation Algorithm for Path Delay Faults Considering Various Test Classes," *IEEE Trans. CAD*, vol. 13, no. 12, pp. 1550-1562, Dec. 1994.
- [75] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *ICCAD*, pp. 148-151, 1986.
- [76] S. M. Reddy, C. J. Lin and S. Patil, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults," *ICCAD*, pp. 284-287, 1987.
- [77] S. Patil and S. M. Reddy, "A Test Generation System for Path Delay Faults," *ICCAD*, pp. 40-43, 1989.
- [78] P. C. McGeer et al, "Timing Analysis and Delay-Fault Test Generation Using Path-Recursive Functions," *ICCAD*, pp. 180-183, 1991.
- [79] W. Chen, S. Gupta and M. Breuer, "Test Generation for Crosstalk-Induced Delay in Integrated Circuits," *ITC*, pp. 191-200, 1999.
- [80] A. Krstic, J. J. Liou, Y. M. Jiang and K. T. Cheng, "Delay Testing Considering Crosstalk-Induced Effects," *ITC*, pp. 558-567, 2001.
- [81] V. R. Sar-Dessai and D. M. H. Walker, "Resistive Bridge Fault Modeling, Simulation and Test Generation," *ITC*, pp. 596-605, 1999.
- [82] S. Chakravarty et al, "Experimental Evaluation of Scan Tests for Bridges," *ITC*, pp. 509-518, 2002.
- [83] W. Moore, G. Gronthoud, K. Baker and M. Lousberg, "Delay-Fault Testing and Defects in Sub-Micron ICs: Does Critical Resistance Really Mean Anything?," *ITC*, pp. 95-104, 2000.
- [84] R. Desineni, K. N. Dwarkanath and R. D. Blanton, "Universal Test Set Generation Using Fault Tuples," *ITC*, pp. 812-819, 2000.
- [85] J. Dworak et al, "Defect-Oriented Testing and Defective-Part-Level Prediction," *IEEE Design & Test of Computers*, vol. 18, no. 1, pp. 31-41, Jan.-Feb. 2001.
- [86] M. Sivaraman and A. J. Strojwas, "Delay Fault Coverage: A Realistic Metric and an Estimation Technique for Distributed Path Delay Faults," *ICCAD*, pp. 494-501, 1996.
- [87] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," PhD Thesis, University of California, Berkeley, 1975.
- [88] R. W. Webb and E. W. George, "A Statistical PRocess and Device Simulator (SPADS)," *ICCAD*, pp. 209-210, 1983.
- [89] A. J. Strojwas, S. R. Nassif and S. W. Director, "A Methodology for Worst Case Design of Integrated Circuits," *ICCAD*, pp. 152-153, 1983.
- [90] S. R. Nassif, A. J. Strojwas and S. W. Director, "FABRICS II: A Statistical Simulator for the IC Manufacturing Process," *IEEE Trans. CAD*, vol. CAD-3, no. 1, pp. 40-46, Jan. 1984.
- [91] S. R. Nassif, "Statistical Worst-Case Analysis for Integrated Circuits," in *Advances in CAD for VLSI: Statistical Analysis of VLSI Circuits* Ed. S. W. Director and W. Maly, Elsevier, 1994.
- [92] S. R. Nassif, A. J. Strojwas and S. W. Director, "A Method for Worst Case Analysis of Integrated Circuits," *IEEE Trans. CAD*, vol. CAD-5, no. 1, pp. 104-113, Jan. 1986.

- [93] W. Maly et al, "Statistical Simulation of VLSI IC Cell," *ICCAD*, pp. 254-255, 1983.
- [94] H. Walker and S. W. Director, "Yield Simulation For Integrated Circuits," *ICCAD*, pp. 256-257, 1983.
- [95] M. Styblinski and L. Opalski, "Software Tools for IC Yield Optimizations with Technological Process Parameters," *ICCAD*, pp. 158-160, 1984.
- [96] W. Maly, "Modeling of Point Defect Related Yield Loss," *ICCAD*, pp. 161-163, 1984.
- [97] S. Liu and K. Singhal "A Statistical Model for MOSFETs," *ICCAD*, pp. 78-80, 1985.
- [98] N. Herr and J. Barnes, "Statistical Modeling for Circuit Simulation of CMOS VLSI," *ICCAD*, pp. 81-83, 1985.
- [99] D. Hocevar, P. Cox and P. Yang, "Parameteric Yield Optimization for VLSI," *ICCAD*, pp. 312-314, 1985.
- [100] H. Walker and S. W. Director, "VLASIC: A Yield Simulator for Integrated Circuits," *ICCAD*, pp. 318-320, 1985.
- [101] K. Antreich and H. Graeb, "Circuit Optimization by Worst Case Distances," *ICCAD*, pp. 166-169, 1991.
- [102] S. Aftab and M. Styblinski, "A New Efficient Approach to Statistical Delay Modeling of CMOS Digital Combinational Circuits," *ICCAD*, pp. 200-203, 1994.
- [103] E. Felt, A. Narayan and A. Sangiovanni-Vincentelli, "Measurermant and Modeling of MOS Transistor Current Mismatch in Analog ICs," *ICCAD*, pp. 272-277, 1994.
- [104] W. Maly, H. Heineken, J. Khare and P. K. Nag, "Design for Manufacturability in Submicron Domain," *ICCAD*, pp. 690-697, 1996.

# A METHODOLOGY FOR WORST CASE DESIGN OF INTEGRATED CIRCUITS

A. J. Strojwas, S. R. Nassif and S. W. Director

*Center for Computer Aided Design,  
Department of Electrical Engineering,  
Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.*

## Abstract

This paper presents a formal approach to the worst-case design of Integrated Circuits, yielding realistic estimates of variations in device performances. The worst-case analysis is performed in terms of statistically independent process disturbances and employs the statistical process simulator, FABRICS II.

## 1. Proposed Methodology

In order to achieve satisfactory manufacturing yield, the design of an IC is usually verified under some worst-case conditions. This process, called worst-case design, has been traditionally performed in terms of the electrical parameters of the IC devices, such as threshold voltages and transconductances for MOSFET's. In reality, however, these parameters are statistically dependent (correlated) random variables with a multilevel structure of variance (intra-die and inter-die). In traditional approaches to worst-case design, the correlation coefficients between device parameters are not taken into account, and therefore IC performances are estimated for some unrealistic combinations of the device parameters. Hence the results of such an analysis are usually too pessimistic.

In this paper we propose a more rigid approach to worst-case design, which yields realistic estimates to variations in device performances. This approach is based upon the availability of the statistical process simulator, FABRICS II [2] which contains a sequence of process and device models. FABRICS II generates samples of device parameters for a set of process and layout parameters, as well as process disturbances which model random fluctuations inherent in the IC fabrication process (e.g. diffusivity of impurity atoms, or linewidth variations and misalignments in the lithography).

The simulator can be tuned to a particular IC manufacturing process (i.e. the joint probability density function of device parameters estimated from simulated data is in good agreement with measured results) by finding the probability distribution functions of the process disturbances. Since the process disturbances are chosen in FABRICS in such a way that they are statistically independent [1],

we propose using these parameters in the worst-case analysis. The exact worst-case analysis has to be carried out for each IC performance separately (e.g. for average power dissipated in the IC or inertial delay of a signal).

In the approach proposed, the selection of the significant worst case parameters is performed based upon the sensitivities of the performances to process disturbances. The sensitivities are estimated by perturbations using data obtained from FABRICS II, tuned to a particular IC fabrication process, coupled with a circuit simulator. Since sensitivities are local estimates of this dependence, a more accurate method of approximating the relationship over a wide range of changes in process disturbances is to build non-linear regression models relating performances to the process disturbances. Such models have been successfully built [3] and it was found that these dependencies are monotonic over a wide range of process disturbances. Therefore, sensitivities can be reliably estimated by large perturbations and the worst-case combinations of significant process disturbances can be obtained for each IC performance. Then realistic worst case sets of device parameters may be generated by FABRICS II if the significant process disturbances are changed by, for example, one standard deviation from their identified values in the "worst-case direction". Due to independence of the process disturbances, it is possible to estimate the probability of occurrence of this case, which would be valuable information for IC designers. Furthermore, if the device models are defined in such a way as to be independent of device dimensions, then the designer may alter IC layout to improve performance. If, however, some changes in the fabrication process parameters are necessary, the worst-case device models have to be evaluated once again. Observe that such an evaluation is computationally inexpensive because in the proposed approach Monte Carlo simulations are not required. Moreover, the process disturbances are chosen to be independent of process parameters and layout dimensions, so FABRICS II does not have to be re-tuned.

Theoretically, worst-case device models have to be evaluated for each IC under consideration. However, we can obtain these models for typical performances of the basic cells of VLSI circuits (e.g. power and speed of an inverter) and use these models for the approximate worst-case analysis of large IC's. Note that due to the multilevel structure of the process disturbances implemented in FABRICS II, the worst-case design can be performed for both intra-die and inter-die fluctuations.

To illustrate the methodology proposed in this paper we present an example of a worst-case analysis for an exclusive-OR gate with two inputs. This gate is implemented in a  $3 \mu$  NMOS process and consists of 7 MOS transistors. The gate was loaded with a capacitive load equivalent to a fanout of 4. The worst-case analysis was performed for the following performances:

- ◇  $P$  - power dissipated

- ◊  $\tau_d$  - delay from input to output signals
- ◊  $\tau_r$  - time of the output signal
- ◊  $\tau_f$  - time of the output signal

The sensitivity analysis was performed based on data obtained from FABRICKS II tuned to this NMOS fabrication process and SPICE. A number of simulations were performed: for the nominal design (with the disturbances equal to their mean values) and for two perturbations of each process disturbance. The sensitivities of the circuit performances with respect to each process disturbances were calculated and normalized by the corresponding nominal values. The performances under consideration were most sensitive to the following process disturbances:

- ◊  $L_N$  - linewidth variation in nitride lithography
- ◊  $L_P$  - linewidth variation in polysilicon lithography
- ◊  $D_B$  - Boron diffusivity
- ◊  $D_{As}$  - Arsenic diffusivity
- ◊  $R_{ox}$  - parabolic dry oxide growth rate

The normalized sensitivities of the circuit performances to these process disturbances are given in Table 1.

performance	$L_N$	$L_P$	$D_B$	$D_{As}$	$R_{ox}$
$P$	-0.199	0.137	-0.025	-0.064	-0.277
$\tau_r$	0.158	-0.282	-0.001	-0.188	-0.028
$\tau_f$	0.096	-0.466	0.089	-0.170	0.603
$\tau_d$	0.127	-0.352	0.459	-0.304	0.247

Table 1. Sensitivity of performance to process disturbances.

The worst-case combination of the process disturbances for each circuit performance was determined based upon the signs of the sensitivities. We decided to verify the performance of the circuit under consideration in the case when all the process disturbances are shifted by two standard deviations from their respective mean values in the worst-case direction. As an example, the nominal power dissipation of 0.58mW was increased to 1.1mW in the worst case for power, similarly, the nominal inertial delay of 1.8ns was increased to a worst case value of 3.56ns.

The most significant device model parameters corresponding to these cases are shown in Table 2.

$V_{thD}$ (v)	$Kp_D$ (a/v <sup>2</sup> )	$V_{thE}$ (v)	$Kp_E$ (a/v <sup>2</sup> )	$T_{ox}$ (Å)	$X_j$ (μm)
nominal					
-3.6	3.32E-5	1.18	4.16E-5	817	0.403
WC1-P					
-3.75	3.55E-5	1.04	4.71E-5	739	0.436
WC2- $\tau_d$					
-3.55	3.18E-5	1.27	3.87E-5	870	0.358

Table 2. Nominal and worst case device performance.

## 2. Summary

In closing, we presented a methodology for a formal analysis of the worst case performance of IC's, which gives results that are accurate, and is computationally efficient. Furthermore, the same methodology may be used to generate generic worst case device parameters independent of circuit layout.

## References

- [1] Wojciech Maly and Andrzej J. Strojwas. "Statistical Simulation of the IC Manufacturing Process", IEEE Trans. CAD, Vol. 1, March 1982.
- [2] Sani R. Nassif and Andrzej J. Strojwas and Stepehn. W. Director. "FABRICS-II: A Statistical Simulator of the IC Fabrication Process", Proceedings of International Conference on Circuits and Computers, September 1982.
- [3] Andrzej. J. Strojwas. "A Pattern Recognition Based System for IC Failure Analysis", PhD Thesis, Carnegie-Mellon University, 1982.

# TIMING ANALYSIS USING FUNCTIONAL RELATIONSHIPS

Daniel Brand and Vijay S. Iyengar

*IBM Thomas J. Watson Research Center  
Yorktown Heights, New York 10598, USA*

## Abstract

The usual block oriented timing analysis for logic circuits does not take into account functional relations between signals. If we take functional relations into consideration we may find that a long path is never activated. This observation can be used to calculate improved and more accurate delays. It is not practical to consider the complete truth table with all the relationships between signals. We use a procedure that considers only a subset of the relationships between signals and checks for non-functional paths. The delay calculation then takes into account the discovered non-functional paths to determine less pessimistic delays through the logic.

## 1. Motivation

Static timing analysis tools [1] use block oriented algorithms to compute the worst case delays in a combinational network. The arrival time of a signal, namely when the value of that signal is valid, is calculated assuming that information propagates over all the paths to that signal.

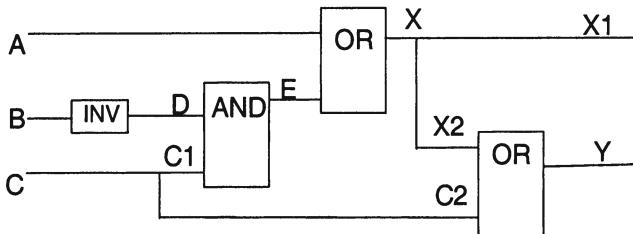


Figure 1. Simple example of a logic circuit

An example of a combinational network is shown in Figure 1. It contains four gates with their functions indicated. It contains seven signals A, B, C, D, E, X, Y and nine connections A, B, C1, C2, D, E, X1, X2, Y. (A signal is a set of connections with a common source.) There are four paths to the connection Y, namely  $A*X2*Y$ ,  $B*D*E*X2*Y$ ,  $C1*E*X2*Y$ ,  $C2*Y$ . A path is an ordered sequence of connections in the usual sense and we use \* as a separator or concatenation symbol.

In all our examples we will assume, for simplicity, that each logic block (AND, OR, INV) has a delay of 1 unit to propagate a value from any of its inputs to any of its outputs. Let us also assume that the primary inputs A, B, C are available at time 0. A block oriented algorithm such as [1] would compute the arrival time at Y to be 4 units. We can see that this is too pessimistic by considering the analysis in Table 1. The two cases in Table 1 correspond to values 0 and 1 for the primary input C. Input A has value a, input B has value b. When C is 0 the output Y reaches its steady state (value a) after only 3 units of time. When C is 1 the output Y reaches its final value 1 in just 1 unit of time. In both cases, the signal Y does have a valid value at time 3.

The block oriented algorithm did not consider the logical relationship between the connections C1 and C2. The long path  $B*D*E*X2*Y$  that contributed to the 4 unit arrival time at Y is always blocked since the AND gate and the OR gate cannot get gating values simultaneously.

It is not true in general that reduction in arrival time at a primary output implies non-testability of a connection in the sense of [2] (stuck-at faults) through that primary output. For example, the single output circuit in Figure 2 has no untestable stuck faults (stuck-at-zero and stuck-at-one) on its connections. However, the path  $B*P*R2$  is always blocked and taking this into account could improve the arrival time of the output O (for some combination of arrival times on the inputs).

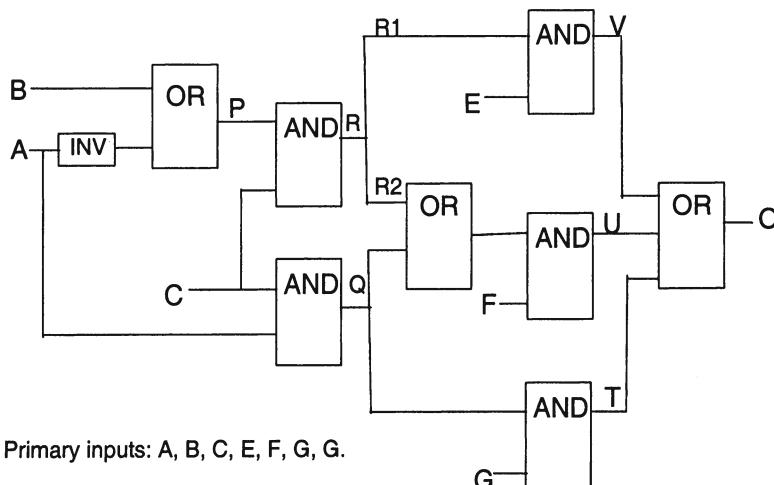


Figure 2. Example of a single output logic circuit with the path  $B*P*R2$  always blocked but without any untestable stuck faults.

Conversely it is not true in general that non-testability of a connection through a primary output implies that the delay from that connection to the primary

output can be ignored. For example, in Figure 3, the connections A1 and B1 could be removed without changing the function of the output G. However, that does not imply that the arrival time at G can be reduced to 2; the input A=0, B=0, C=0 requires three stages to propagate to G.

In our analysis we will only consider combinational logic networks. All the input combinations to these networks are assumed possible. For a synchronous sequential machine we will extract the combinational logic determining the outputs and values latched in each cycle. It is possible that certain combinations of latch states are not achievable during normal operation. This could then also result in various blocked paths. Our analysis considering only one cycle paths will not identify such blocked paths, unless the impossible combinations are given to us explicitly as don't cares.

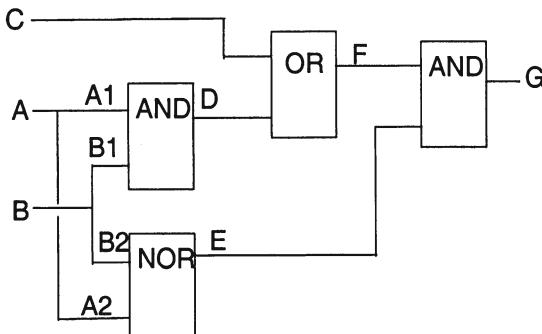


Figure 3. Example of a logic circuit with some untestable faults.

## 2. Algorithm

We will briefly describe the algorithm here. More details and a proof of correctness can be found in [3]. It is based on tracing a number of paths from an output towards inputs and collecting conditions under which the path is blocked. If it obtains an inconsistent condition then the path is always blocked and its contribution to the output's arrival time can be ignored.

For example assume that we trace the longest path in Figure 1. We start at the output Y and try to proceed through the connection X2. In order to avoid blockage we must set C=0. Then from X we proceed through E, for which we must set A=0. In order to go from E through D, we must set C=1, which is inconsistent with our previous requirements and therefore we can ignore the arrival time of D. Ignoring the arrival time of a signal is equivalent to assuming that it is always available.

An important part of our timing analysis is a procedure that collects conditions, derives more conclusions from them and checks for consistency. We use the same procedure as in [4]. It makes the following deductions:

- if  $c = \text{AND}(a, b)$  and  $a=1, b=1$  then  $c=1$ .
- if  $c = \text{AND}(a, b)$  and  $c=0, b=1$  then  $a=0$ .
- if  $c = \text{AND}(a, b)$  and  $a=0$  then  $c=0$ .
- if  $c = \text{AND}(a, b)$  and  $c=1$  then  $a=1, b=1$ .

It also makes all the other deductions obtained by adding gate inputs, permuting gate inputs, or changing the gate's function. However, it will never split into cases; if it is given for the above AND gate that  $c=0$  then it will not split into the two case  $a=0$  and  $b=0$  to see if both lead to inconsistency. The failure to split into cases means that the procedure may declare a set of conditions consistent, even though they are not. This may cause our delay calculation to be too conservative, but it will not cause it to be wrong. Completeness is sacrificed for performance reasons; we find that we have a good trade-off between efficiency and deductive power.

### 3. Experimental results

The method was implemented in PL/I as part of the logic synthesis system [5] and run on IBM 3081 and IBM 4381 computers. Two types of experiments were done. In the first one our method of doing timing analysis was applied to several pieces of logic in order to determine to what degree one can improve on the usual timing analysis. All the examples were implemented in the book set of LSI Inc. 5000 Series TFLH [6] and corresponding delay equations were used. We set our timing specifications to be not achievable, so that all outputs and all paths were late. This had two consequences – maximum reduction in arrival time, and larger CPU time that what one can expect with more realistic timing requirements.

Table 2 shows for each example

- (i) size in terms of the number of gates and connections,
- (ii) the total number of outputs and number of outputs with reduced arrival time,
- (iii) CPU time for the standard timing analysis B, CPU time to compute the reduced arrival time R for all outputs as well as per output.

Table 3 shows the amount of reduction for all the outputs. Each line corresponds to one example; the last line is the total over all. The column labeled "0" gives the number of outputs with no reduction. The column labeled "0 to 2" gives the number of outputs whose reduction was more than 0, but no more than 2%. The last column given the number of outputs with a reduction between 60%

and 62%. The percentage of reduction for an output O is calculated as follows: Let b be the lower bound of B(O) and let r be the lower bound of R(O). Then the percentage is  $100*(b-r)/b$ .

In the second type of experiments we investigated the effect of the delay model on the amount of reduction achieved. The examples used for this were originally used as test cases for test generation programs [7]. The examples are described in terms of generic logic gates (ANDs, ORs, etc.). Exclusive ORs were expanded in our analysis. Two delay models were used for the analysis. The first was a unit delay model at the gate level and the second was a model where the delay through a gate is proportional to the logarithm of the number of inputs to the gate (one input gates have zero delay). The log delay model is applicable in estimating delay for a dynamic CMOS technology.

Table 4 gives the characteristics of the examples in the second set. Table 5 summarizes the reduction achieved for these examples using both delay models. Lastly, Table 6 gives the actual reduction of arrival times for outputs of those examples that were reduced. It is clear that the analysis and the arrival time reduction is sensitive to the actual delay model.

## 4. Conclusions

The purpose of our study was to see whether one can reduce delay by considering functional relations between signals. The answer is affirmative and Table 3 shows the amount of reduction.

Another question is what happens in large pieces of logic, which are of real interest. One can expect that a longer path is more likely reducible than a short one; our data are consistent with this expectation. Unfortunately we were unable to run larger examples; our implementation did not have performance as its main objective, and indeed proved too slow.

In terms of CPU time our performance is much worse than the standard timing analysis of [1]. This is likely to remain true for any algorithm that takes functionality into consideration. Therefore this type of analysis probably cannot be applied to the whole machine, only to some troublesome areas.

We cannot draw a conclusion that with our method one could reduce the cycle time of a machine. Because critical paths may be implemented with sufficient care, taking functionality into consideration. However, we can expect that one could save on area and power, which must be sometimes used to speed up some slow paths.

## 5. Acknowledgments

We would like to express our thanks to Z. Barzilai, G. Ditlow, W. Donath, S. Gundersen, D. Ostapko, B. Rosen, and L. Trevillyan for their suggestions.

## References

- [1] R.B. Hitchcock, Sr., G.L. Smith, D.D. Cheng. "Timing Analysis of Computer Hardware," IBM Journal of Research and Development, January 1982, 100-105.
- [2] M.A. Breuer and A.D. Friedman. "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, Inc., 1976.
- [3] D. Brand, V.S. Iyengar. "Timing Analysis Using Functional Analysis," IBM Research Report, No. RC 11768, March 1986.
- [4] D. Brand. "Redundancy and Don't cares in Logic Synthesis," IEEE Transactions on Computers, Vol. C-32, No. 10, October 1983.
- [5] J.A. Darringer, D. Brand, J.V. Gerbi, W.H. Joyner, Jr., L. Trevillyan. "LSS: A system for Production Logic Synthesis," IBM Journal of research and Development, Vol. 28, No. 5, September 1984.
- [6] "CMOS Macrocell Manual AR50-000001-20 B," LSI Inc.
- [7] Special session on "Recent Algorithms for Gate-level ATPG with Fault Simulation and Their Performance Assessment," at International Symposium on Circuits and Systems, June 1985.

*Table 1.* Propagation of signal values in the circuit shown in Figure 1.

time	Signals							
	A	B	C	D	E	X	Y	
0	<i>a</i>	<i>b</i>	0	-	-	-	-	
1	<i>a</i>	<i>b</i>	0	$\bar{b}$	0	-	-	
2	<i>a</i>	<i>b</i>	0	$\bar{b}$	0	<i>a</i>	-	
3	<i>a</i>	<i>b</i>	0	$\bar{b}$	0	<i>a</i>	<i>a</i>	
0	<i>a</i>	<i>b</i>	1	-	-	-	-	
1	<i>a</i>	<i>b</i>	1	$\bar{b}$	-	-	1	
2	<i>a</i>	<i>b</i>	1	$\bar{b}$	$\bar{b}$	-	1	
3	<i>a</i>	<i>b</i>	1	$\bar{b}$	$\bar{b}$	<i>x</i>	1	

Table 2. Characteristics of one set of examples and run times.

No.	SIZE		Num OUTPUTS		CPU TIME on IBM 3091K min:sec			
	Gates	Connects	Total	Reduced by R	For B		For R	
					Total		Per Output	
1	331	606	23	12	0:06	0:28		0:01
2	466	854	54	0	0:08	0:11		0:01
3	526	975	69	0	0:04	1:19		0:01
4	787	1353	106	0	0:16	0:26		0:01
5	1030	2375	107	0	0:24	0:52		0:01
6	1628	3008	79	0	0:11	0:43		0:01
7	3049	6355	516	69	0:23	24:00		0:03
8	4948	9914	602	73	0:35	80:22		0:08

Table 3. Reduction in arrival time for outputs in the first set of examples.

Ex.	No. of outputs with arrival time reduced by given percentage														
	0	2	4	6	8	10	12	14	16	18	20	22	26	28	60
	to	to	to	to	to	to	to	to	to	to	to	to	to	to	to
0	2	4	6	8	10	12	14	16	18	20	22	24	28	30	62
1	11										4	3	4	1	
2	54														
3	69														
4	106														
5	107														
6	71	2													
7	447		19	16											
8	529	3	4		33	18	7	5	2	1					
Tot.	1369	5	23	16	36	20	17	22	6	5	5	1	3	1	1

Table 4. Characteristics for the second set of examples.

Name	SIZE	
	GATES	SIGNALS
C432	160	196
C499	202	243
C880	383	443
C1355	546	587
C1908	880	913
C2670	1193	1426
C3540	1669	1719
C5315	2307	2485
C6288	2416	2448
C7522	3512	3719

**Table 5.** Reduction achieved and run time for two delay models on the second set of examples.  
 (\* The analysis in this case did not complete.)

Name	Total no. of outputs	Unit delay model		Log delay model	
		no. of outputs reduced by R	CPU Time for R (on IBM 4381)	no. of outputs reduced by R	CPU Time for R (on IBM 4381)
C432	7	0	0:33	0	0:12
C499	32	0	0:36	0	0:35
C880	26	0	0:18	0	0:20
C1355	32	0	0:46	0	0:54
C1908	25	8	8:29	6	9:40
C2670	140	0	3:14	0	5:32
C3540	22	15	100:16	6	4:31
C5315	123	6	55:18	0	3:15
C6288	32	0	5:32	*	*
C7522	108	13	35:58	0	5:32

*Table 6.* Reduction in arrival time for the outputs for examples in the second set for two delay models. (Note: Only examples for which any reduction occurred included)

# ON THE DESIGN OF ROBUST MULTIPLE FAULT TESTABLE CMOS COMBINATIONAL LOGIC CIRCUITS

Sandip Kundu

*T. J. Watson Res. Ctr.*

*PO Box 218*

*Yorktown Heights, NY 10598*

Sudhakar M. Reddy

*Dept. of Elec. & Comp. Engg.*

*University of Iowa*

*Iowa City, IA 52242*

Niraj K. Jha

*Dept. of Elec. Engg.*

*Princeton University*

*Princeton, NJ 08544*

## Abstract

It is known that circuit delays and timing skews in input changes influence choice of tests to detect delay faults. Tests for stuck-open faults in CMOS logic circuits could also be invalidated by circuit delays and timing skews in input changes. Tests that detect modeled faults independent of the delays in the circuit under test are called robust tests. An integrated approach to the design of combinational logic circuits in which all single stuck-open faults and path delay faults are detectable by robust tests was presented by the authors earlier. This paper considers design of CMOS combinational logic circuits in which all *multiple* stuck-at, stuck-open and all *multiple* path delay faults are robustly testable.

## 1. Introduction

Testing is needed to ensure reliability of VLSI chips. There are two areas of testing. One is to verify the input-output logic relations of a circuit, and the other is to ensure that path delays in manufactured circuits meet specifications. CMOS has emerged as the dominant technology for manufacturing digital logic circuits. The classical fault model consisting of circuit lines stuck-at-0 or 1 faults was shown to be inadequate for CMOS circuits [1-3]. Transistor stuck-on (TSON) and transistor stuck-open (TSOP) faults were added to this model to capture the effects of physical faults [4]. Testing TSOP faults in a static CMOS

combinational logic circuit requires *two-pattern* testing [4,5]. A two-pattern test  $< T_1, T_2 >$  consists of a sequence of two inputs, the first one of which is called the *initializing input* ( $T_1$ ) and the later one is called the *test input* ( $T_2$ ).

Among the two popular delay fault models, namely the *gate delay fault model* and *path delay fault model*, the latter is deemed to be more general since it captures the cumulative effect of small delay variations in gates along a path [7] as well as the faults caused by a single gate. We use path delay fault model in this paper.

In a two-pattern testing environment when an initializing input is changed to a test input, transients may be produced on circuit lines causing *test invalidation problem* [4]. In path delay testing, besides test invalidation problem, it may not always be possible to sensitize a signal transition to an output along a desired path of a given circuit, thus rendering the selected path *untestable* [8]. To circumvent the test invalidation problem, tests called *robust tests* were introduced [9]. Robust tests for TSOP faults in CMOS circuits are defined to be the two-pattern tests that are not invalidated by transient signals caused by arbitrary circuit delays and/or timing skews in input changes. A robust test for a path-delay fault is a two-pattern test that is not invalidated by transient inputs and delays in other paths. However, for given faults, in a circuit under test, such robust tests may not exist [4,9].

To overcome these and other problems in testing, a necessity has always been felt for *testable designs*. With earlier testable designs, it is not possible to design reliable CMOS gates with fan-in greater than 4-8, without using extra inputs [4-6,9,10] called *control inputs*. Recently, we have presented a testable design that can accommodate any fan-in restriction, and requires no extra input or special gates. These designs are robustly testable for all single TSOP and all multiple path-delay faults [11]. In this paper, it is shown that the circuit produced by the design procedure given in [11] is testable with respect to *multiple stuck-at and stuck-open* faults as well. The tests for multiple faults are given.

## 2. Preliminaries

In this section, relevant earlier results are reviewed and notation used is developed.

A function  $F$  of  $n$  variables  $x_1, x_2, \dots, x_n$  is said to be *positive unate* (*negative unate*) in variable  $x_i$  iff  $F(x_1, x_2, \dots, x_n)$ , a two-level logic expression for  $F$ , exists using  $x_i$  exclusively in uncomplemented (complemented) form [12]. For example,  $F(a, b, c, d) = ab + \bar{c}\bar{b} + \bar{c}\bar{a} + \bar{a}\bar{d}$  is positive unate in  $b$ , negative unate in  $d$  and *non-unate or binate* in  $a$  or  $c$ . A *unate function* (*positive unate function*) is unate (positive unate) in all of its variables. Given that  $F$  is a unate function, it is known that *minimal sum of products* (*SOP*) as well as *minimal product of*

*sums* (POS) expressions for  $F$  are unique. A two-level circuit of NAND (NOR) gates realizes a function in its sum of products form (product of sums form).

If  $F(x_1, x_2, \dots, x_n)$  is a non-unate function and  $x_i$  is a non-unate variable of  $F$ , then from Shannon's expansion theorem,

$$F(x_1, x_2, \dots, x_n) = x_i F_{x_i} + \bar{x}_i F_{\bar{x}_i} \quad \text{and,} \quad (1)$$

$$F(x_1, x_2, \dots, x_n) = (\bar{x}_i + F_{x_i})(x_i + F_{\bar{x}_i}) \quad (2)$$

where,  $F_{\bar{x}_i}$  and  $F_{x_i}$  are defined as below:

$$F_{\bar{x}_i} = F(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

$$F_{x_i} = F(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

$F_{\bar{x}_i}$  and  $F_{x_i}$  are independent of  $x_i$  and depend on the other  $(n - 1)$  variables. The decomposition of Equation (1) can be implemented by the circuit shown in Figure 1(a) and Equation (2) can be implemented by the circuit in Figure 1(b).

The authors have earlier established that if a unate function  $F$  is realized by a two-level NAND (NOR) gate circuit then it is robustly testable for all path delay faults [11]. Furthermore, *high fan-in gates in a testable two-level realization of a unate function can be replaced by trees of primitive gates of lower fan-in without affecting single fault testability*. Thus, testable realizations with fan-in restrictions can be obtained for unate functions. It was also shown that if a two-level circuit is robustly testable for all path delay faults, then it is robustly testable for all single TSOP faults, and the following procedure was suggested to derive testable designs for an arbitrary function.

#### Procedure\_Testable\_Design( $F$ )

**Step 1:** Minimize function  $F$  for a two-level design.

**Step 2:** Check if the path delay faults in the two-level circuit resulting from the above minimization are detectable. If yes, stop here.

**Step 3:** If the result of Step 2 is negative, then apply heuristic for selecting the splitting variable ( $x_i$ ).

**Step 4:** Call Procedure\_Testable\_Design ( $F_{x_i}$ ) and Procedure\_Testable\_Design ( $F_{\bar{x}_i}$ ) to realize the circuit form shown in Figure 1.

### 3. Multiple-fault Testability

In this section, we prove that the circuit produced by the above procedure is robustly testable with respect to all *multiple* line stuck-at and TSOP faults. We need not consider multiple path delay faults for it was shown in [11] that the circuit produced by Procedure\_Testable\_Design( $F$ ) is testable for all multiple path delay faults.

#### A Useful Review

Given a product  $P_i$  of a positive unate function,  $P_i = x_{i_1}x_{i_2}\dots x_{i_r}$ , the *vertex corresponding to  $P_i$* , named  $V_{P_i}$ , is the binary n-tuple that has 1s in positions  $i_1, i_2, \dots, i_r$  and 0s in all other positions. Similarly, given a binary n-tuple  $X$ , the

*product corresponding to X*, named  $P_X$ , is obtained by including the unimplemented variable in  $P_X$  corresponding to each position in X with 1.

**Example:** Let  $x_1, x_2, x_3, x_4, x_5$  be the variables. If Q is the product  $x_2x_4x_5$  then the corresponding vertex  $V_Q$  is 01011, and if V = 10110 is a vertex then the corresponding product is  $x_1x_3x_4$ .

A zero cofactor (0-cofactor) of  $P_i$  of a positive unate function is obtained by substituting 0 for a 1 in  $V_{P_i}$  and is denoted by  $V_{P_i}(k)$ , when a 1 at position  $k$ , is flipped to a 0. Thus, if  $P_i$  consists of  $r$  literals, it has  $r$  0-cofactors. In the example above,  $x_2x_4x_5$  is a product term. So  $V_{x_2x_4x_5}$  is 01011 and  $V_{x_2x_4x_5}(4) = 01001$ .

In the method to be proposed for the design of robustly testable circuits, an important fact used is that robust tests for two-level NAND-NAND gate realizations of the minimum SOP of a positive unate function  $F$  can be derived from the minimal true vertices of  $F$  and their 0-cofactors. For this, it is useful to discuss the following properties of 0-cofactors and minimal true vertices.

Let  $F = P_1 + P_2 + \dots + P_m$  be the minimal sum of products of the unate function  $F$ . Then, clearly,  $F(V_{P_i}) = 1, 1 \leq i \leq m$ . Also, if  $V_{P_i}(k)$  is a 0-cofactor of  $P_i$  then  $F(V_{P_i}(k)) = 0$ . Let  $F$  be realized by a NAND-NAND circuit  $N$ , shown in Figure 2, where a first-level gate  $G_i$  corresponds to the product  $P_i$ , and  $G_o$  is the output gate.

Theorems 1-4 presented in the sequel assume that two-level circuits are constructed of NAND gates. The proofs for the two-level circuits of NOR gates can be easily established as the dual case. For easy understanding, the proof of Theorem 1 is given for positive unate functions only. Extension of the proof for more general unate function is trivial on the lines discussed earlier [11].

**Theorem 1:** A two-level irredundant realization of a positive unate function  $F$  is robustly testable with respect to all multiple stuck-at and stuck-open faults.

**Proof:** A first-level gate  $G_i$  realizes the complement of a product  $P_i$ . Let us consider tests for pFET TSOP faults first. If the pFET of  $G_i$  connected to input  $x_{i_1}$  is to be tested for a TSOP fault, then we would normally apply  $\langle V_{P_i}, V_{P_i}(i_1) \rangle$  for fault detection. The output of each gate  $G_j, 1 \leq j \leq m$  and  $j \neq i$ , is supposed to be 1 for both inputs in the two-pattern sequences while the output of gates  $G_i$  and  $G_o$  are expected to change from  $0 \rightarrow 1$  and  $1 \rightarrow 0$ , respectively. If a  $1 \rightarrow 0$  is observed at the output of  $G_o$  the following may be concluded.

1. nFETs of  $G_o$  are fault free.
2.  $V_{P_i}(i_1)$  produces all 1s at the input of  $G_o$ .
3. If the TSOP fault of the pFET being considered or a line stuck-at 1 fault for the input  $x_{i_1}$  of  $G_i$  exists, it can not produce a 1-to-0 transition at the output of  $G_o$  unless
  1. one or more nFETs of  $G_i$  are open, or,
  2. there is a line stuck-at 1 fault at the output of  $G_i$ .

If  $G_i$  has an nFET(s) TSOP fault and the output of  $G_i$  is known to be 1 at some point, then it may be assumed to be a line stuck-at 1 fault thereafter. So, if the test performed above is invalidated, we may assume that the output of  $G_i$  has a stuck-at 1 fault. In that case the 1 produced at the output of  $G_o$  when  $V_{P_i}$  was applied must be due to some other gate(s)  $G_l$  producing a 0 for this input. This could happen only if  $G_l$  had a fault, more specifically if  $G_l$  had an input, say  $x_n$  (or several inputs) stuck-at 1, or a pFET of  $G_l$  connected to the input  $x_n$  was stuck-open and the input preceding  $V_{P_i}$  initialized  $G_l$  to 0. If it was a matter of TSOP faults alone then the input  $V_{P_i}(i_1)$  followed by  $V_{P_i}$  would be able to detect the faults, since the output of  $G_l$  cannot return to 0 in that case.

If the test sequence  $\langle V_{P_i}, V_{P_i}(i_1), V_{P_i}, V_{P_i}(i_2), V_{P_i}, \dots, V_{P_i}(i_r) \rangle$  is applied to the circuit, from the arguments presented above it is clear that in presence of a fault in  $G_i$ , a correct output sequence can be obtained at  $G_o$  iff  $x_{i_1}, x_{i_2}, \dots, x_{i_r}$  are all inputs to  $G_l$ , but then the product realized by  $G_l$  is redundant, contradicting the postulate in this theorem. One could use the same argument for all gates  $G_j, 1 \leq j \leq m$ . Note that selecting this order ensures that we do not have to apply any more tests than those needed for single fault detection. The tests above also detects any fault in  $G_o$ , so we do not have to apply any additional tests.

$\langle Q.E.D \rangle$

In the theorem presented above, we have actually constructed tests for multiple faults. The test length being no more than that for single faults is naturally the best we could attain. The following theorem is about decomposition of primitive gates into tree structures. Theorems 1 and 2 together give us a tool to achieve multi-level testable design for unate functions satisfying all fan-in requirements.

**Theorem 2:** If a high fan-in primitive gate is replaced by a primitive gate tree then the tests for single TSOP faults of the tree circuit would detect all multiple faults in the tree, which consist of stuck-open and stuck-at faults.

**Proof:** Decomposition of a primitive gate (such as NAND or NOR) leads to a CMOS circuit that is absolutely fan-out free. There are no fan-outs even at the primary input level. Every primary input of this circuit has a unique path to the output. Let  $\Phi$  be the set of faults present in this circuit.  $\Phi$  consists of  $\{\phi_1, \phi_2, \dots, \phi_n\}$ . Let the fault  $\phi_1$  lie in the path from primary input  $x_i$  to the output. A test for this fault under the single fault assumption model essentially involves changing input  $x_i$  while holding all others constant. When this test is applied, two situations are possible, namely, (1) the path on which  $\phi_1$  lies is sensitized to the output even in the presence of multiple faults, (2) the path is not sensitized. In Case 1, even if another fault is present in the path on which  $\phi_1$  is situated it would be detected. In Case 2, there is nothing left to prove. The

above proof is similar to proof given in [13] for robust multiple stuck-open fault detection in fan-out free circuits.

*< Q.E.D >*

**Comments :** The tests for all single TSOP faults of the high fan-in gate *may* not be adequate to test for all single faults of the gate tree circuit.

To illustrate this consider a NAND gate with inputs  $a, b, c, d, e$ . If fan-in is restricted to 2, then the gate may be decomposed as shown in Figure 3. A complete set of tests for the high fan-in gate is:

$$\begin{aligned} &< 11111, 01111 > \\ &< 11111, 10111 > \\ &< 11111, 11011 > \\ &< 11111, 11101 > \\ &< 11111, 11110 > \\ &< 11110, 11111 > \end{aligned}$$

Suppose, an nFET of the gate realizing  $\overline{abcde}$  is stuck-open then the test is  $< 10\_\_, 11111 >$  or  $< 01\_\_, 11111 >$ . Neither is present in the above set. However, one observes that if the above tests are reversed then all single (and hence multiple) faults can be tested. Thus, the test set is the same in size, the number of times they need to be applied is more.

**Theorem 3:** If a two-level circuit is robustly testable with respect to all path delay faults, then it is robustly testable with respect to all multiple stuck-at and stuck-open faults. (This theorem includes the results stated in Theorem 1, but Theorem 1 was stated separately because the number of tests proposed in Theorem 1 is smaller).

**Proof:** As before (in Theorem 1), at first we apply tests for pFET stuck-open faults in the first level of the circuit. Since the two-level network is robustly testable with respect to all path delay faults, it is given to us that there exists a two pattern test  $< T_1, T_2 >$  such that  $T_1$  and  $T_2$  differ in only one bit position. When applied in succession, they launch a 1-to-0 transition at input  $x_{i_1}$  of gate  $G_i$ , holding all other inputs of  $G_i$  and all inputs of  $G_o$  except the output of  $G_i$  at a constant hazard free 1 in a fault-free situation. When the input sequence  $< T_1, T_2, T_1 >$  is applied to the network, it is clear from the discussion of Theorem 1 that it tests for either

- (i) line stuck-at 1 fault at the input  $x_{i_1}$  of  $G_i$ , or
- (ii) TSOP fault of the pFET of  $G_i$  connected to lead  $x_{i_1}$ .

If however, the tests fail to indicate an error even when faults may be present, then the faults invalidating this test must include input line stuck-at-1 fault(s) in some other gate at the first level (and not any stuck-open fault since they are caught by  $T_2, T_1$  transition). Every two-level irredundant circuit is testable for multiple line stuck-at faults [12] by any single stuck-at fault test set. Thus, at

some stage of this three-pattern testing a stuck-at fault would be detected. If the circuit passes all the three-pattern tests, then there are no line stuck-at faults. Therefore, all the TSOP tests for the first-level pFETs are also valid tests, and there are no faults in the pFETs of the first-level gates. Hence, there are no faults in the nFETs of the first-level gates (they yield line stuck-at faults). We have also applied all the tests needed for  $G_o$ . Hence, once it is verified that there are no faults in the first-level gates, it is clear that there is no fault in  $G_o$  either.

$< Q.E.D >$

Though the testing philosophy is same, the test length in Theorem 1 is shorter than the test length in Theorem 3, owing to the fact that for unate functions,  $V_{P_i}$  can serve as  $T_1$  for all the inputs through a gate  $G_i$ . In general, for non-unate functions we may not be able to find such an input that can serve as initializing input for all leads through the gate, thereby reducing the overlap between the tests.

**Lemma 1** [11]: Given  $F$ , a non-unate function and  $x_i$ , a binate variable of  $F$ , there exists a combination of inputs  $x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n$  such that  $F_{x_i} = 1$  and  $F_{\bar{x}_i} = 0$  ( $F_{x_i} = 0$  and  $F_{\bar{x}_i} = 1$ ).

**Proof:** Assume that there does not exist an input such that  $F_{x_i} = 1$  and  $F_{\bar{x}_i} = 0$ . This implies that for every input for which  $F_{x_i} = 1$ ,  $F_{\bar{x}_i}$  also equals 1. This implies that  $F_{\bar{x}_i}$  covers  $F_{x_i}$ , and hence  $F_{\bar{x}_i}$  can be written as  $F_{\bar{x}_i} = F_{x_i} + \hat{F}_{x_i}$  where  $\hat{F}_{x_i}$  is not equal to 0 and depends only on variables  $x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n$ . Hence, from Equation (1):

$$\begin{aligned} F &= \bar{x}_i F_{\bar{x}_i} + x_i F_{x_i} \\ &= \bar{x}_i (F_{x_i} + \hat{F}_{x_i}) + x_i F_{x_i} \\ &= \bar{x}_i \hat{F}_{x_i} + F_{x_i} \end{aligned} \quad (3)$$

Equation (3) implies that  $F$  is unate in  $x_i$ , a contradiction. Similarly, we can prove that there exists an input such that  $F_{x_i} = 0$  and  $F_{\bar{x}_i} = 1$ , for non-unate function  $F$  and its binate variable  $x_i$ .

$< Q.E.D >$

**Theorem 4:** When a non-unate function  $F$  is decomposed with respect to a binate variable  $x_i$  in either of the forms given in (1) or (2) and implemented by circuits shown in Figure 1, then all multiple line stuck-at faults and TSOP faults in the resultant circuits are detectable by robust tests if these faults in the circuits realizing  $F_{x_i}$  and  $F_{\bar{x}_i}$  are detectable by robust tests.

**Proof:** Refer to the circuit shown in Figure 1(a). It was shown in Lemma 1 that there exists a combination of inputs for which  $F_{x_i} = 1$  and  $F_{\bar{x}_i} = 0$ . Now, with this combination of inputs let us change  $x_i$  from  $1 \rightarrow 0$ . The output of  $F$  is also expected to go from  $1 \rightarrow 0$ . If this transition is observed at the output, then

- (i) the nFETs of the output gate are fault-free,
- (ii) both the lines feeding the output gate are at 1.

Note that  $\bar{x}_i$  may only change from 0 to 1. Therefore, in presence of a fault  $\bar{x}_i F_{\bar{x}_i}$  may change only from 1 to 0 and not in the reverse direction. Thus, the test performed above cannot be invalidated. If  $x_i$  is changed back to 1, a pFET of the output gate and the nFETs of the gate realizing  $x_i F_{\bar{x}_i}$  are also tested robustly. Similarly, by choosing  $F_{x_i} = 0$  and  $F_{\bar{x}_i} = 1$ , the lead  $\bar{x}_i$ , the pFET and nFET connected to it and the other pFET of the output gate are tested. Testing the rest is simple, because if  $x_i$  ( $\bar{x}_i$ ) is held at 1, and the inputs to the circuit realizing  $F_{x_i}$  ( $F_{\bar{x}_i}$ ) are changed, the output  $F_{x_i}$  ( $F_{\bar{x}_i}$ ) is guaranteed to be sensitized to the external output without any external interference.

*< Q.E.D >*

In Step 2 of Procedure\_Testable\_Design( $F$ ), if the two-level realization of a function is found to be robustly delay testable, then we do not have to modify the circuit to achieve multiple fault testability (cf. Theorem 3). However, we may need to replace gates by gate trees to satisfy fan-in restrictions. Theorem 2 guarantees that all fan-in constraints can be met while preserving the testability of a circuit derived from Procedure\_Testable\_Design( $F$ ). If two-level realization of a function is not robustly delay testable then it is clear from Steps 3 and 4 of Procedure Testable\_Design( $F$ ), that it would be expanded with respect to a binate variable using Shannon's expansion. Theorem 4 asserts that this expansion ensures multiple fault testability. Therefore, it is proved that Procedure\_Testable\_Design( $F$ ) gives us multiple fault testable design.

## 4. Conclusion

Until recently, it was not known whether it is possible to design CMOS combinational logic circuits that are testable with respect to all single transistor stuck-open faults. Earlier, the authors presented a design for arbitrary combinational logic functions that require no special gates and technology constraints such as fan-in and fan-out are easily accommodated [11]. In this paper, it is shown that the design presented earlier actually results in circuits in which all multiple path delay faults and all multiple line stuck-at and transistor stuck-open faults are detected. The tests to detect such faults are also described.

## Acknowledgments

The research done by SK and SMR was supported in part by a grant from US office of Naval Research SDIO/IST contract No. N00014-87k-0419. The work by NKJ was supported by Siemens Corp. under Grant No. 170-4051.

## References

- [1] R. L. Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *Bell System Technological Journal*, vol. 57, pp. 1449-1474, May-June 1978.
- [2] J. Galiay, Y. Crouzet, and M. Vergnault, "Physical versus logical fault models for MOS LSI circuits: Impact on their testability," *IEEE Trans. on Comp.*, vol. C-29, pp. 524-531, June 1980.
- [3] Y. M. El-Ziq and R. J. Cloutier, "Functional-level test generation for stuck-open faults in CMOS VLSI," in *Proc. Int. Test Conf.*, pp. 536-546, Oct. 1981.
- [4] S. M. Reddy, M. K. Reddy, and J. G. Kuhl, "On testable design for CMOS logic circuits," in *Proc. Int. Test Conf.*, pp. 435-445, Oct. 1983.
- [5] S. M. Reddy and M. K. Reddy, "Testable realization for FET stuck-open faults in CMOS combinational logic circuits," *IEEE Trans. on Comp.*, pp. 742-754, Aug. 1986.
- [6] N. K. Jha and J. A. Abraham, "Design of testable CMOS logic circuits under arbitrary delays," *IEEE Trans. on CAD*, vol. CAD-4, no. 3, pp. 264-269, July 1985.
- [7] G.L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, pp. 342-349, Nov. 1985.
- [8] J. Savir and W. H. McAnney, "Random pattern testability of delay faults," in *Proc. Int. Test Conf.*, pp. 263-273, Sept. 1986.
- [9] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. on CAD*, vol. CAD-6, pp. 694-703. Sept. 1987.
- [10] S. Muroga, *VLSI System Design*, Wiley, NY, 1982.
- [11] S. Kundu and S. M. Reddy, "On the design of robust testable CMOS combinational logic circuits," in *Proc. Int. Symp. on Fault-Tolerant Computing*, pp. 220-225, June 1988.
- [12] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1978.
- [13] N. K. Jha, "Multiple stuck-open fault detection in CMOS logic circuits," *IEEE Trans. on Comp.*, vol. 37, pp. 426-432, Apr. 1988.

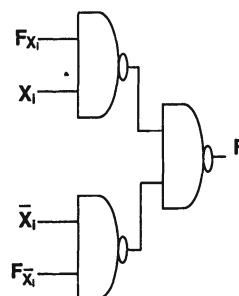


Figure 1(a).  $F = x_i F_{x_i} + \bar{x}_i F_{\bar{x}_i}$

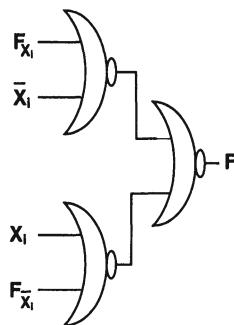


Figure 1(b).  $F = (x_i + F_{\bar{x}_i})(\bar{x}_i + F_{x_i})$

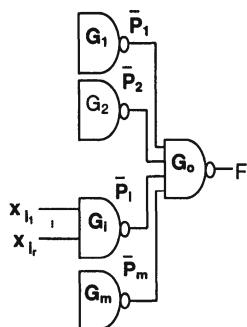


Figure 2.  $F = \sum_{i=1}^m P_i$

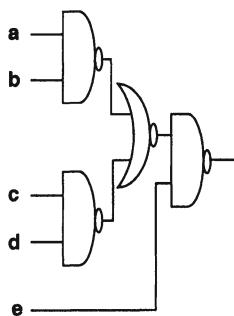


Figure 3. Tree realization of  $\overline{abcde}$

# CIRCUIT OPTIMIZATION DRIVEN BY WORST-CASE DISTANCES

Kurt J. Antreich and Helmut E. Graeb

*Institute of Electronic Design Automation,*

*Technical University of Munich, D-8000 Munich 2, Germany*

## Abstract

In this paper, a new method for circuit optimization in face of manufacturing process variations is presented. It is based on the characterization of the feasible design space by worst-case points and related gradients. The expense for this characterization is linear with the number of circuit performances. On the contrary, the complexity of other geometric approaches to tolerance oriented circuit designs increases exponentially with the dimension of the design space. A deterministic optimization procedure based on the so-called "worst-case distances" will be introduced, combining nominal and tolerance design in a single design objective. The entire optimization process with regard to performance, yield, and robustness uses sensitivity analyses and requires a much smaller number of simulations than the Monte Carlo based approaches. Moreover, the new method takes account of the partitioning of the parameter space into deterministic and statistical parameter, which is an inherent property of integrated circuit design.

## 1. Introduction

The practical use of the large variety of existing methods for tolerance oriented circuit optimization [7, 6, 5, 15, 12, 11] within integrated circuit design has been obstructed by two major problems: the incomplete statistical data of manufacturing processes and the prohibitive simulation costs of optimization methods. While the increasing reliability of statistical measurements will solve the first problem in the near future, the high simulation costs remain the crucial limitation of methods for IC optimization despite increasing computer power. The statistical methods for yield optimization are often based on Monte Carlo analysis [3, 10, 16], which usually involves thousands of circuit simulations. Recent efforts aim at reducing the high computational costs for Monte Carlo circuit simulations by approximating the performance functions. Furthermore, the Monte Carlo based methods cannot readily be applied to IC design, as the design space includes deterministic parameters. On the other hand, deterministic methods for design centering are often based on geometric concepts [7, 14, 9]. Beyond the yield maximization they aim at a general improvement of the circuit's robustness. However, their simulation costs increase exponentially with the number of parameters.

## 2. Basic Relationships

Let  $f \in \mathcal{R}^{n_f}$  denote the vector of circuit performances, such as delay, gain, and let  $z \in \mathcal{R}^{n_z}$  denote the vector of circuit parameters, such as channel width, oxide thickness. In general, the performances  $f(z)$  for a certain set  $z$  of parameters are not given explicitly and have to be computed by numerical circuit simulation.  $z$  is divided into the three subsets: statistical fixed parameters  $c$ , statistical adjustable parameters  $p$ , deterministic adjustable parameters  $x$

$$z^T = [c^T \ p^T \ x^T] = [s^T \ x^T] = [c^T \ a^T] \quad (1)$$

where  $s \in \mathcal{R}^{n_s}$  denotes the vector of statistical parameters and  $a \in \mathcal{R}^{n_a}$  denotes the vector of adjustable parameters.

The inevitable variations in the manufacturing process are described by a probability density function  $pdf(s)$  of the statistical parameters. A usual pdf is the multinormal distribution  $pdf_n(s, s_0, C)$ , which is determined by the mean values  $s_0$  of the statistical parameters and the covariance matrix  $C$ . The  $pdf_n$  is constant on  $n_s$ -ellipsoids with center  $s_0$ , as shown in Fig. 1. These  $n_s$ -ellipsoids are called tolerance bodies  $T_s(\beta)$ .

$$T_s(\beta, s_0, C) = \{s \in \mathcal{R}^{n_s} | (s - s_0)^T \cdot C^{-1} \cdot (s - s_0) \leq \beta^2\} \quad (2)$$

The volume of a tolerance body  $T_s$  can be determined by [1]:

$$V_s(\beta, C) = \int_{T_s} \int ds = \sqrt{\det C} \cdot \beta^{n_s} \cdot V_0 \quad (3)$$

$V_0$  is the volume of an  $n_s$ -sphere with radius 1. While the circuit parameters have the defined statistics, the circuit performances are supposed to keep within certain bounds called performance specifications. The specifications are given as upper and/or lower bounds that define the acceptance region  $A_f$  in the performance space, as shown in Fig. 1.

$$A_f = \{f(z) \in \mathcal{R}^{n_f} | f^L \leq f(z) \leq f^U\} \quad (4)$$

Because of the mentioned nature of the performance function  $f(z)$ , the acceptance region  $A_s$  in the statistical parameter space and the tolerance bodies  $T_f$  in the performance space (dotted curves in Fig. 1) are unknown. The acceptance of a parameter set can only be checked by simulation:

$$f(z) \in A_f \Leftrightarrow s \in A_s(x_0) \Leftrightarrow z \in A_z \quad (5)$$

We now define the parametric yield  $Y$  as the probability that the performances of a manufactured circuit are within the specified bounds.

$$Y = \text{prob} \{f(z) \in A_f\} = \int_{A_s(x_0)} \int pdf(s) ds, \quad 0 \leq Y \leq 1 \quad (6)$$

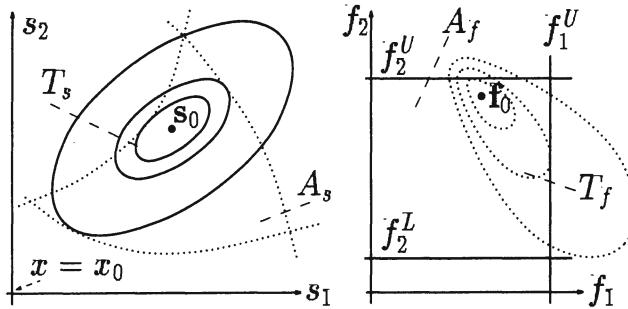


Figure 1. Tolerance body  $T_s$  in the statistical parameter space with  $n_s = 2$  (cut of total parameter space at value  $x_0$  of deterministic parameters  $x$ ) and acceptance region  $A_f$  in the performance space with  $n_f = 2$ .

Note, that the acceptance region  $A_s(x_0)$  in the subspace of statistical parameters is a cut of the total parameter space dependent on the nominal values  $x_0$  of the deterministic parameters [8, 9].

### 3. The circuit optimization task in the performance space

Consider the set of all performance bounds  $f_i^B$ ,  $i = 1, \dots, n_B$ , where  $f_i^B$  can be a lower or upper bound  $B \in \{L, U\}$ . Let  $f_i(s, x)$  denote the performance function corresponding to  $f_i^B$ , and let  $f(s_0, x_0) \in A_f$ . Then, we define the performance tolerance  $\alpha_i$  by

$$\alpha_i = |f_i^B - f_i(s_0, x_0)| \quad (7)$$

In order to formulate the circuit optimization task for an arbitrary initial point, we have to distinguish between satisfied and violated performance bounds at the nominal point. In the case of an upper bound  $f_i^U$  we define

$$\varphi_i = +1 \Leftrightarrow f_i^U \geq f_i(s_0, x_0), \quad \varphi_i = -1 \Leftrightarrow f_i^U < f_i(s_0, x_0) \quad (8)$$

In the case of a lower bound  $f_i^L$  we define

$$\varphi_i = +1 \Leftrightarrow f_i^L \leq f_i(s_0, x_0), \quad \varphi_i = -1 \Leftrightarrow f_i^L > f_i(s_0, x_0) \quad (9)$$

Then, the following assertion can be made:

$$\forall_{i \leq n_B} \varphi_i(s_0, x_0) = +1 \Leftrightarrow f(z_0) \in A_f \Leftrightarrow s_0 \in A_s(x_0) \quad (10)$$

In order to prepare the new approach to circuit optimization, we first repeat the problem of circuit optimization in the performance space.

$$\max_{p_0, x_0} \alpha(s_0, x_0), \quad \alpha = [\dots \varphi_i(s_0, x_0) \cdot \alpha_i(s_0, x_0) \dots]^T \quad (11)$$

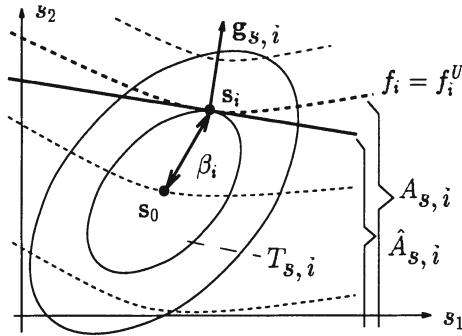


Figure 2. Worst-case point  $s_i$ , worst-case distance  $\beta_i$ , maixmal tolerance body  $T_{s,i}$  inside acceptance region  $A_{s,i}$ , gradient  $g_{s,i}$  and approximation  $\hat{A}_{s,i}$  to  $A_{s,i}$  determined by  $g_{s,i}$  and  $s_i$  for one specification  $f_i^U$  (dashed lines: level contours of  $f_i$ ).

Due to the tradeoff between the vector components of  $\alpha$ , a unique solution to (11) usually does not exist. Multiple Criterion Optimization methods determine efficient solution points, i.e., points, from which it is impossible to improve one objective without making another one worse.

#### 4. Worst-case distances and related gradients

In order to consider the variations of the statistical parameters  $s$  as well as the performance sensitivities with respect to  $s$ , the performance tolerances  $\alpha_i$  are transformed into the subspace of statistical parameters  $\alpha_i \rightarrow \beta_i$ . Due to the ambiguity in the solution of  $f_i^B = f_i(s, x_0)$  in  $s$ , a decision has to be made. We choose the point  $s_i$  with the smallest distance  $\beta_i$  to  $s_0$ , measured in a well-defined norm  $\|M^{-1} \cdot (s_i - s_0)\|$ , using the weighting matrix  $M^{-1}$ , as illustrated in Fig. 2.

$$\beta_i = \min_s \|M^{-1} \cdot (s - s_0)\| \text{ subject to } f_i^B = f_i(s, x_0) \quad (12)$$

For the first time in [13], Mueller-L. has introduced a general definition or the worst-case point  $s_i$ . Mueller-L. applies the worst-case points to the solution of an analysis task. He calculates the worst-case points approximately, calling them “limit parameters”, and uses them for the characterization of the worst-case problem. In this paper, the idea of limit parameters is adopted. We calculate the limit parameters exactly by solving the standard optimization problem (12) and call them “worst-case points”. Furthermore, the worst-case distance  $\beta_i = \|M^{-1} \cdot (s_i - s_0)\|$  and the performance gradients at these points will be applied to a deterministic optimization procedure. The concept of worst-case distances is particularly suited for the characterization and solution of the circuit optimization problem and leads to a unified view of nominal and tolerance design.

To the norm  $\|M^{-1} \cdot (s - s_0)\|$  the norm body  $N_s$  can be associated.

$$N_s(\beta, s_0, M) = \{s \in \mathcal{R}^{n_s} | \|M^{-1} \cdot (s - s_0)\| \leq \beta\} \quad (13)$$

In order to represent the parameter statistics in problem formulation (12), we define the norm to be used in (12) such that the tolerance bodies of the pdf equal the norm bodies of (13). For a normal pdf, the equality of (2) and (13) is achieved by using a weighted  $l_2$ -norm according to the Cholesky decomposition of the covariance matrix  $C = M \cdot M^T$ :

$$\beta = \|M^{-1} \cdot (s - s_0)\|_2 = \sqrt{(s - s_0)^T \cdot C^{-1} \cdot (s - s_0)} \quad (14)$$

Fig. 2 shows the level contours (dashed lines) of a performance  $f_i$  corresponding to one upper performance bound  $f_i^U$  in the statistical parameter space, the tolerance bodies  $T_s$  (ellipses) according to (2), (14), and the result of (12). The bold dashed line represents the parameter sets for which the constraint  $f_i = f_i^U$  in (12) is satisfied. It is the boundary of the acceptance region  $A_{s,i}(x_0) = \{s \in \mathcal{R}^{n_s} | f_i(s, x_0) \leq f_i^U\}$  determined by the single specification  $f_i^U$ . Generally, we define

$$A_{s,i}(x_0) = \{s \in \mathcal{R}^{n_s} | \phi_i(s, x_0) = +1\} \quad (15)$$

The solution of (12) for a single bound  $f_i^B$  yields the worst-case point  $s_i$ , the worst-case distance  $\beta_i$ , the corresponding maximal tolerance body  $T_{s,i}$ , inside  $A_{s,i}(x_0)$  (shaded ellipse), the volume  $V_{s,i}(x_0)$  of  $T_{s,i}$ , and the yield  $\hat{Y}_i$  according to the one-dimensional marginal distribution

$$\hat{Y}_i = \text{prob} \left\{ s \in \hat{A}_{s,i}(x_0) \right\} = \int_{-\infty}^{\beta_i} 1/\sqrt{2\pi} \cdot \exp(-t^2/2) dt \quad (16)$$

With the results  $s_i$ ,  $\beta_i$ ,  $T_{s,i}$ , and  $\hat{Y}_i$  of (12), a nominal design can be rated with respect to performance, worst-case, yield, and robustness.

Considering (2) and the properties of a pdf, it follows that (12) is equivalent to the following problem formulations

$$\max_s p d f(s) \text{ subject to } f_i^B = f_i(s, x_0) \quad (17)$$

$$\max_\beta V_s(\beta) \text{ subject to } T_s(\beta, s_0) \subset A_{s,i}(x_0) \quad (18)$$

While (18) aims at maximizing the volume of a tolerance body lying completely inside  $A_{s,i}$ , (17) defines the worst-case point  $s_i$  as the point with highest probability among the statistical parameter sets violating  $f_i^B$ .

Note, that the presented derivations are valid for various pdfs and norms, as illustrated by Fig. 3. Obviously, the classic worst-case analysis [14] is included by using the  $l_\infty$ -norm in (12).

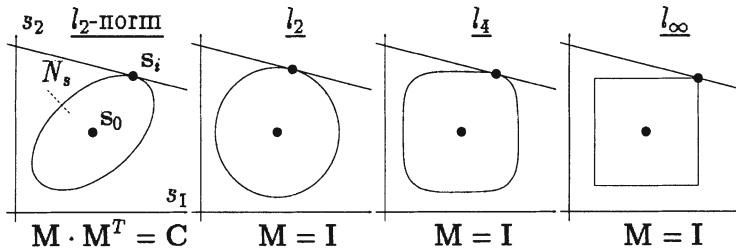


Figure 3. Points  $s_i$  on a hyperplane with smallest distance  $\beta_i = \|M^{-1} \cdot (s_i - s_0)\|$  to point  $s_0$  and associated norm bodies  $N_s$  according to different norms.

In the following, we will state the gradient of the performance distance  $\beta_i$  with respect to the nominal parameter values  $(s_0, x_0)$  for a normal pdf [2], which is essential for the solution of the circuit optimization task formulated in the next section.

The necessary conditions for the solution of (12) with the norm of (14) can be derived using the Lagrangian function of (12)

$$\mathcal{L}(s, x_0) = \beta^2 - 2\lambda \cdot (f_i(s, x_0) - f_i^B) \xrightarrow{\downarrow} \min \quad (19)$$

For the stationary point  $s_i$  of (19), the following conditions are necessary

$$f_i(s_i, x_0) = f_i^B \quad (20)$$

$$M^{-1} \cdot (s_i - s_0) = \lambda_i \cdot M^T \cdot g_{s,i} \quad (21)$$

using the abbreviations for the performance gradients

$$g_{z,i} = \partial f(z) / \partial z|_{z^T = z_i^T = [s_i^T \ x_0^T]} \quad (22)$$

$$g_{z,i}^T = [g_{c,i}^T \ g_{p,i}^T \ g_{x,i}^T] = [g_{s,i}^T \ g_{x,i}^T] = [g_{c,i}^T \ g_{a,i}^T]$$

In the course of the solution of (19), considering (20), (21), and (22), the gradient of the performance distance  $\beta_i$  with respect to the nominal values  $a_0$  of the adjustable parameters can be determined as

$$\frac{\partial \beta_i}{\partial a_0} = \frac{-\text{sign}(\lambda_i)}{\|M^T \cdot g_{s,i}\|_2} \cdot g_{a,i} \quad (23)$$

$\partial \beta_i / \partial a_0$  is a function of the performance gradient and thereby is an inherent quality of the deterministic solution of (12).

Note, that the partitioning of the parameter space is naturally included in (23).

Note also, that for the classic worst-case optimization the norm in the denominator of (23) is the  $l_1$ -norm (dual norm to  $l_\infty$ -norm).

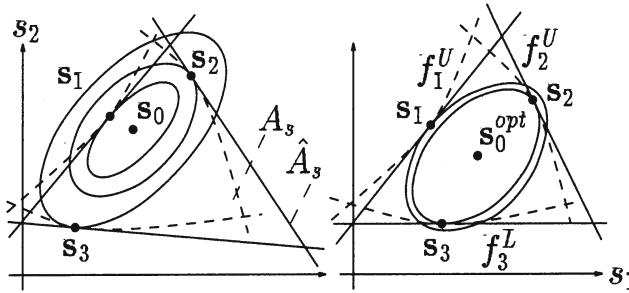


Figure 4. Acceptance region  $A_s$  spanned by three performance bounds  $f_1^U, f_2^U, f_3^U$  leading to three worst-case points  $s_1, s_2, s_3$  (left: initial design, right: optimized design).

## 5. Circuit optimization based on worst-case distances

Fig. 4 illustrates the statistical parameter space for three performance bounds. Obviously, the acceptance region  $\hat{A}_s$  can be characterized by the worst-case points  $s_i$  and the performance gradients  $g_{s,i}$  with an expense linear to the number of specs.

$$A_s = \bigcap_{i \leq n_B} A_{s,i}, \quad \hat{A}_s = \bigcap_{i \leq n_B} \hat{A}_{s,i} \quad (24)$$

We can see that no consideration of the convexity of  $A_s$  is necessary.  $\hat{A}_s$  can be used to estimate the yield without performing circuit simulations according to

$$\hat{Y} = \text{prob} \left\{ s \in \hat{A}_s(x_0) \right\} \quad (25)$$

Based on the worst-case distances and their gradients introduced in the previous section, we now formulate the problem of circuit optimization in the parameter space.

$$\max_{p_0, x_0} \beta(s_0, x_0), \quad \beta = [\dots \varphi_i(s_0, x_0) \cdot \beta_i(s_0, x_0) \dots]^T \quad (26)$$

(26) requires the maximization of all tolerance volumes with respect to all performance bounds  $f_i^B$ . As in (11), the tradeoff between the vector components of  $\beta$  has to be considered.

Thereby, (26) optimizes performance, yield, worst-case, and robustness of a nominal design.

In order to consider all these aspects, we finally formulate the problem of circuit optimization with a scalar objective function to be:

$$\min_{p_0, x_0} \gamma^T \cdot \gamma, \quad \gamma = [\dots \exp(-\varphi_i(s_0, x_0) \cdot \beta_i(s_0, x_0)) \dots]^T \quad (27)$$

From (27) we can see: when the performance bound is satisfied, the corresponding component of  $\gamma$  obtains a value between 0 and 1, when it is violated, it obtains a value greater 1. In that way, all violated specs obtain an adequate penalty

	<i>initial</i>		<i>preliminary</i>		<i>optimized</i>	
	$\varphi \cdot \beta_i$	$\hat{Y}_i$	$\varphi \cdot \beta_i$	$\hat{Y}_i$	$\varphi \cdot \beta_i$	$\hat{Y}_i$
$t_{dhl}^L$	5.36	100%	4.02	100%	3.17	100%
$t_{dhl}^U$	1.08	86%	2.41	99%	3.15	100%
$t_{dlh}^L$	6.32	100%	3.04	100%	2.79	100%
$t_{dlh}^U$	-4.45	0%	2.33	99%	2.83	100%
$\hat{Y}_{total}$	0%		99%		100%	
$t_{dhl}$	3.5		3.0		2.8	
$t_{dlh}$	7.9		3.0		2.9	
$W_p, W_n$	$4.0\mu, 4.0\mu$		$11.6\mu, 4.9\mu$		$12.4\mu, 5.4\mu$	

Table 1. CMOS inverter optimization progress.

weight, and we can start the optimization procedure with an arbitrary initial value of  $a_0$ .

(27) corresponds to a least squares optimization problem, for which efficient solution techniques have been developed.

The interactive trust-region method presented in [4], which is based on quadratic programming, is especially well-suited for circuit optimization according to (26) and (27). It is based on gradient information of the objective functions according to (23) and is especially well-suited for ill-conditioned problems. Fig. 4 shows an initial and optimized design according to (27) with equalized worst-case distances and tolerance bodies, respectively.

If the parameter distribution is not known, we set:  $C = I$  (identity matrix). Then, (27) fulfills the pure nominal optimization with respect to performance sensitivities.

## 6. Examples

The first example is an integrated CMOS inverter from a standard cell library. We will inspect the delay times  $t_{dhl}$  and  $t_{dlh}$  for the falling and rising output slope with the specification bounds  $t_{dhl}^L = t_{dlh}^L = 2$  and  $t_{dhl}^U = t_{dlh}^U = 4$ . For the underlying manufacturing process, the variations and correlations of the transistor model parameters are determined as in [13]. In total, 11 device parameters are considered, which constitute the set of statistical fixed parameters. The channel widths  $W_p$  and  $W_n$  of the p- and n-channel transistor constitute the set of deterministic adjustable parameters. Typically, in IC design there are no statistical adjustable parameters. Table 1 shows the nominal performances, worst-case distances, and estimated yields provided by the new method for the initial, a preliminary, and the optimized design according to (27). The worst-case distances of the initial indicate that  $t_{dlh}^U$  is violated ( $\varphi = -1$ ) and that the nominal point is closer to  $t_{dhl}^L$  than to  $t_{dlh}^L$ . The latter cannot be learned from the yield values which are both 100%. The performance values of a preliminary design achieved

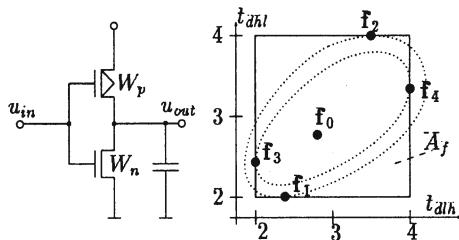


Figure 5. Performance space of CMOS inverter with two delay times, optimized design with nominal point  $f_0$  and worst-case performances  $f_i$  associated to worst-case points  $s_i$ .

during the design procedure are right between the lower and upper bounds, the corresponding yield value has reached approximately 100%. Thus, the preliminary design is optimal with respect to the yield optimization problem. However, the  $\beta_i$  show that the optimum with respect to the circuit's robustness has not been reached yet. For the optimized design, the  $\beta_i$  have been equalized. We can see that  $t_{dlh}$  is somewhat more sensitive than  $t_{dhl}$  (smaller value of  $\beta_i$ ) and that the nominal performance point is not in the middle of  $A_f$ . Fig. 5 shows the performance space for the optimized design including the worst-case performance values. A comparison with a genuine Monte Carlo analysis shows that the estimated yield values are very precise (1-2% difference).

The second example is an SC-filter, described in [3], which can be investigated by others and which is known to be very ill-conditioned. It is characterized by 12 statistical adjustable parameters (no fixed or deterministic parameters), 44 performances, and 73 spec bounds. In [17], this example had been optimized by several methods, including sophisticated Monte Carlo based methods, with a best optimization result of 60% yield (initial yield was 24%). We have applied our new method to this circuit and achieved an optimized design with the parameter values given by  $[... p_i ...] = [0.1972 \ 0.0874 \ 0.3141 \ 0.3052 \ 0.0508 \ 0.1771 \ 0.1976 \ 0.0990 \ 0.5518 \ 0.3593 \ 0.1269 \ 0.2124]$  and a yield of 64% at a total expense of 80 sensitivity analyses. Thus, the presented method obtained a better result than statistical methods at lower simulation costs.

## 7. Conclusion

In this paper, a new approach to circuit optimization has been introduced, based on worst-case distances related to the manufacturing process variations.

For each performance specification, there exists exactly one characteristic tolerance body touching the performance specification in the worst-case point. The number of worst-case points equals the number of performance specifications. The acceptance region in the parameter space can be characterized by the worst-case points with a linear expense due to the number of specs. In contrast to this,

other geometric approaches to tolerance design have a complexity exponentially increasing with the number of parameters.

The new design target “worst-case distance” measures the distance of the nominal point to the worst-case point in terms of the underlying manufacturing process. For each performance, there exists exactly one worst-case distance, which characterizes the robustness margin and the yield due to one performance spec.

A unified approach to the nominal and tolerance design problem is made possible by the worst-case points, the worst-case distances, and the related gradients. The new method for circuit design enables an extensive problem diagnosis and includes the partitioning of the parameter space in deterministic and statistical, adjustable and fixed parameters.

The new approach includes both yield optimization and design centering and can be established by a deterministic optimization procedure. Compared to the Monte Carlo analysis, equivalent optimization results are obtained at significantly lower simulation costs.

In the case that no statistical parameter distribution is given, the new method fulfills the pure nominal design task including the consideration of performance sensitivities.

## Acknowledgments

We thank Mr. Schinagel and Dr. Mueller-L. from SIEMENS Corp. for the efficacious co-operation and helpful discussion.

## References

- [1] K. J. Antreich and J. Armaos. *A general approach to statistical circuit design*. Proc. ECCTD, pages 409-412, 1983.
- [2] K. J. Antreich and H. E. Graeb. *A unified approach towards nominal and tolerance design*. Proc. IMACS World Congress on Applied Mathematics, 1991.
- [3] K. J. Antreich and R. K. Koblitz. *Design centering by yield prediction*. Trans. IEEE CAS, 29, 1982.
- [4] K. J. Antreich, P. Leibner, and F. Poernbacher. *Nominal design of integrated circuits on circuit level by an interactive improvement method*. Trans. IEEE CAS, 35:1501-1511, 1988.
- [5] J. W. Bandler and S. H. Chen. *Circuit optimization: The state of the art*. Trans. IEEE MTT, 36:424-442, 1988.
- [6] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. *A survey of optimization techniques for integrated-circuit design*. Proc. of the IEEE, 69:1334-1363, 1981.
- [7] S. W. Director, W. Maly, and A. J. Strojwas. *VLSI Design for Manufacturing: Yield Enhancement*. Kluwer Academic Publishers, USA, 1990.
- [8] P. Feldmann and S. W. Director. *Accurate and efficient evaluation of circuit yield and yield gradients*. Proc. IEEE ICCAD, pages 120-123, 1990.

- [9] D. E. Hocevar, P. F. Cox, and P. Yang. *Parametric yield optimization for MOS circuit blocks*. Trans. IEEE CAD, 7:645-658, 1988.
- [10] G. Kjellstroem and L. Taxen. *Stochastic optimization in system design*. Trans. IEEE CAS, 28:702-715, 1981.
- [11] M. R. Lightner, T. N. Trick, and R. P. Zug. *Circuit optimization and design*. In A.E. Ruehli, editor, *Circuit Analysis, Simulation and Design, Part 2*, North-Holland, 1987.
- [12] W. Maly. *Computer-aided design for VLSI circuit manufacturability*. Proc. of the IEEE, 78:356-392, 1990.
- [13] G. E. Mueller-L. *Limit-parameters: the general solution of the worst-case problem for the linearized case*. Proc. IEEE ISCAS, 1990.
- [14] H. Schjaer-Jacobsen and K. Madsen. *Algorithms for worst-case tolerance optimization*. Trans. IEEE CAS, 9:645-658, 1979.
- [15] R. Spence and R. S. Soin. *Tolerance Design of Electronic Circuits*. Addison-Wesley, England, 1988.
- [16] M. A. Styblinski and L. J. Opalski. *Algorithms and software tools for IC yield optimization based on fundamental fabrication parameters*. Trans. IEEE CAD, 5:79-89, 1986.
- [17] E. Wehrhahn and R. Spence. *The performance of some design centering methods*. Proc. IEEE ISCAS, 1984.

# VERIFYING CLOCK SCHEDULES

Thomas G. Szymanski<sup>1</sup>

*AT&T Bell Laboratories*

*Murray Hill, NJ 07974*

Narendra Shenoy<sup>2</sup>

*University of California*

*Berkeley, CA 94720*

## Abstract

Many recent papers have formulated both timing verification and optimization as mathematical programming problems. Such formulations correctly handle level-sensitive latches, long and short path considerations, and sophisticated multi-phase clocking schemes.

This paper deals with the computational aspects of using such a formulation for verifying clock schedules. We show that the formulation can have multiple solutions, and that these extraneous solutions can cause previously published algorithms to produce incorrect or misleading results. We characterize the conditions under which multiple solutions exist, and show that even when the solution is unique, the running times of these previous algorithms can be unbounded. By contrast, we exhibit a simple polynomial time algorithm for clock schedule verification. The algorithm was implemented and used to check the timing of all the circuits in the ISCAS-89 suite. Observed running times are linear in circuit size and quite practical.

## 1. Introduction

An elegant mathematical programming formulation for timing verification and clock schedule optimization was presented in [1]. The formulation, henceforth referred to as the *SMO formulation*, handles both long and short path constraints, deals correctly with both edge-triggered and level-sensitive latches, handles circuits with complex, multi-phase clocking schemes, and is sufficiently flexible to handle various system constraints that might be externally imposed upon the circuit being analyzed. It also provides a rigorous framework upon which one can discuss convexity of solution spaces, robustness of solutions, etc. For these reasons, several recent papers have built upon this model to treat more advanced timing problems including retiming [2] and wave-pipelining [3].

In the SMO formulation, the timing properties of a circuit are modeled using the parameters and variables appearing in Figure 1. In this paper, we shall deal only with level-sensitive latches, the extension to edge-triggered flip-flops being

---

Authors are currently with <sup>1</sup>Avaya Labs Research and <sup>2</sup>Synopsys.

<b>Parameters</b>	
$n$	number of latches in circuit
$p_i$	clock phase controlling latch $i$
$S_i$	setup time of latch $i$
$H_i$	hold time of latch $i$
$\delta_{i,j}$	minimum combinational delay from latch $i$ to latch $j$
$\Delta_{i,j}$	maximum combinational delay from latch $i$ to latch $j$
<b>Variables defining the clock schedule</b>	
$\pi$	clock period
$w_i$	length of time that clock phase $i$ is active
$e_i$	absolute time within period when phase $i$ begins
<b>Other variables</b>	
$E_{i,j}$	time between start of phase $i$ and <i>next</i> phase $j$
$a_i$	earliest signal arrival time at latch $i$
$A_i$	latest signal arrival time at latch $i$
$d_i$	earliest signal departure time from latch $i$
$D_i$	latest signal departure time from latch $i$
<b>Equations and constraints</b>	
$E_{i,j} = \begin{cases} e_j - e_i & \text{if } i < j \\ \pi + e_j - e_i & \text{otherwise} \end{cases}$	
$d_i = \max(a_i, \pi - w_{p_i})$	
$D_i = \max(A_i, \pi - w_{p_i})$	
$a_i = \min_{j \rightarrow i}(d_j + \delta_{j,i} - E_{p_j, p_i})$	
$A_i = \max_{j \rightarrow i}(D_j + \Delta_{j,i} - E_{p_j, p_i})$	
$a_i \geq H_i$	
$A_i \leq \pi - S_i$	

Figure 1. The SMO timing model for level-sensitive latches. It may easily be extended to accomodate edge-triggered flip-flops.

a straightforward exercise. We identify the latches of the circuit with the integers from 1 to  $n$ . The symbol  $\rightarrow$  denotes the “fans out to” relation on latches, that is,  $i \rightarrow j$  if and only if there is a path of strictly combinational logic elements extending from the output of latch  $i$  to the input of latch  $j$ . Without loss of generality, we assume that each latch has at least one fanin, that is, for every  $j$ , there exists an  $i$  such that  $i \rightarrow j$ . The term *path* henceforth means a sequence of  $k \geq 0$  latches,  $j_0, \dots, j_k$ , such that  $j_i \rightarrow j_{i+1}$  for  $0 \leq i < k$ .

The *clock schedule optimization problem* asks us to find the minimum value of  $\pi$  for which there is an assignment to all variables consistent with the constraints in Figure 1. The *clock schedule verification problem*, on the other hand, presents us with values for  $\pi$ ,  $w$  and  $e$  and asks us to find values for the rest of the

<b>Variables determined directly by clock schedule</b>
$E_{i,j} = \begin{cases} e_j - e_i & \text{if } i < j \\ \pi + e_j - e_i & \text{otherwise} \end{cases}$
$B_i = \pi - w_i$
$\lambda_{j,i} = \delta_{j,i} - E_{p_j,p_i}$
$\Lambda_{j,i} = \Delta_{j,i} - E_{p_j,p_i}$
<b>Equations defining arrival and departure times</b>
$d_i = \max(a_i, B_i)$
$D_i = \max(A_i, B_i)$
$a_i = \min_{j \rightarrow i}(d_j + \lambda_{j,i})$
$A_i = \max_{j \rightarrow i}(D_j + \Lambda_{j,i})$
<b>Constraints for correct operation</b>
$a_i \geq H_i$
$A_i \leq \pi - S_i$

Figure 2. The SMO timing model as simplified for clock schedule verification.

variables so as to satisfy the constraints. Since this paper deals exclusively with the schedule verification problem, we can simplify the SMO constraints to better suit our purpose. We introduce the auxiliary variables  $B$ ,  $\lambda$ , and  $\Lambda$  as shown in Figure 2. It is easy to see that  $B$ ,  $\lambda$ , and  $\Lambda$  are uniquely determined by the clock schedule, as is  $E$ . We may therefore recast the clock schedule verification problem as asking whether there exist  $a$ ,  $d$ ,  $A$ , and  $D$  that obey the *equations* and satisfy the *constraints* appearing in Figure 2.

The following lemma will be invoked many times throughout this paper.

**Lemma 1.1** *Let  $(a, d, A, D)$  be a solution to the equations. Let  $j_0, \dots, j_k$  be a path with  $k \geq 0$ . Then  $D_{j_k} \geq A_{j_k} \geq D_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Moreover, if  $j_0 = j_k$ , then  $0 \geq \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ .*

*Proof.* From the equations,  $D_i \geq A_i \geq D_j + \Lambda_{j,i}$  for any  $i$  and  $j$  for which  $j \rightarrow i$ . In particular,  $D_{j_k} \geq A_{j_k} \geq D_{j_{k-1}} + \Lambda_{j_{k-1}, j_k}$ . Continuing the substitution,  $D_{j_k} \geq A_{j_k} \geq D_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Setting  $D_{j_k} = D_{j_0}$  and simple manipulation complete the result.  $\square$

The following observation, although elementary, is used so frequently throughout this paper that it is worth stating explicitly.

**Fact 1.2** *The functions min and max are both monotonic in their arguments, that is, increasing the value of any argument cannot decrease the value of the function, nor can decreasing the value of any argument cause an increase in the value of the function.*

Finally, a solution  $(a^*, d^*, A^*, D^*)$  is called a *minimum* solution, if for any other solution  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$ , we have  $a_i^* \leq \tilde{a}_i$ ,  $d_i^* \leq \tilde{d}_i$ ,  $A_i^* \leq \tilde{A}_i$ , and  $D_i^* \leq \tilde{D}_i$ , for all  $i$ .

## 2. Solving the Equations

Iteration is a popular method for solving a set of equations  $V = \mathcal{E}(V)$  over a set of variables  $V = \{V_j \mid 1 \leq j \leq n\}$ . One begins with a trial “solution”  $V^0$ , and computes a succession of iterates,  $V^{i+1} = \mathcal{E}(V^i)$ , until convergence occurs,  $V^{i+1} = V^i$ , at which point  $V^i$  is a solution to  $\mathcal{E}$ . When the equations are formed from monotonic operators, as in our case, the entire sequence of iterates will be monotonic nondecreasing or nonincreasing, provided we choose the initial iterate properly. Thus, when  $V^0 \leq V^1$  componentwise (that is,  $V_j^0 \leq V_j^1$  for all  $j$ ) we are guaranteed that  $V^0 \leq V^1 \leq V^2 \dots$ . Similarly, when  $V^0 \geq V^1$ , we will have  $V^0 \geq V^1 \geq V^2 \dots$ . The monotonicity of the sequence of iterates can be used to prove convergence provided a suitable upper (or lower) bound is known for the solution.

For clock schedule verification, we can partition the equations into two independent subsets, one subset (the *earliest time equations*) involving the variables  $a_i$  and  $d_i$ , and the other subset (the *latest time equations*) involving the variables  $A_i$  and  $D_i$ . By choosing appropriate starting points for either subset, we can choose whether we converge upwards or downwards to a solution. The choice of starting point is crucial; it affects both the correctness of the result and the time needed for the process to converge. We shall show that the best approach is to converge upwards in both sets of equations to what will turn out to be the minimum fixed point of the verification equations. The rest of this section defines this technique more precisely, proves that it yields a minimum solution if any solution exists at all, and establishes that the method runs in polynomial time.

**Construction 2.1** *For a given clock schedule, compute  $E$ ,  $B$ ,  $\lambda$ ,  $\Lambda$  as described above. Then, for all  $m \geq 0$ , define*

$$\begin{aligned} a_i^0 &= -\infty \\ A_i^0 &= -\infty \\ a_i^m &= \min_{j \rightarrow i} (d_j^{m-1} + \lambda_{j,i}) \quad \text{if } m > 0 \\ A_i^m &= \max_{j \rightarrow i} (D_j^{m-1} + \Lambda_{j,i}) \quad \text{if } m > 0 \\ d_i^m &= \max(a_i^m, B_i) \\ D_i^m &= \max(A_i^m, B_i). \end{aligned}$$

It is easy to see that this process is well defined. Simply compute  $A^0$  and  $a^0$ , then  $D^0$  and  $d^0$ , then  $A^1$  and  $a^1$ , etc. The values of the variables turn out to be monotonic nondecreasing in  $m$ , as shown in the following lemma.

**Lemma 2.2 (Monotonicity)** *For all  $i$  and  $m \geq 1$ ,  $a_i^m \geq a_i^{m-1}$ ,  $d_i^m \geq d_i^{m-1}$ ,  $A_i^m \geq A_i^{m-1}$ , and  $D_i^m \geq D_i^{m-1}$ .*

*Proof.* By induction on  $m$ . We will only provide the argument for  $a_i^m$  and  $d_i^m$ , the argument for  $A_i^m$  and  $D_i^m$  being similar. For the basis with  $m = 1$ , we have  $a_i^1 \geq a_i^0$  because  $a_i^0 = -\infty$ . Moreover,  $d_i^0 = B_i$ , and  $d_i^1 \geq B_i$ , so  $d_i^1 \geq d_i^0$ .

For the inductive step, assume the lemma is true for  $m - 1$  with  $m > 1$ . By the monotonicity of min and the inductive assertion, we have  $\min_{j \rightarrow i}(d_j^{m-1} + \lambda_{j,i}) \geq \min_{j \rightarrow i}(d_j^{m-2} + \lambda_{j,i})$ . By definition,  $a_i^m = \min_{j \rightarrow i}(d_j^{m-1} + \lambda_{j,i})$  and  $a_i^{m-1} = \min_{j \rightarrow i}(d_j^{m-2} + \lambda_{j,i})$ , so  $a_i^m \geq a_i^{m-1}$ . The monotonicity of max then implies that  $\max(a_i^m, B_i) \geq \max(a_i^{m-1}, B_i)$ , and so  $d_i^m \geq d_i^{m-1}$ .  $\square$

The next lemma shows that the iterated variables approach a solution from below. As a consequence, if the process converges, it will converge to a solution which is a minimum solution.

**Lemma 2.3** *Let  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$  be a solution to the equations. Then for all  $i$  and  $m \geq 0$ ,  $a_i^m \leq \tilde{a}_i$ ,  $d_i^m \leq \tilde{d}_i$ ,  $A_i^m \leq \tilde{A}_i$ , and  $D_i^m \leq \tilde{D}_i$ .*

*Proof.* By induction on  $m$  using an argument nearly identical to Lemma 2.2. We will only provide the argument for  $a_i^m$  and  $d_i^m$ , the argument for  $A_i^m$  and  $D_i^m$  being similar. For the basis with  $m = 0$ , we have  $a_i^0 \leq \tilde{a}_i$  because  $a_i^0 = -\infty$ . Moreover,  $d_i^0 = B_i$ , and  $\tilde{d}_i \geq B_i$  directly from the equations, and so  $d_i^0 \leq \tilde{d}_i$ .

For the inductive step, assume the lemma is true for  $m - 1$ . By the monotonicity of min and the inductive assertion, we have  $\min_{j \rightarrow i}(d_j^{m-1} + \lambda_{j,i}) \leq \min_{j \rightarrow i}(\tilde{d}_j + \lambda_{j,i})$ . By definition,  $a_i^m = \min_{j \rightarrow i}(d_j^{m-1} + \lambda_{j,i})$  and  $\tilde{a}_i = \min_{j \rightarrow i}(\tilde{d}_j + \lambda_{j,i})$ , so  $a_i^m \leq \tilde{a}_i$ . The monotonicity of max then implies that  $\max(a_i^m, B_i) \leq \max(\tilde{a}_i, B_i)$ , and so  $d_i^m \leq \tilde{d}_i$ .  $\square$

As the iteration proceeds, the values of the variables increase. The next lemma will be used to relate each such change to some latch which is “to blame.”

**Lemma 2.4** *For any  $i$ , if  $m \geq 1$  and  $d_i^m > d_i^{m-1}$ , then there exists a  $j$  such that  $j \rightarrow i$ , and  $d_i^m = a_i^m \leq d_j^{m-1} + \lambda_{j,i}$ . Moreover, if  $m > 1$ , then  $d_j^{m-1} > d_j^{m-2}$ .*

*Proof.* Since  $d_i^m > d_i^{m-1}$  by hypothesis,

$$d_i^m = \max(a_i^m, B_i) > \max(a_i^{m-1}, B_i) = d_i^{m-1}.$$

Lemma 2.2 then implies that  $d_i^m = a_i^m$  and  $a_i^m > a_i^{m-1}$ .

Case 1:  $m = 1$ . By definition,  $a_i^1 = \min_{p \rightarrow i}(d_p^0 + \lambda_{p,i})$ . Pick  $j$  such that  $j \rightarrow i$  and  $d_j^0 + \lambda_{j,i} = \min_{p \rightarrow i}(d_p^0 + \lambda_{p,i})$ . Then  $a_i^1 \leq d_j^0 + \lambda_{j,i}$ , as required.

Case 2:  $m > 1$ . Since  $a_i^m > a_i^{m-1}$ , we have  $\min_{p \rightarrow i}(d_p^{m-1} + \lambda_{p,i}) > \min_{p \rightarrow i}(d_p^{m-2} + \lambda_{p,i})$ . Clearly, there must exist at least one  $p$  for which  $p \rightarrow i$

and  $d_p^{m-1} > d_p^{m-2}$ . Take  $j = p$ . Then  $a_i^m = \min_{p \rightarrow i}(d_p^{m-1} + \lambda_{p,i}) \leq d_j^{m-1} + \lambda_{j,i}$ , satisfying the lemma.  $\square$

The next lemma is the key technical result of this section. It shows that the iterations need not be continued indefinitely. More specifically, it shows that if the earliest time equations do not converge within  $n$  iterations, then the latest time equations do not converge at all.

**Lemma 2.5** *If  $d_i^n \neq d_i^{n-1}$  for some  $i$ , then the equations have no solution.*

*Proof.* By Lemma 2.2,  $d_i^n > d_i^{n-1}$ . Construct a path  $j_0, \dots, j_n$  by setting  $j_n = i$  and then applying Lemma 2.4  $n$  times to obtain  $j_{n-1}, \dots, j_0$ . Then, for each  $k$ ,  $0 \leq k < n$ , we have  $j_k \rightarrow j_{k+1}$  and  $d_{j_{k+1}}^{k+1} \leq d_{j_k}^k + \lambda_{j_k, j_{k+1}}$ . Moreover, for  $1 \leq k < n$ , we have  $d_{j_k}^k > d_{j_k}^{k-1}$ . Accordingly, for any  $p$  and  $q$  with  $p < q$ , we have  $d_{j_q}^q \leq d_{j_p}^p + \sum_{k=p}^{q-1} \lambda_{j_k, j_{k+1}}$ . We also have  $d_{j_q}^q > d_{j_q}^{q-1}$ , which implies by Lemma 2.2 that  $d_{j_q}^q > d_{j_q}^p$ . Since each  $j_i$  is an integer between 1 and  $n$  inclusive, we can pick  $p$  and  $q$  so that  $j_p = j_q$ . Thus  $d_{j_q}^q \leq d_{j_q}^p + \sum_{k=p}^{q-1} \lambda_{j_k, j_{k+1}}$ . Hence,  $0 < \sum_{k=p}^{q-1} \lambda_{j_k, j_{k+1}}$ , implying that  $0 < \sum_{k=p}^{q-1} \Lambda_{j_k, j_{k+1}}$ . However, Lemma 1.1 applied to the subsequence  $j_p, \dots, j_q$ , says that if a solution exists, we must have  $0 \geq \sum_{k=p}^{q-1} \Lambda_{j_k, j_{k+1}}$ . Therefore the equations are unsolvable.  $\square$

The next two lemmas do for the variables  $A$  and  $D$  what the previous two lemmas did for  $a$  and  $d$ . Note that the statements of the lemmas are very close, but not identical, to the earlier lemmas.

**Lemma 2.6** *For any  $i$ , if  $m \geq 1$  and  $D_i^m > D_i^{m-1}$ , then there exists a  $j$  such that  $j \rightarrow i$ , and  $D_i^m = A_i^m = D_j^{m-1} + \Lambda_{j,i}$ . Moreover, if  $m > 1$ , then  $D_j^{m-1} > D_j^{m-2}$ .*

*Proof.* Similar to Lemma 2.4.  $\square$

**Lemma 2.7** *If  $D_i^n \neq D_i^{n-1}$  for some  $i$ , then the equations have no solution.*

*Proof.* Similar to Lemma 2.5.  $\square$

Next we show that Construction 2.1 converges to a solution if it converges at all.

**Lemma 2.8** *Let  $p$  and  $q$  be integers such that  $d_i^{p-1} = d_i^p$  for every  $i$ , and  $D_i^{q-1} = D_i^q$  for every  $i$ . Then  $(a^p, d^p, A^q, D^q)$  is a minimum solution to the equations.*

*Proof.* It is trivial to verify that  $(a^p, d^p, A^q, D^q)$  satisfies the equations. Lemma 2.3 shows that the solution is a minimum solution.  $\square$

Finally we arrive at the key result of this section, namely, a polynomial time algorithm for solving the equations.

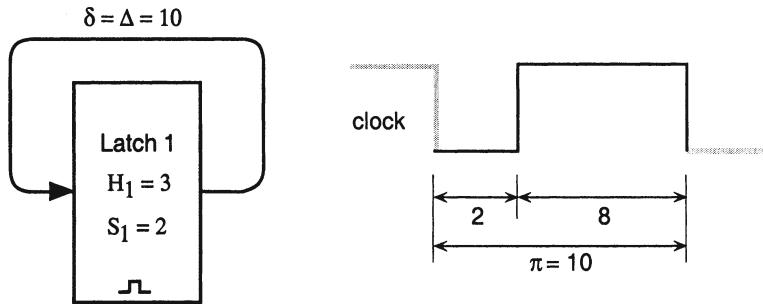


Figure 3. A circuit and clock schedule for which the SMO equations have multiple solutions.

**Theorem 2.9** *If the equations have a solution, then they have a minimum solution. Moreover, this minimum solution can be found in  $O(n^3)$  time.*

*Proof.* Clearly we can compute the vectors  $d^n$  and  $D^n$  in  $O(n^3)$  time. In  $O(n)$  additional time, we can apply Lemmas 2.5 and 2.7 to test whether any solution exists. If one does, then  $(a^n, d^n, A^n, D^n)$  is a solution. Moreover, by Lemma 2.8, it is a minimum solution.  $\square$

The reader will note that our suggested solution to the latest time equations is equivalent to the Bellman-Ford algorithm for finding shortest paths in a directed graph with positive and negative edge weights. Ishii [4] has previously advocated this approach as well.

### 3. Uniqueness

The algorithm of the previous section can be used to find a solution to the equations. Clock schedule verification can then be performed by checking whether the solution obeys the constraints on setup and hold times listed in Figure 2. This of course raises an important question, namely, what if the equations have multiple solutions? Some of the solutions might satisfy the constraints, while other solutions violate them. For example, consider the circuit and clock schedule shown in Figure 3. The corresponding SMO equations and constraints become

$$\begin{array}{ll} \lambda_{1,1} = 0 & \Lambda_{1,1} = 0 \\ a_1 = d_1 & A_1 = D_1 \\ d_1 = \max(a_1, 2) & D_1 = \max(A_1, 2) \\ a_1 \geq 3 & A_1 \leq 8. \end{array}$$

For any  $x \geq 2$ ,  $A_1 = D_1 = a_1 = d_1 = x$  is a valid solution to these equations. Solutions having  $x < 3$  exhibit a hold violation, solutions having  $x > 8$  exhibit a setup violation, and solutions with  $3 \leq x \leq 8$  are free of violations.

Thus we see that solutions to the SMO equations are not, in general, unique, and different solutions can give differing results vis-a-vis the timing constraints.

In this section, we will characterize the circumstances under which multiple solutions exist. More specifically, we will show that multiple solutions can only exist at the optimal clock period, and are due to cycles of zero aggregate delay in the circuit.

Throughout this section, we will use  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$  to denote an arbitrary solution to the equations, and  $(a^*, d^*, A^*, D^*)$  to denote the minimum solution to the equations. We begin with a pair of lemmas needed in the proof of the theorem which is to follow.

**Lemma 3.1** *If  $\tilde{d}_i > d_i^*$ , then there exists a  $j$  such that  $\tilde{d}_j > d_j^*$ ,  $j \rightarrow i$  and  $\tilde{d}_i \leq \tilde{d}_j + \lambda_{j,i}$ .*

*Proof.* By hypothesis,  $\tilde{d}_i = \max(\tilde{a}_i, B_i) > \max(a_i^*, B_i) = d_i^*$ , implying  $\tilde{a}_i > a_i^*$ . Thus,  $\tilde{a}_i = \min_{p \rightarrow i}(\tilde{d}_p + \lambda_{p,i}) > \min_{p \rightarrow i}(d_p^* + \lambda_{p,i}) = a_i^*$ . Hence we can find a  $j$  such that  $j \rightarrow i$  and  $\tilde{d}_j > d_j^*$ . Moreover,  $\tilde{a}_i \leq \tilde{d}_j + \lambda_{j,i}$ .  $\square$

**Lemma 3.2** *If  $\tilde{D}_i > D_i^*$ , then there exists a  $j$  such that  $\tilde{D}_j > D_j^*$ ,  $j \rightarrow i$  and  $\tilde{D}_i = \tilde{D}_j + \Lambda_{j,i}$ .*

*Proof.* Similar to Lemma 3.1. By hypothesis,  $\tilde{D}_i = \max(\tilde{A}_i, B_i) > \max(A_i^*, B_i) = D_i^*$ , implying  $\tilde{A}_i > A_i^*$ . Thus,  $\tilde{A}_i = \max_{p \rightarrow i}(\tilde{D}_p + \Lambda_{p,i}) > \max_{p \rightarrow i}(D_p^* + \Lambda_{p,i}) = A_i^*$ . Take  $j$  such that  $j \rightarrow i$  and  $\tilde{D}_j + \Lambda_{j,i} = \max_{p \rightarrow i}(\tilde{D}_p + \Lambda_{p,i})$ . Certainly  $\tilde{D}_j > D_j^*$ . Moreover,  $\tilde{A}_i = \tilde{D}_j + \Lambda_{j,i}$ .  $\square$

We are now ready to show that multiple solutions to the equations are due to the presence of zero weight cycles in the circuit. Such a cycle represents a path through the circuit in which a signal returns to some latch at precisely the same time (relative to the clock period) as it left that latch.

**Theorem 3.3** *If the equations have more than one solution, then there exists a path  $j_0, \dots, j_k$ ,  $k \geq 1$ , such that  $j_0 = j_k$  and  $0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ .*

*Proof.* Let  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$  be a solution other than the minimum solution  $(a^*, d^*, A^*, D^*)$ .

Case 1:  $\tilde{d}_m > d_m^*$  for some  $m$ . Construct a path  $j_0, \dots, j_n$  by setting  $j_n = m$  and then applying Lemma 3.1  $n$  times to obtain  $j_{n-1}, \dots, j_0$ . Then, for each  $i$ ,  $0 \leq i < n$ , we have  $j_i \rightarrow j_{i+1}$  and  $\tilde{d}_{j_{i+1}} \leq \tilde{d}_{j_i} + \lambda_{j_i, j_{i+1}}$ . Clearly, this path must contain repeated indices, so pick  $p$  and  $q$  such that  $p < q$  and  $j_p = j_q$ . The path that will satisfy the lemma is  $j_p, \dots, j_q$ . Substituting,  $\tilde{d}_{j_q} \leq \tilde{d}_{j_p} + \sum_{i=p}^{q-1} \lambda_{j_i, j_{i+1}}$ . Thus  $0 \leq \sum_{i=p}^{q-1} \lambda_{j_i, j_{i+1}} \leq \sum_{i=p}^{q-1} \Lambda_{j_i, j_{i+1}}$ . By Lemma 1.1, we conclude that  $0 = \sum_{i=p}^{q-1} \Lambda_{j_i, j_{i+1}}$ .

Case 2:  $\tilde{D}_m > D_m^*$  for some  $m$ . Use Lemma 3.2 to construct a path  $j_0, \dots, j_n$  with  $j_n = m$ , such that  $j_i \rightarrow j_{i+1}$  and  $\tilde{D}_{j_{i+1}} = \tilde{D}_{j_i} + \Lambda_{j_i, j_{i+1}}$  for each  $i$ ,  $0 \leq i < n$ .

Pick  $p$  and  $q$  such that  $p < q$  and  $j_p = j_q$ . Now consider the subpath  $j_p, \dots, j_q$ . Clearly,  $\tilde{D}_{j_q} = \tilde{D}_{j_p} + \sum_{i=p}^{q-1} \Lambda_{j_i, j_{i+1}}$ , and we have  $0 = \sum_{i=p}^{q-1} \Lambda_{j_i, j_{i+1}}$ .

Case 3:  $\tilde{d}_m = d_m^*$  and  $\tilde{D}_m = D_m^*$  for every  $m$ . This implies that  $\tilde{a}_m = a_m^*$  and  $\tilde{A}_m = A_m^*$  for every  $m$ , in contradiction to the assumption that the two solutions are distinct.  $\square$

An immediate consequence is that non-unique solutions are a phenomenon that can only occur at the optimal clock period. Said another way, the equations have a unique solution whenever the clock schedule has a suboptimal period.

**Corollary 3.4** *If the equations have more than one solution for a clock schedule with period  $\pi$ , then  $\pi$  is optimal, that is, no valid schedule has a period less than  $\pi$ .*

*Proof.* Let  $j_0, \dots, j_k$  fulfill the conditions of Theorem 3.3. Then

$$0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}} = \sum_{i=0}^{k-1} \Delta_{j_i, j_{i+1}} - \sum_{i=0}^{k-1} E_{p_{j_i}, p_{j_{i+1}}},$$

which may be further rewritten as

$$0 = \sum_{i=0}^{k-1} \Delta_{j_i, j_{i+1}} - \sum_{i=0}^{k-1} (e_{p_{j_{i+1}}} - e_{p_{j_i}}) - c\pi,$$

where  $c$  is the number of  $i$ ,  $0 \leq i < k$ , for which  $p_{j_{i+1}} \leq p_{j_i}$ . Observing that

$$\sum_{i=0}^{k-1} (e_{p_{j_{i+1}}} - e_{p_{j_i}}) = \sum_{i=1}^k e_{p_{j_i}} - \sum_{i=0}^{k-1} e_{p_{j_i}} = e_{p_{j_k}} - e_{p_{j_0}} = 0,$$

we can conclude that

$$0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}} = \sum_{i=0}^{k-1} \Delta_{j_i, j_{i+1}} - c\pi.$$

Reducing  $\pi$  would certainly violate Lemma 1.1, and so no solutions exist for smaller values of  $\pi$ .  $\square$

Having shown that zero-weight cycles are a necessary condition for multiple solutions to the SMO equations, we shall next show that they are also sufficient. We'll do this by showing how such a cycle in a solution can be used to construct other solutions as well. The basic idea is to add some  $\varepsilon$  to all the  $A_i$ 's and  $D_i$ 's along the cycle. In general, this alone will not yield a solution. However, the "perturbed" values can be used to reinitialize the algorithm of §2, which can then be used to converge upwards to a new solution. Before proceeding, we note a consequence of the previous theorem.

**Corollary 3.5** Let  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$  be a solution to the equations and let  $j_0, \dots, j_k$  be as in Theorem 3.3. Then  $\tilde{D}_{j_{i+1}} = \tilde{A}_{j_{i+1}} = \tilde{D}_{j_i} + \Lambda_{j_i, j_{i+1}}$ , for  $0 \leq i < k$ .

*Proof.* First, suppose that some  $D_{j_i} \neq A_{j_i}$ . Renumbering the  $j$ 's if necessary, assume that it is  $D_{j_k} \neq A_{j_k}$ . By Lemma 1.1,  $D_{j_k} \geq A_{j_k} \geq D_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}} = D_{j_0} = D_{j_k}$ . This clearly implies that  $D_{j_k} = A_{j_k}$ .

Second, suppose that some  $D_{j_i} = A_{j_i} \neq D_{j_{i-1}} + \Lambda_{j_{i-1}, j_i}$ . Once again, renumbering if necessary, assume that  $D_{j_1} = A_{j_1} \neq D_{j_0} + \Lambda_{j_0, j_1}$ . By Lemma 1.1,  $D_{j_k} \geq D_{j_1} + \sum_{i=1}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Thus  $D_{j_k} > D_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}} = D_{j_0}$ . This is impossible since  $j_k = j_0$ .  $\square$

**Construction 3.6** Suppose that  $(\tilde{a}, \tilde{d}, \tilde{A}, \tilde{D})$  is a solution to the equations and that  $j_0, \dots, j_k$ ,  $k \geq 1$ , is a path for which  $j_0 = j_k$  and  $0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Let  $C$  be the set  $\{j_i \mid 0 \leq i < k\}$  and let  $\varepsilon$  be any positive real. For all  $m \geq 0$ , define

$$\begin{aligned}\hat{A}_i^0 &= \begin{cases} \tilde{A}_i + \varepsilon & \text{if } i \in C \\ \tilde{A}_i & \text{otherwise} \end{cases} \\ \hat{D}_i^m &= \max(\hat{A}_i^m, B_i) & \text{if } m \geq 0 \\ \hat{A}_i^m &= \max_{j \rightarrow i}(\hat{D}_j^{m-1} + \Lambda_{j,i}) & \text{if } m \geq 1.\end{aligned}$$

**Lemma 3.7**

$$\hat{D}_i^0 = \begin{cases} \hat{A}_i^0 = \tilde{D}_i + \varepsilon = \tilde{A}_i + \varepsilon & \text{if } i \in C \\ \tilde{D}_i & \text{if } i \notin C \end{cases}$$

*Proof.* Suppose that  $i \in C$ . Corollary 3.5 tells us  $\tilde{D}_i = \max(\tilde{A}_i, B_i) = \tilde{A}_i \geq B_i$ . Hence,  $\hat{D}_i^0 = \max(\hat{A}_i^0, B_i) = \max(\tilde{A}_i + \varepsilon, B_i) = \tilde{A}_i + \varepsilon$ . Since  $\tilde{D}_i = \tilde{A}_i$ , we have  $\hat{D}_i^0 = \tilde{D}_i + \varepsilon$ .

Next suppose that  $i \notin C$ . Then  $\hat{D}_i^0 = \max(\hat{A}_i^0, B_i) = \max(\tilde{A}, B_i) = \tilde{D}_i$ .  $\square$

The next several lemmas show that the iterations performed on the perturbed solution yield monotonically non-decreasing values, must converge to another solution within  $n$  steps, and does not affect the  $A_i$  and  $D_i$  values for those latches  $i$  that are on the cycle  $C$ .

**Lemma 3.8** For all  $i$  and  $m \geq 1$ ,  $\hat{A}_i^m \geq \hat{A}_i^{m-1}$ , and  $\hat{D}_i^m \geq \hat{D}_i^{m-1}$ .

*Proof.* By induction on  $m$ . For the basis,  $m = 1$ . First consider  $\hat{A}_i^1$ . There are two cases, depending on whether  $i$  lies on the cycle.

Case 1:  $i \notin C$ . Lemma 3.7 tells us that  $\hat{D}_j^0 \geq \tilde{D}_j$  for all  $j$ . Then  $\hat{A}_i^1 = \max_{j \rightarrow i}(\hat{D}_j^0 + \Lambda_{j,i}) \geq \max_{j \rightarrow i}(\tilde{D}_j + \Lambda_{j,i}) = \tilde{A}_i = \hat{A}_i^0$ .

Case 2:  $i \in C$ . By Corollary 3.5, there exists some  $p \in C$  such that

$$\tilde{A}_i = \max_{j \rightarrow i}(\tilde{D}_j + \Lambda_{j,i}) = \tilde{D}_p + \Lambda_{p,i}.$$

By Lemma 3.7,

$$\hat{A}_i^1 = \max_{j \rightarrow i} (\hat{D}_j^0 + \Lambda_{j,i}) \geq \hat{D}_p^0 + \Lambda_{p,i}, i = \tilde{D}_p + \varepsilon + \Lambda_{p,i}.$$

Together, these imply that

$$\hat{A}_i^1 \geq \tilde{A}_i + \varepsilon = \hat{A}_i^0,$$

as was to be shown.

Having established that  $\hat{A}_i^1 \geq \hat{A}_i^0$ , it is easy to see that  $\hat{D}_i^1 = \max(\hat{A}_i^1, B_i) \geq \max(\hat{A}_i^0, B_i) = \hat{D}_i^0$  by the monotonicity of max. Thus  $\hat{D}_i^1 \geq \hat{D}_i^0$ .

For the inductive step, assume the lemma is true for  $m - 1$  with  $m > 1$ . By the monotonicity of max and the inductive assertion, we have  $\max_{j \rightarrow i} (\hat{D}_j^{m-1} + \Lambda_{j,i}) \geq \max_{j \rightarrow i} (\hat{D}_j^{m-2} + \Lambda_{j,i})$ . Recall that  $\hat{A}_i^m = \max_{j \rightarrow i} (\hat{D}_j^{m-1} + \Lambda_{j,i})$  and  $\hat{A}_i^{m-1} = \max_{j \rightarrow i} (\hat{D}_j^{m-2} + \Lambda_{j,i})$ , so substitution gives  $\hat{A}_i^m \geq \hat{A}_i^{m-1}$ . The monotonicity of max then implies that  $\max(\hat{A}_i^m, B_i) \geq \max(\hat{A}_i^{m-1}, B_i)$ , and so  $\hat{D}_i^m \geq \hat{D}_i^{m-1}$  as was to be shown.  $\square$

**Lemma 3.9** *For any  $i$ , if  $m \geq 1$  and  $\hat{D}_i^m > \hat{D}_i^{m-1}$ , then there exists a  $j$  such that  $j \rightarrow i$ , and  $\hat{D}_i^m = \hat{A}_i^m = \hat{D}_j^{m-1} + \Lambda_{j,i}$ . Moreover, if  $m > 1$ , then  $\hat{D}_j^{m-1} > \hat{D}_j^{m-2}$ .*

*Proof.* Analogous to Lemma 2.6. Since  $\hat{D}_i^m > \hat{D}_i^{m-1}$  by hypothesis,  $\hat{D}_i^m = \max(\hat{A}_i^m, B_i) > \max(\hat{A}_i^{m-1}, B_i) = \hat{D}_i^{m-1}$ . Lemma 3.8 then implies that  $\hat{D}_i^m = \hat{A}_i^m$  and  $\hat{A}_i^m > \hat{A}_i^{m-1}$ .

Case 1:  $m = 1$ . By definition,  $\hat{A}_i^1 = \max_{p \rightarrow i} (\hat{D}_p^0 + \Lambda_{p,i})$ . Pick any  $j$  such that  $j \rightarrow i$  and  $\hat{D}_j^0 + \Lambda_{j,i} = \max_{p \rightarrow i} (\hat{D}_p^0 + \Lambda_{p,i})$ . Hence  $\hat{A}_i^1 = \hat{D}_j^0 + \Lambda_{j,i}$ .

Case 2:  $m > 1$ . Since  $\hat{A}_i^m > \hat{A}_i^{m-1}$ , we have  $\max_{p \rightarrow i} (\hat{D}_p^{m-1} + \Lambda_{p,i}) > \max_{p \rightarrow i} (\hat{D}_p^{m-2} + \Lambda_{p,i})$ . Pick any  $j$  be such that  $j \rightarrow i$  and  $\hat{D}_j^{m-1} + \Lambda_{j,i} = \max_{p \rightarrow i} (\hat{D}_p^{m-1} + \Lambda_{p,i})$ . By our choice of  $j$ ,  $\hat{D}_i^m = \hat{A}_i^m = \hat{D}_j^{m-1} + \Lambda_{j,i}$ . Moreover,  $\hat{D}_j^{m-1} > \hat{D}_j^{m-2}$ .  $\square$

**Lemma 3.10** *For any  $i \in C$ , if  $m \geq 1$  then  $\hat{D}_i^m = \tilde{D}_i + \varepsilon$ .*

*Proof.* Since  $\tilde{D}_i + \varepsilon = \hat{D}_i^0$  by Lemma 3.7, it suffices to show that  $\hat{D}_i^m = \hat{D}_i^0$  for all  $m$ . Suppose the contrary, that is, suppose that  $\hat{D}_i^m > \hat{D}_i^0$  for some  $m \geq 1$ . Accordingly, there is some  $k \geq 1$  such that  $\hat{D}_i^k > \hat{D}_i^{k-1}$ . Apply Lemma 3.9 to construct a path  $j_0, \dots, j_k$  such that

$$\hat{D}_i^k = \hat{D}_{j_k}^k = \hat{D}_{j_0}^0 + \sum_{p=0}^{k-1} \Lambda_{j_p, j_{p+1}}.$$

Moreover, Lemmas 3.8 and 3.7 tell us that  $\hat{D}_i^k > \hat{D}_i^0 = \tilde{D}_i + \epsilon$ , and so we have,

$$\tilde{D}_i + \epsilon < \hat{D}_{j_0}^0 + \sum_{p=0}^{k-1} \Lambda_{j_p, j_{p+1}}.$$

Next, apply Lemma 1.1 to path  $j_0, \dots, j_k$  to see that

$$\tilde{D}_i \geq \tilde{D}_{j_0} + \sum_{p=0}^{k-1} \Lambda_{j_p, j_{p+1}}.$$

Thus,  $\hat{D}_{j_0}^0 > \tilde{D}_{j_0} + \epsilon$ , contradicting Lemma 3.7.  $\square$

**Theorem 3.11 (converse of Theorem 3.3)** *Let  $j_0, \dots, j_k$ ,  $k \geq 1$ , be a path such that  $j_0 = j_k$  and  $0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Then the equations either have no solutions, or else they have infinitely many solutions, and the solutions are unboundedly large.*

*Proof.* If the equations have any solution at all, then we can apply Construction 3.6 to that solution. We claim that  $\hat{D}_i^n = \hat{D}_i^{n-1}$  for all  $i$ . It is easy to show this using an argument along the same lines as the proof of Lemma 2.7. That is, assume the contrary, apply Lemma 3.9 to produce a path passing through the same node twice, and from there show the existence of a cycle of positive weight contradicting Lemma 1.1.

It is now straightforward to verify that  $(\tilde{a}, \tilde{d}, \hat{A}^n, \hat{D}^n)$  is a solution to the equations. Moreover, each value of  $\epsilon$  used in Construction 3.6 gives rise to a different solution, as readily seen from Lemma 3.10. We conclude that there are infinitely many solutions to the equations.  $\square$

**Corollary 3.12** *If there are more than one solution to the equations, then there are solutions that violate the setup constraints.*

*Proof.* Simply pick  $\epsilon$  large enough.  $\square$

**Theorem 3.13** *The uniqueness of a solution can be determined in  $O(n^2)$  time.*

*Proof.* Let  $(\tilde{a}, \tilde{d}, \hat{A}, \tilde{D})$  be a solution to the equations. Define the relation *determines* (symbolized  $\triangleright$ ) by  $j \triangleright i$  if and only if  $j \rightarrow i$  and  $\tilde{D}_i = \tilde{D}_j + \Lambda_{j,i}$ . We claim that the equations have multiple solutions if and only if  $\triangleright$  is cyclic, that is, if and only if there exists some  $i$  for which  $i \triangleright^+$   $i$ .

For the “if” part, suppose that  $\triangleright$  is cyclic. Then we can find a sequence  $j_0, \dots, j_k$  with  $k \geq 1$ , such that  $j_0 = j_k$  and  $j_i \triangleright j_{i+1}$  for each  $i$  with  $0 \leq i < k$ . By definition of  $\triangleright$ , this means that  $\tilde{D}_{j_{i+1}} = \tilde{D}_{j_i} + \Lambda_{j_i, j_{i+1}}$  for  $0 \leq i < k$ . Substituting, we have  $\tilde{D}_{j_k} = \tilde{D}_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Since  $j_0 = j_k$ , it must be that

$0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . Since we hypothesized that the equations have at least one solution, Theorem 3.11 tells us that multiple solutions exist.

For the “only if” part, suppose that the equations have multiple solutions. By Theorem 3.3, there exists a path,  $j_0, \dots, j_k$ ,  $k \geq 1$ , such that  $j_0 = j_k$  and  $0 = \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . We shall show that  $j_i \triangleright j_{i+1}$  for each  $i$  with  $0 \leq i < k$ , by assuming the contrary and deriving a contradiction. Accordingly, after renumbering the  $j$ ’s if necessary, assume that  $j_{k-1} \not\triangleright j_k$ . Then  $\tilde{D}_{j_k} \neq \tilde{D}_{j_{k-1}} + \Lambda_{j_{k-1}, j_k}$ . Since the equations require  $\tilde{D}_{j_k} \geq \tilde{D}_{j_{k-1}} + \Lambda_{j_{k-1}, j_k}$ , we must have  $\tilde{D}_{j_k} > \tilde{D}_{j_{k-1}} + \Lambda_{j_{k-1}, j_k}$ . If we apply Lemma 1.1 to the path  $j_0, \dots, j_{k-1}$ , we see that  $\tilde{D}_{j_{k-1}} \geq \tilde{D}_{j_0} + \sum_{i=0}^{k-2} \Lambda_{j_i, j_{i+1}}$ . Combining these, we get  $\tilde{D}_{j_k} > \tilde{D}_{j_0} + \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ . This implies that  $0 > \sum_{i=0}^{k-1} \Lambda_{j_i, j_{i+1}}$ , in contradiction to our earlier statement that this sum was exactly 0. We conclude that  $j_0 \stackrel{+}{\triangleright} j_0$  and hence that  $\triangleright$  is cyclic.

Having established the claim, we can present the desired algorithm. Given a solution, construct the  $\triangleright$  relation in  $O(n^2)$  time. Then use a topological sort algorithm or a strong components algorithm to determine whether  $\triangleright$  is cyclic.  $\square$

Having determined that the SMO equations might have multiple solutions, we are faced with a question of interpretation, namely, which is the “right” solution? At least two viewpoints are possible.

One viewpoint is that the minimum fixed point of the equations is the preferred solution because this is what the actual circuit “does” when started from an arbitrary stable initial state. When the circuit first begins to be clocked, signals leave latches at the opening edge of the latches. As operation continues, departure times get pushed out later and later into the transparent interval of the latches. Eventually the circuit reaches a steady state and the arrival and departure times stay constant. Like Construction 2.1, the actual circuit begins with arrival and departure times as small as possible and converges upwards to an equilibrium position. Indeed, we can interpret variable  $A_i^m$  in that construction as the latest arrival time at latch  $i$  of any signal that began at the opening edge of some latch and subsequently passed through  $m$  or fewer latches during their transparent phases.

Another viewpoint is that it is not permissible to operate a circuit under a clock schedule that has multiple solutions. The rationale here is that each solution represents a possible operating point of the circuit, and all the operating points are in equilibrium with each other. The slightest perturbation of a delay or of the arrival of a clock signal (possibly caused by a power supply fluctuation or other environmental effect) can cause the circuit to switch from operating point to another. Since we know that some of the solutions violate setup requirements, we conclude that it is too dangerous to operate the circuit this close to the “edge.” Said another way, there is no margin for error under such a schedule.

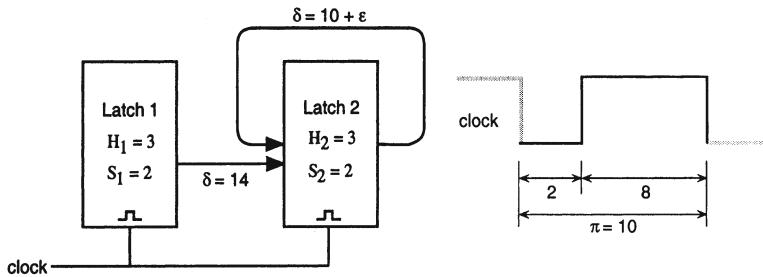


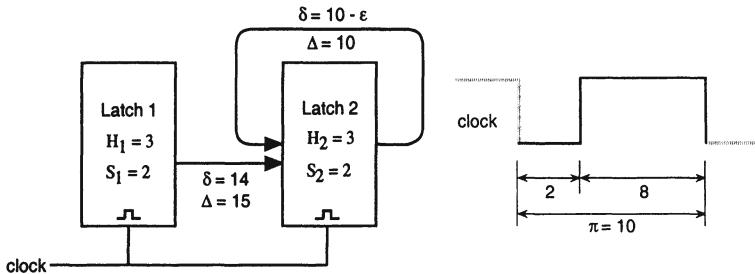
Figure 4. A circuit and schedule in which upwards convergence of early arrival times takes arbitrarily long. For  $m < 4/\epsilon$ ,  $d_2^m = 2 + m\epsilon$ , and for  $m \geq 4/\epsilon$ ,  $d_2^m = 6$ .

#### 4. Traps for the Unwary

Even in the absence of multiple solutions, iterative methods for solving the  $a$  and  $d$  equations can take arbitrarily many iterations regardless of the choice of starting point and the direction of convergence. In Figure 4 we see a circuit and schedule which takes  $4/\epsilon$  iterations to reach a solution under upwards convergence. Note however that the  $A$  and  $D$  equations for this same circuit and schedule have no solution. A close rereading of the proof of Lemma 2.5 reveals that the  $a$  and  $d$  equations are only guaranteed to converge in  $n$  iterations when the  $A$  and  $D$  equations are known to be solvable.

Earlier approaches [1, 5] initialize the  $A$  and  $D$  equations using lower bounds, and the  $a$  and  $d$  equations using upper bounds. The convergence for the  $A$  and  $D$  equations is upwards and for the  $a$  and  $d$  equations downwards. It is easy to show that the latter process must converge downwards because the first iterate is componentwise less than or equal to the zeroth iterate and the solutions are bounded below. Another approach [6] attempts to first solve the  $A$  and  $D$  equations by upward convergence and then uses this solution to provide initial values for the  $a$  and  $d$  equations which then converge downwards to a solution. Unfortunately, both these methods can take arbitrarily long as illustrated in Figure 5. Moreover, neither process necessarily finds a minimum fixed point for the equations, so we might not get the “right” answer vis-a-vis the hold constraints (it is easy to see that if any solution has a hold violation, then the minimum solution will too).

When the SMO formulation is used for finding an optimal clock schedule, a mathematical optimization program is used to find a clock schedule and a set of arrival and departure times,  $(a, d, A, D)$ . Unfortunately, as we saw in Corollary 3.4, it is likely that multiple solutions exist for an optimal schedule, and so one must carefully interpret whether such a schedule is in fact correct. Moreover, if the min and max operators in the equations were relaxed to inequalities before being passed to the solver as advocated in [7], then the schedule returned by the optimizer is correct but the arrival and departure times returned by the



*Figure 5.* A circuit and schedule in which downwards convergence of early arrival times takes arbitrarily long. The least fixed point of the latest equations has  $A_2 = D_2 = 7$ . Taking this as a starting point for iterating the earliest time equations, we have  $a_2^m = 7 - m\epsilon$  for  $m < 1/\epsilon$ , and  $a_2^m = 6$  for all larger  $m$ .

solver might not even satisfy the original SMO equations. It has been suggested that iterating the equations using the times from the solver as starting values is will yield a solution. This is true, but probably unwise. First, it might take a long time to converge if the  $a$  and  $d$  values decrease. Second, you might converge to a solution other than the minimum solution. It probably makes more sense to simply discard the arrival and departure times found by the solver and use the verification algorithm of §2 to calculate a minimum solution directly from the optimal schedule. Depending on one's view on the multiple solution phenomenon, one might then test whether the solution satisfies the setup and hold constraints, or else test the solution for uniqueness using Theorem 3.13.

An amusing insight is provided by considering the operation of the algorithm of §2 as a simulation of the operation of the circuit during the first  $n$  clock cycles after power is turned on. As the early arrival and departure times increase monotonically to their steady-state values, they might very well violate hold requirements at various latches. This can happen even if the equations have a unique solution and no hold constraints are violated at that solution. This implies that any reset operations intended to initialize the circuit to a consistent state should persist for at least as many cycles as it takes the algorithm to converge.

## 5. Implementation Experience

We implemented the algorithm shown in Figure 6. It is easy to show that this algorithm gives the same answer as Construction 2.1, converges at least as fast, and only uses  $O(n)$  storage. Not shown are a number of features that make the algorithm more useful in practice. First, critical short and long paths can be recovered after execution by storing with each  $A_i$  or  $a_i$  the value of  $j$  which last caused it to be increased. Second, this same backtracking can be used to print out a critical long path if the algorithm diverges. Finally, the  $A_i$  and  $a_i$  should be “clipped” to  $\pi - S_i$  in order to prevent the propagation of false errors which

```

for each  $i$  with  $1 \leq i \leq n$  do
   $A_i \leftarrow a_i \leftarrow -\infty;$ 
for  $m \leftarrow 1$  step 1 until  $m > n$ 
  for each  $i$  with  $1 \leq i \leq n$  do
     $D_i \leftarrow \max(A_i, B_i);$ 
     $d_i \leftarrow \max(a_i, B_i);$ 
  for each  $i$  with  $1 \leq i \leq n$  do
     $A_i \leftarrow \max(A_i, \max_{j \rightarrow i}(D_j + \Lambda_{j,i}));$ 
     $a_i \leftarrow \max(a_i, \min_{j \rightarrow i}(d_j + \lambda_{j,i}));$ 
  if no  $A_i$  or  $a_i$  changed during this pass then
    return "algorithm converged";
  return "algorithm diverged";

```

*Figure 6.* Pseudo-code for the algorithm as implemented.

might otherwise occur when an overly long path continues on to other latches. Users usually only want to see a diagnostic at the first late latch in each such path. Various heuristics may be employed to make the algorithm run faster, for example, only evaluating those equations for which an argument has changed during the previous iteration.

The program itself is only 558 lines of C, most of which are concerned with reading the circuit description or formatting the output. We ran the program on transformed versions of all the ISCAS '89 benchmarks as described in [8] using clock schedules that had been found to be optimal by other means. The largest such circuit had 3272 latches and 67704 edges in the  $\rightarrow$  relation. In all cases the running time of the algorithm was less than 20 seconds, almost all of which was consumed reading the circuit description and building data structures. Moreover, only a few iterations were ever necessary for the algorithm to converge, implying (for these circuits anyway) that signals do not usually flow continuously through very long chains of transparent latches without having to stop and wait.

## 6. Summary

Although the solution to the SMO equations is not necessarily unique, multiple solutions can only occur at the optimal clock period. The presence of a critical cycle in the circuit is both a necessary and sufficient condition for these multiple solutions to exist. We offered some viewpoints on the physical significance of non-unique solutions, and pointed out some of the complications that they cause in both timing verification and optimization.

Simple iterative techniques can be used to solve the equations, but one must use care in picking the starting point. The wrong starting point can lead to arbitrarily long running times and incorrect results, but the correct starting point

is guaranteed to converge in  $n$  iterations, where  $n$  is the number of latches in the circuit. The total running time is at most  $O(n^3)$  time, or  $O(ne)$  time where  $e$  now is the number of edges in the circuit graph. Given a solution, we can test in  $O(n^2)$  time (alternatively,  $O(e)$  time) whether it is a unique solution.

The algorithm is simple to implement, and we ran it on enough test circuits to conclude that it takes more time in practice to read the circuit than to run the algorithm.

## References

- [1] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, “*checkT<sub>c</sub>* and *minT<sub>c</sub>*: timing verification and optimal clocking of synchronous digital circuits,” in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 552–555, Nov. 1990.
- [2] T. M. Burks, K. A. Sakallah, and T. N. Mudge, “Multi-phase retiming using *minT<sub>c</sub>*,” in *ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Mar. 1992.
- [3] W.-H. Lien and W. Burleson, “Wave-domino logic: Timing analysis and applications,” in *ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Mar. 1992.
- [4] A. T. Ishii and C. E. Leiserson, “A timing analysis of level-clocked circuitry,” in *Proceedings of the Sixth MIT Conference* (W. J. Dally, ed.), Advanced Research in VLSI, pp. 57–69, MIT Press, 1990.
- [5] R.-S. Tsay and I. Lin, “A system timing verifier for multi-phase level-sensitive clock designs,” Tech. Rep. RC 17272, IBM, Oct. 1991.
- [6] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, “A pseudo-polynomial algorithm for verification of clocking schemes,” in *ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Mar. 1992.
- [7] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, “Analysis and design of latch-controlled synchronous digital circuits,” *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 322–333, Mar. 1992.
- [8] T. G. Szymanski, “Computing optimal clock schedules,” in *Proceedings of the IEEE/ACM Design Automation Conference*, vol. 29, pp. 399–404, June 1992.

# EFFICIENT IMPLEMENTATION OF RETIMING

Narendra Shenoy and Richard Rudell

*Synopsys Inc.*

*700 E. Middlefield Road*

*Mountain View CA 94043*

## Abstract

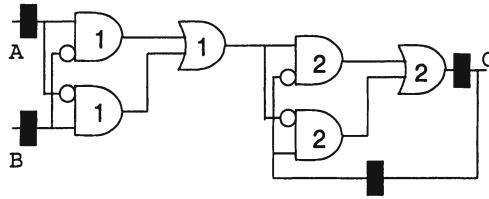
Retiming is a technique for optimizing sequential circuits. It repositions the registers in a circuit leaving the combinational cells untouched. The objective of retiming is to find a circuit with the minimum number of registers for a specified clock period. More than ten years have elapsed since Leiserson and Saxe first presented a theoretical formulation to solve this problem for single-clock edge-triggered sequential circuits. Their proposed algorithms have polynomial complexity; however naive implementations of these algorithms exhibit  $O(n^3)$  time complexity and  $O(n^2)$  space complexity when applied to digital circuits with  $n$  combinational cells. This renders retiming ineffective for circuits with more than 500 combinational cells. This paper addresses the implementation issues required to exploit the sparsity of circuit graphs to allow min-period retiming and constrained min-area retiming to be applied to circuits with as many as 10,000 combinational cells. We believe this is the first paper to address these issues and the first to report retiming results for large circuits.

## 1. Introduction

Retiming is a sequential logic optimization technique. Leiserson and Saxe provided the first formulation and theoretical solution to this problem in 1983 [4] although their later paper [5] has the most complete overview of this work. Retiming uses the flexibility provided by repositioning memory elements to optimize a circuit for one of several objective functions:

- 1 **min-period**:minimize the clock period of the circuit
- 2 **min-area**:minimize the number of registers in the circuit
- 3 **constrained min-area**:minimize the number of registers in the circuit subject to a maximum constraint on the clock period, or indicate failure if the target period cannot be achieved.

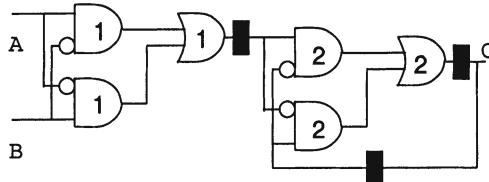
As a means of motivating and introducing the concept of retiming, we present a simple example in Figure 1. Assume that each gate has the delay shown inside it. The solid rectangles represent edge-triggered registers. A single clock is used to drive the clock pins of registers. The best clock period for such a circuit (neglecting clock skew and set-up time of registers) is given by the maximum delay of a path consisting of gates. The clock period for the circuit shown in



*Figure 1.* A simple circuit.

Figure 1 is 6 units. An equivalent circuit with 3 registers and a clock period of 4 units can be obtained by repositioning registers as shown in Figure 2. This circuit has the minimum number of registers. On the other hand, the minimum period achievable by moving registers is 2 units at a cost of 4 registers as shown in Figure 3. Thus a simple re-configuration of memory elements yields designs with differing area costs (number of registers) and performance (clock period). It is this trade-off that we are interested in investigating.

For digital circuit design, the only interesting objective function is constrained minimum area retiming. However, the minimum period retiming problem remains important as a step in solving the constrained min-area problem. This is because the min-period problem is computationally less intensive and it provides a lower-bound for the best delay achievable by the constrained min-area problem. For these reasons, we address both the min-period and constrained min-area optimization problems in this paper.



*Figure 2.* Retiming for minimum registers.

## 2. Previous Work

For the case of circuits with edge-triggered memory elements (registers) clocked by the same signal, solutions to all three problems are described by Leiserson and Saxe [5]. Without taking anything away from their significant contribution, we mention simply that the implementation details necessary for large sparse circuits were not reported as part of their work. Ishii, Leiserson,

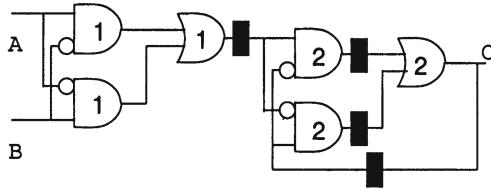


Figure 3. Retiming for minimum period.

and Papaefthymiou [2], extend the concepts to handle level-sensitive memory elements. Lockyear and Ebeling [6] present an alternative approach to retiming circuits with level-sensitive memory elements. However, both of these papers address the theoretical issues involved and not implementation or efficiency details. Papaefthymiou and Randall report experimental results for level-sensitive memory elements [8], but the largest circuit they handle has 379 gates. Munzner and Hemme [7] propose a heuristic algorithm for constrained min-area retiming to convert a combinational circuit into a pipeline. Even though retiming can be used for the same effect, they justify the use of a heuristic algorithm by stating that the retiming algorithms cannot handle circuits with more than 400 gates.

Although theoretical solutions to edge-triggered flip-flop retiming and related problems have been presented in the literature, very few papers have reported experimental results using retiming. To the best of our knowledge, experimental results for constrained min-area retiming problem have not been reported. The only reported results for min-period retiming we have found are for small circuits. We believe the primary reason for this is that, although the algorithms are polynomial in the circuit size, naive implementations suffer the worst-case  $O(n^3)$  time and  $O(n^2)$  space for all circuits.

### 3. Definitions

A sequential circuit is an interconnection of logic gates and memory elements. A sequential circuit can be represented by a directed graph  $G(V, E)$ , where each vertex  $v$  corresponds to a gate  $v$ . Each directed edge  $e_{uv}$  represents a flow of signal from the output of gate  $u$  at its source to the input of gate  $v$  at its sink. Each edge has a weight  $w(e_{uv})$  which indicates the number of registers that the signal at the output of gate  $u$  must propagate through before it is available at the input of gate  $v$ . Each vertex  $v$  has a constant delay  $d(v)$ . If there is an edge from vertex  $u$  to vertex  $v$ ,  $u$  is called a fanin of  $v$  and  $v$  is called a fanout of  $u$ . The set of fanouts (fanins) of  $u$  is denoted by  $FO(u)$  ( $FI(u)$ ). A special vertex called the host vertex is introduced in the graph with edges directed from the

host vertex to all vertices that represent primary inputs and edges directed from all vertices representing primary outputs to the host vertex.

A retiming is a labeling of the vertices  $r: V \rightarrow Z$  where  $Z$  is the set of integers. The weight of an edge  $e_{uv}$ , after retiming is denoted by  $w_r(e_{uv})$  and is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \quad (1)$$

The retiming label  $r(v)$ , for a vertex  $v$ , represents the number of registers moved from its output towards its inputs. A path  $p$  is defined as a sequence of alternating vertices and edges, such that each successive vertex is a fanout of the previous vertex and the intermediate edge is directed from the former to the later. A path can start and end at vertices only. The existence of a path from vertex  $u$  to vertex  $v$  is represented as  $u \rightarrow \dots \rightarrow v$ . The weight of a path  $w(p)$  is the sum of the edge weights for the path. The delay of a path  $d(p)$  is the sum of the delays of the vertices on the path. A 0-weight path  $p$ , is a path with  $w(p) = 0$ . The clock period  $c$  is determined by the following equation:

$$c = \max_{p|w(p)=0} \{d(p)\} \quad (2)$$

We briefly summarize the results obtained by Leiserson and Saxe [5]. An important concept to the retiming algorithms is the definition of the  $W$  matrix and the  $D$  matrix. They are defined as:

$$W(u, v) = \min_{p: u \rightarrow \dots \rightarrow v} \{w(p)\}, \quad (3)$$

$$D(u, v) = \max_{\substack{p: u \rightarrow \dots \rightarrow v \\ \text{and } w(p) = W(u, v)}} \{d(p)\}. \quad (4)$$

The matrices are defined for all pairs of vertices  $(u, v)$  such that  $v$  is reachable from  $u$  by a sequence of edges and the path does not include the host vertex.  $W(u, v)$  determines the minimum latency, in clock cycles, for data flowing from  $u$  to  $v$  and  $D(u, v)$  gives the maximum delay from  $u$  to  $v$  for the minimum latency.

### 3.1 Minimum period retiming

The objective is to obtain a circuit with the minimum clock period without any consideration to the area penalty due to an increase in the number of registers. The retiming constraints for a target clock cycle  $c$  translate into two sets of inequalities:

- 1 Non-negativity of edge-weights after retiming requires  $w_r(e_{uv}) \geq 0$ ,  $e_{uv} \in E$ , i.e.

$$r(v) - r(u) \geq -w(e_{uv}), \quad \forall e_{uv} \in E. \quad (5)$$

- 2 Correct clocking at a clock period  $c$  requires that all paths  $u \rightarrow \dots \rightarrow v$  with  $D(u, v) > c$ , after retiming have at least one register on it, i.e.

$$r(v) - r(u) \geq -W(u, v) + 1, \quad \forall u \rightarrow \dots \rightarrow v, D(u, v) > c. \quad (6)$$

The sets of constraints from Equations 5 and 6 can be solved using the Bellman-Ford relaxation technique developed for the “shortest path on a graph” problem [3]. Leiserson and Saxe introduce three algorithms to solve the min-period retiming problem; we describe the most efficient one known as the relaxation method. Let  $\Delta(v)$  denote the largest delay seen along any combinational path that terminates at the output of  $v$ .

$$\Delta(v) = d(v) + \max_{\substack{u \in FI(v) \text{ and} \\ w(e_{uv})=0}} \{\Delta(u)\}. \quad (7)$$

It can be shown that the clock period  $c$  is given by the expression

$$c = \max_{v \in V} \{\Delta(v)\}. \quad (8)$$

The relaxation algorithm has the following  $O(|V||E|)$  subroutine which determines if a retiming exists for a given clock period  $c$ . We refer the reader to [9] for a proof of correctness of this algorithm.

```

min_period_relaxation algorithm {
  For each  $v \in V$  set  $r(v) = 0$ 
  For  $|V| - 1$  times {
    Compute retimed edge weights (Equation 1)
    Compute  $\Delta(v)$ , for all  $v \in V$  (Equation 7)
    For all  $v \in V$ , such that  $\Delta(v) > c$ , do  $r(v) +=$ 
  }
  Compute retimed edge weights (Equation 1)
  If  $\max_{v \in V} \{\Delta(v)\} > c$ , then no feasible retiming,
  else the current  $r$  yields a legal retiming.
}
```

### 3.2 Constrained Min-Area Retiming

Under the assumption that all registers have the same area, the min-area retiming problem reduces to seeking a solution with the minimum number of registers. Constraints for retiming to be valid are the same as in Equations 5 and 6. The formulation for constrained min-area retiming is:

$$\begin{aligned}
 & \min \sum_{v \in V} (|FI(v)| - |FO(v)|)r(v) \\
 & r(v) - r(u) \geq -w(e_{uv}) \quad \forall e_{uv} \in E \\
 & r(v) - r(u) \geq -W(u, v) + 1 \quad \forall D(u, v) > c
 \end{aligned}$$

These are linear constraints and the objective function is also a linear function of the retiming variables, so linear programming techniques can be used to solve this problem. Leiserson *et al.* [5] indicate that the dual of this problem is an instance of minimum cost circulation on a graph for which efficient algorithms exist. They also indicate that an initial feasible solution can be obtained directly from the problem.

We will not review the construction of the minimum cost circulation problem (details may be found in [9]), except to note that the retiming graph is augmented with dummy vertices, dummy edges, and capacity edges to transform it to the graph on which the minimum cost circulation is solved. The edges in this graph that originate due to the edge weights are termed “circuit” edges as there is one edge for every edge in the original circuit. The edges in this graph that come from the clock period constraints are called “period” edges. Note that the number of period edges can be very large and destroy the sparsity of the graph; we will deal with this issue in the next section.

## 4. Implementation issues

Our goal is to handle circuits with up to 10,000 combinational cells. However, we expect circuit graphs to be sparse, *i.e.*  $|E| = k|V|$  for small  $k$ . For min-period retiming, the bottleneck is the requirement that we iterate  $O(|V|)$  times to prove that there is no feasible retiming. For constrained min-area retiming, the bottleneck is that even when the graphs are sparse, the  $W$  and  $D$  matrices are dense. Further, the number of clock period edges which are implied by the retiming equations indicate that the retiming graph augmented with clock period edges will be dense. We demonstrate in this section how to solve each of these problems.

### 4.1 Minimum period retiming

Let us focus on the relaxation algorithm to determine if  $c$  is a feasible clock period. It is empirically observed that if  $c$  is feasible then the retiming labels converge rapidly before completing  $|V| - 1$  iterations. On the other hand, one cannot determine that a clock period  $c$  is infeasible until all  $|V| - 1$  iterations have been exhausted and the retiming labels have failed to converge. Thus any hope of speeding up min-period retiming must focus on detecting if a clock period is infeasible before completing the requisite  $|V| - 1$  iterations if possible. This is the principal motivation for this section.

The Bellman-Ford algorithm solves the following problem. Given a directed graph  $G(V, E)$  with arbitrary edge-weights  $f : E \rightarrow R$  ( $R$  is the set of reals), and vertices with an initial distance marked on them, find the shortest distance (measured by sum of edge weights) to every vertex that respects the initial distance

marking. The algorithm can be described as follows (where  $r(v)$  now denotes the distance marking to a vertex  $v$ ):

```

Bellman-Ford algorithm {
    For each  $v \in V$ ,  $r(v) = \begin{cases} \text{known original distance,} \\ +\infty \text{ otherwise.} \end{cases}$ 
    Loop  $|V| - 1$  times {
        For each edge  $e_{uv}$  {
             $r(v) = \min_{u \in FI(v)}(r(v), r(u) + f(e_{uv}))$ 
        }
    }
    For each edge  $e_{uv}$  {
        If  $r(v) > r(u) + f(e_{uv})$  then FAIL (negative cycle)
    }
}

```

The graph is permitted to have negative edge weights and hence can have negative cycles. The presence of a negative cycle makes the shortest distance to any vertex on that cycle undefined. In the presence of a negative cycle, Bellman-Ford must report failure to converge.

We can abort the iteration at any point if we discover a negative cycle in the graph. Let us call such a negative cycle a certificate of infeasibility. We present a technique inspired by Szymanski who used a similar approach to compute lower bounds for clock periods [10]. The predecessor heuristic maintains a predecessor vertex pointer denoted by  $\text{pred}()$  with each vertex. The Bellman-Ford algorithm starts with all  $\text{pred}()$  pointers set to be empty. Every time vertex  $v$  has its distance decreased, the fanin node that caused the change is stored as  $\text{pred}(v)$ ; i.e., in the Bellman-Ford algorithm, if during the relaxation of edge  $e_{uv}$ , we discover that  $r(v) > r(u) + f(e_{uv})$ , then we set  $\text{pred}(v) = u$ . Thus at every instant of the iteration, we have a sub-graph of the original graph maintained by the predecessor edges with  $|V|$  edges and  $|V|$  vertices.

Each vertex  $v$  has a predecessor graph associated with it, defined by repeated traversing of the  $\text{pred}()$  pointers, starting at  $v$  and ending when either the predecessor is empty or a cycle is found. At every iteration the predecessor graphs of all vertices are examined to see if a negative cycle exists. If  $v$  is marked, its predecessor graph has been examined during an earlier traversal. A cycle is detected by checking if a vertex has already been visited during the walk started from the current  $v$ . The traversal is stopped whenever a cycle is detected, a marked vertex is visited, or the end of the predecessor chain has been reached. The complexity of traversal algorithm is  $O(|V|)$  and is dominated by the  $O(|E|)$  relaxation of the edges. The traversal mechanism is outlined below:

```

cycle detection using predecessor heuristic {
    For each  $v \in V$ ,  $\text{mark}(v) = 0$ 
    For each  $v \in V$ ,  $\text{cycle}(v) = 0$ 
}

```

```

cycle_count = 0
For each  $v \in V$  {
    if (!mark( $v$ )) {
        cycle_count++
         $u = v$ 
        while( $u \neq \text{NIL}$ ) {
            if (mark( $u$ ) && cycle( $u$ ) == cycle_count) declare
                cycle exists and exit
            if (mark( $u$ )) no cycle in this sub-tree, break out of inner loop
            mark( $u$ ) = 1
            cycle( $u$ ) = cycle_count
             $u = \text{pred}(u)$ 
        }
    }
}

```

Let us now extend this analogy to the min-period retiming problem. When  $\Delta(v)$  is computed for each vertex  $v$ , we store with  $v$ , a vertex  $u$ , such that there exists a 0-weight path  $u \rightarrow \dots \rightarrow v$  and  $\Delta(v) = d(p)$ . If  $\Delta(v) > c$ , then we set  $\text{pred}(v) = u$ . Consider a cycle in the predecessor sub-graph that includes vertices  $u_1, \dots, u_k = u_0$ , i.e.  $\text{pred}(u_i) = u_{i-1}$ ,  $i = 1, \dots, k$ . Let  $p_{i-1,i}$  denote the path  $u_{i-1} \rightarrow \dots \rightarrow u_i$  which is used in the computation of  $\Delta(u_i)$ . During the iterations, retiming labels increase only by 1. Recall that before the labels are updated,  $w(p_{i-1,i}) = 0$ . After update,

$$\begin{aligned}
w_r(p_{i-1,i}) &= r(u_i) - r(u_{i-1}) + w(p_{i-1,i}) \\
&= r(u_i) - r(u_{i-1}) \\
&\leq 1.
\end{aligned}$$

In addition  $d(p_{i-1,i}) > c$ . Thus for the cycle

$$\begin{aligned}
\sum_{i=1,\dots,k} d(p_{i-1,i}) &> kc \\
\sum_{i=1,\dots,k} d(p_{i-1,i}) &> c \sum_{i=1,\dots,k} w_r(p_{i-1,i}).
\end{aligned}$$

The term on the left hand side represents the total delay encountered as a signal traverses the cycle. The term on the right hand side is the total time available for the signal to propagate under the current clock period. Retiming cannot change the number of registers in a cycle of a circuit (see Lemma 1 in [5]). This implies that for a clock period  $c$ , this cycle will prevent any feasible

retiming. Retiming can only exist for a clock period given by the inequality,

$$c' \geq \frac{\sum_{i=1,\dots,k} d(p_{i-1,i})}{\sum_{i=1,\dots,k} w(p_{i-1,i})}. \quad (9)$$

The implementation of this technique results in dramatic speed-up in execution time. Not only can infeasibility be detected early, but the cycle that caused infeasibility provides a new lower bound for the clock period. This can be used to bias the binary search effectively. The memory overhead consists of a pointer per vertex and an extra field used for detecting the cycle.

**4.1.1 Experimental results.** All experiments are run on a Solbourne Series 5e. We select some circuits from the ISCAS89 suite chosen so that they reflect the variation in size of this suite.

We compare an implementation of the retiming algorithm without predecessor pointers to an implementation that uses it in Table 1. The two implementations are identical except for the part that traverses the predecessor pointers. We see a substantial reduction in execution time for the predecessor heuristic. Using the cycle obtained as a certificate of infeasibility to update the lower bound on the clock period is useful, as the bias eliminates feasibility checks at clock periods less than the bound and are guaranteed to be infeasible.

name	# gates	CPU (in sec.)		clock period	
		original	new	before	after
s1494	386	96.2	2.1	17.4	17.4
s1423	384	101.0	3.9	36.6	31.4
s5378	887	527.6	13.7	11.9	10.4
s9234	1107	159.8	13.8	19.9	12.9
s13207	1854	1973.2	28.8	23.2	21.4
s15850	2240	2856.1	37.4	40.1	22.9
s38584	7882	39025.9	306.2	35.5	34.1

Table 1. Results for minimum period retiming.

## 4.2 Minimum area retiming

Minimum area retiming poses 2 major hurdles;

- 1 computing the  $W$  and  $D$  matrices, and
- 2 implementing minimum cost circulation.

**4.2.1 Computing  $W$  and  $D$  matrices.** We shall not describe the method proposed by Leiserson *et al.* [5] to compute the  $W$  and  $D$  matrices, except to say that an algorithm with  $O(|V|^3)$  time and  $O(|V|^2)$  memory is proposed. The  $W$  and  $D$  matrices are required to add the clock period edges which

in turn are required to solve the minimum cost circulation problem using cost scaling techniques. Further, all of the clock period edges must be added prior to solving the circulation problem.

The number of clock period edges greatly increases the density of the original graph. However, only a small subset of the period edges implied by Equation 6 are required for the computation; the rest form redundant constraints. The original algorithm has two drawbacks: the  $O(|V|^2)$  memory and the inability to prune the clock period edges. We propose an algorithm which has a worse complexity than the original formulation, but whose memory is  $O(|V|)$  and is able to generate a smaller set of constraints for sparse circuit graphs.

Equation 6 describes the conditions under which clock period edges need to be added to the retiming graph. In the original formulation of constrained min-area retiming, clock period edges are required between all vertices  $u$  and  $v$  such that

$$u \rightarrow \dots \rightarrow v \text{ and } D(u, v) > c. \quad (10)$$

To see why a smaller set of clock period edges is sufficient for constrained min-area retiming, note that if

$$r(v) - r(u) \geq -W(u, v) + 1 \quad (11)$$

is true for a sub-path of a path, then it is also true for the entire path. Hence a period edge need only be added to vertex  $v$ , reachable from  $w$ , such that:

$$D(w, v) > c \text{ and } D(w, u) \leq c \ \forall u \text{ lying on } w \rightarrow \dots \rightarrow FI(v) \quad (12)$$

Consider a vertex  $w$ . We are interested in the period edges that have their source as  $w$ . To do this, it is necessary to examine only a single row of the  $W$  and  $D$  matrices (*i.e.*, the row  $W(w, .)$  and the row  $D(w, .)$ ). The set of vertices can be partitioned into disjoint sets depending on the value of  $W(w, .)$  as shown in Figure 4. The directed edges in Figure 4 represent paths to other vertices from  $w$ . The dashed curve represents the set of vertices  $v$  that meet the condition of Equation 12. We can ignore any period edges between  $w$  and the fanouts of such vertices. Thus only some of the entries of  $W(w, .)$  and  $D(w, .)$  need to be computed. The elements of  $W(w, .)$  can be computed using Dijkstra's algorithm since the edge weights  $w(e_{uv}) \geq 0, \forall e_{uv} \in E$ . The computation of  $D(w, .)$  is complicated due to 2 facts; the dependence on  $W(w, .)$  and the non-monotonicity of the gate delays (akin to finding the longest path in a graph with positive edge weights).

We now describe how to generate a single row of the  $W$  and  $D$  matrices for a single vertex  $w$ , and how to find only the set of vertices which satisfy Equation 12. An ordered pair  $(w(e_{uv}), -d(v))$  is associated with each edge  $e_{uv}$  and is used to compute the shortest distance from  $w$ . Comparisons are done in lexicographical order. Thus for a path  $w \rightarrow \dots \rightarrow v$ , we obtain the distance as an

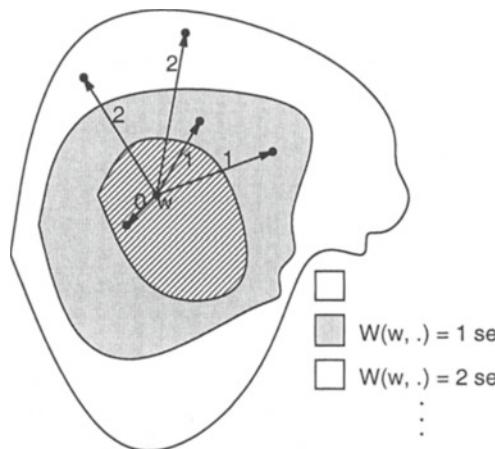


Figure 4. Disjoint partitions of the vertices.

ordered pair denoted by  $(a_v, b_v)$ . Upon termination of the algorithm this pair is the same as  $(W(w, v), D(w, v))$ . The algorithm to compute period edges consists of applying the following mix of Dijkstra's algorithm and Bellman-Ford algorithm. The algorithm maintains a heap for each distinct value of  $a_v$  (the heap is indexed by this value since there could be several such heaps). Since  $w(e_{uv}) \geq 0$ , we are guaranteed that the first component of the distance measure cannot decrease for all vertices in the heap with the lowest index. To compute  $b_v$  for all  $v$  in the smallest indexed heap, the Bellman-Ford algorithm is used.

**adding period constraint edges {**

$c$  = target clock period

$S_k$  = the  $k^{\text{th}}$  heap

For each  $w \in V$  {

$w$  = current vertex

For each  $v \in V$ ,  $a_v = 0$  and  $b_v = +\infty$

$S_0 = \{w\}$ ,  $a_w = d(w)$  and  $b_w = 0$

$k$  = current register weight

do {

$k = \min\{p | S_p \neq \emptyset\}$

$u = \text{pop\_min}(S_k)$

if ( $b_u > c$ ) {

    add period edge from  $w$  to  $u$  with weight  $a_u - 1$

} else {

    For each  $v \in FO(u)$  {

        if  $((a_v, b_v) > (a_u + w(e_{uv}), b_u + (-d(v))))$  {

Dijkstra's algorithm for shortest paths on a graph requires that the distance measure be monotonic. The distance measure (as defined by the lexicographical order) of a vertex  $v$  is a function of the edge-weights and the vertex delays. Edge weights are monotonic, since  $w(e) \geq 0, \forall e \in E$  and we are interested in distances with minimum number of edge-weights. However, the vertex delays are non-monotonic; *i.e.* after popping  $u$  using `pop_min()`, we cannot conclude that  $b_u$  has attained the value of  $D(w, u)$ . To handle this, a Bellman-Ford relaxation is carried out for each value of  $k$  (the minimum index amongst the set of heaps).

Consider the analysis for a given  $w$ . Let the index  $k$  increase from 0 to  $K$ . Let  $V_k$  be the set of vertices for which  $W(w, v) = k$ . Let  $E_k$  be the set of edges with 0 edge weights with either sources or sinks in  $V_k$ . Due to the non-monotonicity of vertex delays, we are forced to execute a Bellman-Ford set of relaxations for each  $k$ . Hence for a given  $k$ , there will be at most  $|V_k||E_k|$  heap queries, each of which requires  $\log |V_k|$  time; yielding a complexity of  $|V_k||E_k|\log |V_k|$ . Since we are guaranteed that every cycle has at least one register, distances cannot be increased arbitrarily by traversing cycles. Thus the algorithm requires  $O(\sum_{k: V_k \neq \emptyset} |V_k||E_k|\log |V_k|)$ . Note that the vertex sets  $V_k$  (and edge sets  $E_k$ ) are disjoint and hence we can bound this by  $O(|V||\bar{E}|\log |\bar{V}|)$ , where  $\bar{V}$  and  $\bar{E}$  are the maximum sized sets amongst  $V_k$  and  $E_k$  (among possible values of  $k$ ). Since this is repeated for each vertex the worst case bound is  $O(|V|^2|E|\log |V|)$  — considerably worse than a Floyd-Warshall ( $O(|V|^3)$ ). There are two benefits to using this algorithm. First the memory overhead is a set of heaps whose total size can be kept to  $O(|V|)$  with some book-keeping. Secondly, the execution time rarely displays the behavior predicted by worst case analysis. On sparse graphs, the term  $\sum_{k: V_k \neq \emptyset} |V_k|$ , rarely reaches its upper bound of  $|V|$  because, not all vertices are reachable from  $w$ , and the pruning of distance propagation once the delay of a path becomes larger than  $c$  effectively restricts the set of vertices examined. This pruning strategy speeds up the convergence of the algorithm.

In practice, the term  $|\bar{E}| \log |\bar{V}|$  is much smaller than  $|E| \log |V|$ . The final proof of this algorithm is in the implementation. For some of the large circuits used in the experiments, it is almost impossible to finish computing the  $W$  and  $D$  matrices in a reasonable amount of time using a Floyd-Warshall implementation.

With the above algorithm, the execution time is comparable to minimum cost circulation computation.

**4.2.2 Minimum cost circulation using cost-scaling.** A disadvantage of cost-scaling techniques is that the graph cannot change during the computations; consequently all the period edges must be determined before starting the cost-scaling algorithm. Our implementation is based on the generalized cost-scaling framework of Goldberg and Tarjan [1] and has a complexity of  $O(|V||E|\log|V|\log(|V|C))$  where  $C$  is the largest cost in the graph *i.e.* one more than the number of registers in the circuit. If  $|V|$  is 10,000, and we restrict ourselves to 32-bit integer arithmetic,  $C$  must be less than 200,000; since  $C$  is the number of registers in the circuit, this is a reasonable assumption. The algorithm operates by maintaining an error from optimality and successively halving it. At the start this error is  $|V|C$ . However if an initial flow is known, the value of  $C$  can be reduced to be the minimum value that has to be added to the cost of each edge so that the graph does not have a negative cost cycle. This fact reduces the value of  $C$  by an order of magnitude. An initial flow for the circulation graph can be constructed easily. For most circuits  $C$  turns out to be less than 10, effectively making the algorithm independent of  $C$ .

As an aside, one has to be careful with the edge capacities. In general these are real numbers and can cause convergence problems. For sake of stability, the edge capacities are scaled to integers using a factor which is a function of the least common multiple of the number of fanouts of vertices in the graph.

**4.2.3 Experimental results.** The experimental setting for min-area retiming consists of the following steps. A min-period retiming is carried out to determine a bound on the best achievable clock period. The number of registers in the min-period solution is examined. A min-area retiming is performed on this circuit with the clock period set to the best achievable clock period. This yields an area optimal solution without any sacrifice in performance.

The results for constrained min-area retiming are compared to the results for min-period retiming in Table 2. The first set of circuits consists of the ISCAS circuits used for min-period retiming. The last 3 circuits (the second set) are multipliers pipelined by placing three registers in series at the outputs in each case (see Section 5 for details). As can be seen, the min-period solution can be very far away from the area optimal solution at the same clock period.

## 5. Tracing area-delay curves

One motivation for constrained min-area retiming algorithms is to examine the area-delay trade-off. This section presents a set of experiments on two multipliers: an 8x8 multiplier with 16 outputs and a 16x16 multiplier with 32 outputs. We choose multipliers because they are common circuits found in many data-

name	# gates	# registers		CPU for min-area
		min period	min-area at min-period	
s1494	386	7	7	58
s1423	384	81	78	84
s5378	887	296	169	300
s9234	1107	328	241	641
s13207	1854	548	481	2467
s15850	2240	655	529	4668
s38584	7882	1433	1429	139328
8x8	542	179	102	38
16x16	3030	710	183	2459
32x32	≈13k	2763	403	67155

Table 2. Results for minimum area retiming.

paths and signal processing designs. Each multiplier has 3 registers in series at each output. The area and delay characteristics for the initial circuits are summarized in Table 3 (columns 2-4). The clock period is the maximum delay from an input to an output, since the registers are all placed at the outputs. As the latency of each output is 3, we expect that min-period retiming will partition the circuit into 4 regions that have almost the same delay. Thus the value of the clock period at a min-period solution is expected to be roughly a fourth of the original clock period (column 5).

name	# gates	regi- sters	clock period	
			original	min-period
8x8	542	48	27.85	7.53
16x16	3030	96	58.82	15.92

Table 3. Properties of pipelined multipliers.

The next experiment concerns the area delay trade-off that is possible by changing the target clock period. The lower bound on the clock period is computed using the min-period retiming algorithm. The original clock period is an upper bound. 4 equi-distant points are picked between the upper and lower bound, as a target clock period for each circuit. The minimum number of registers required for each clock period is computed (in Table 4). An effective area-delay trade-off is seen. The CPU variation with the clock period is also shown in Table 4. This variation can be ascribed to the changing number of edges in the graph, which varies significantly as the clock period is varied. This variation is a response to the different target clock periods. For a given circuit, as the target clock is varied, the number of period edges varies considerably (see Table 5). At clock periods close to the lower bound, the period edges dominate all the

other edges in  $E$ . The number of period edges steadily increases as the target clock period is lowered and then decreases. This can be explained as follows. As the clock period decreases, the number of paths that need to be constrained increases. This is the general trend. Recall the pruning strategy used in the computation of period edges that ignores any path that extends from a sub-path with delay greater than the clock period. This has little effect at large clock periods, since most reachable vertices have delays less than the clock period. But when the clock period decreases, this strategy results in a decrease of period edges. Note that in all the cases, the graph remains sparse even after the period edges are added.

name	# registers at clock = $T_{min} + kt_{step}$					
	k=0	k=1	k=2	k=3	k=4	k=5
8x8	102	69	63	54	52	48
16x16	183	139	126	117	104	96
	CPU time at clock = $T_{min} + kt_{step}$					
	33.8	31.0	31.4	37.9	36	36.1
8x8	2459	4264	2325	2860	3397	2840

Table 4. Variation of register count with clock period.

name	V	E / V  at clock = $T_{min} + kt_{step}$					
		k=0	k=1	k=2	k=3	k=4	k=5
8x8	799	8.6	9.7	6.7	4.1	3.3	3.1
16x16	3944	25.5	38.5	26.5	7.3	3.7	3.2

Table 5. Sparsity of retiming graphs of pipelined multipliers.

## 6. Conclusions

The practicality of constrained min-area retiming (min-area retiming subject to a target clock period) for large circuits has been demonstrated. The issues that need to be addressed for a successful implementation require careful analysis and a good understanding of software techniques and computer algorithms. We hope to have highlighted some of these issues in this technical discussion.

The predecessor technique to detect infeasibility of a clock period has been demonstrated to be very effective for min-period retiming.

Two key contributions are made for the constrained min-area retiming problem. First, we have shown that finding essential period edges can be done efficiently. Secondly, using the information provided by an initial flow is critical for convergence of the cost-scaling algorithm.

The area-delay trade-off demonstrates a big win for the constrained min-area algorithm over the min-period algorithm, although this is hardly surprising. Although both algorithms have been known for several years, the experimental evidence has been lacking, especially for large circuits. The min-area algorithm enables a designer to control the area-delay trade-off of retiming in a precise manner.

Finally, we would like to dispel the notion that the existence of a polynomial-time algorithm implies that the techniques can be applied to large circuits; we must have near-linear complexity to handle entire circuits in one piece.

## Acknowledgments

We thank Albert Wang and Don MacMillen for stimulating several discussions. This work was sponsored by the ARPA Application Specific Electronic Modules (ASEM) Program under the Manufacturing Technology Directorate of Wright Laboratory, Air Force Material Command (ASC).

## References

- [1] A. Goldberg, E. Tardos, and R. E. Tarjan. Network flow Algorithms. Technical report, Department of Computer Science, 1989.
- [2] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing Two-Phase Level-Clocked Circuitry. In *Advanced Research in VLSI*, 1992.
- [3] E. L. Lawler. *Combinatorial Optimization: networks and Matroids*. Holt, Rinehart and Winston, 1976.
- [4] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. In *Journal of VLSI and Computer Systems*, pages 41–67, 1983.
- [5] C. E. Leiserson and J. B. Saxe. Retiming Synchronous Circuitry. In *Algorithmica*, 1991. 6(1).
- [6] B. Lockyear and C. Ebeling. Optimal Retiming of Multi-Phase Level-Clocked Circuits. In *Advanced Research in VLSI*, 1992.
- [7] A. Munzner and G. Hemme. Converting Combinational Circuits into Pipelined Data Path. In *Proceedings of the International Conference on Computer-Aided Design*, 1991.
- [8] M. C. Papaefthymiou and K. H. Randall. TIM: A Timing Package for Two-phase Level-clocked Circuitry. In *Proceedings of the Design Automation Conference*. IEEE/ACM, 1993.
- [9] J. B. Saxe. *Decomposable Searching Problems and Circuit Optimizations by retiming: Two Studies in General Transformations of Computational Structures*. PhD thesis, Carnegie-Mellon University, 1985.
- [10] T. G. Szymanski. Computing Optimal Clock Schedules. In *Proceedings of the Design Automation Conference*, 1992.

## **Part VIII**

# **INDUSTRY VIEWPOINTS**

A Cadence Perspective on ICCAD <i>Louis K. Scheffer</i>	633
ICCAD and Fujitsu CAD activities <i>Hiromu Hayashi</i>	639
ICCAD's Impact in IBM <i>John A. Darringer, Leon Stok and Louise H. Trevillyan</i>	645
Magma and ICCAD <i>Michel Berkelaar</i>	653
Designers Face Critical Challenges and Discontinuities of Analog/Mixed Signal Design and Physical Verification <i>Walden C. Rhines</i>	659
NEC and ICCAD - EDA partners in success <i>P. Ashar, S. Chakradhar, A. Gupta, J. Henkel, A. Raghunathan and K. Wakabayashi</i>	663
The Strong Mutual Impact between Philips Research and the ICCAD <i>Emile Aarts and Frans Theeuwen</i>	675
Contributions from the “Best of ICCAD” to Synopsys <i>Raul Camposano, Ahsan Bootehsaz, Debasish Chowdhury, Brent Gregory, Jim Kukula, Narendra Shenoy and Tom Williams</i>	683
ICCAD and Xilinx <i>Rajeev Jayaraman</i>	689

# A CADENCE PERSPECTIVE ON ICCAD

Louis K. Scheffer

*Cadence Design Systems, Inc.*

*San Jose, CA, USA*

## Abstract

ICCAD is the premier technical conference for CAD tools for ICs, and Cadence is the largest commercial vendor of such tools. Therefore it is not surprising that the two have many connections. Research from Cadence has been presented almost every year at ICCAD, and Cadence contributes heavily to other facets of the conference such as tutorials and panels. There is by no means a uni-directional flow of information, though - considerable research first reported at ICCAD has made its way into commercial products sold by Cadence. This paper covers some of the contributions to and from Cadence in the areas of timing, circuit simulation, layout analysis, physical design, logic verification, synthesis, and place and route.

## 1. Introduction

In a fast moving field such as CAD for IC design, conferences play a crucial role in the dissemination of results. They are available to a much wider audience than internal technical reports, and appear much sooner than the refereed journals. Conferences inspire follow-on research through reports of work in progress and opportunities for informal discussion. ICCAD, as the premier technical conference in the field, has seen the introduction of a number of great ideas. This volume is a tribute to these ideas.

As the largest vendor of CAD tools, Cadence has long had strong connections to ICCAD. This includes presenting internal research, helping with the conference itself, and using ideas brought forth by others. This paper describes both how Cadence has contributed to ICCAD and how ideas from the conference then translated into commercial practice at Cadence. Since the contributions to ICCAD are largely public (and voluminous), they are only summarized. The influence of ICCAD on Cadence products is less readily available, and constitutes the bulk of this paper.

## 2. Cadence Contributions to ICCAD

Almost every year, papers from Cadence appear at ICCAD, and Cadence employees serve on the technical and organizing committees, appear on panels, serve as session chairs and otherwise support the conference. For example, at the 2001 ICCAD alone, Cadence employees

- ◊ Authored or co-authored eight papers
- ◊ Chaired two conference committees

- ◊ Served on the technical program committee (3 people)
- ◊ Presented or co-presented two tutorials
- ◊ Organized a panel
- ◊ Moderated six sessions

There were also many other papers by professors that work closely with Cadence.

Clearly space does not allow listing all the Cadence research that has been reported at ICCAD, but three best papers selected for this book deserve special mention. The paper “Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation” by O’Brien and Savarino, (see page 393), “A Method for Correct by Construction Latency Insensitive Design” by Carloni, McMillan, Saldanha, and Sangiovanni-Vincentelli (see page 143) and “GRASP - A New Search Algorithm for Satisfiability” by Silva and Sakallah, (see page 73) all were ground breaking research done at Cadence. In addition, many authors of academic ICCAD papers have continued their work at Cadence, as demonstrated by “Nonlinear Circuit Simulation in the Frequency Domain” by Kundert (see page 383). Finally, many of the academic authors of ICCAD papers have strong ties to Cadence. Alberto Sangiovanni-Vincentelli, for example, is an author of several papers in this volume and also serves as Chief Technical Advisor to Cadence.

### 3.     ICCAD Influence on Cadence

Many techniques now embedded in commercial products were introduced, or refined, in ICCAD papers. Again, a fully detailed accounting would be quite long, so this section concentrates on the ICCAD papers selected for this book, and examines their influence on Cadence products.

#### 3.1     Timing

In the 1980s, wire delays were short compared to cell delays, and the entire capacitance of the net could be used as the cell load. As processes shrank, this was no longer true, making delay calculations inaccurate. The paper “Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation” by O’Brien and Savarino (see page 393) represented each net as a pi network loading the driver and separate delays to each input. This was easy to implement and gave much more accurate results than lumped models, and so became the standard for representing interconnect delays for many years. The results are still visible as the circuit model in RSPF (Reduced Standard Parasitic Format). However, as processes shrank further, this model too became inac-

curate, and was in turn replaced by more sophisticated moment based methods such as PRIMA, also introduced at ICCAD.

The paper “PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm” by Odabasioglu, Celik, and Pileggi (see page 433) explained how to generate reduced models that are guaranteed to be passive as well as stable. This is a huge benefit since it guarantees stability when models are composed or combined with external sources. All the modern delay calculators in Cadence use algorithms similar to, derived from, or inspired by this work. We also use similar methods on multi-port extracted networks for the analysis of coupled noise (see Shepard, *et al.* [1]) in our noise analysis products Pacific™ and Celtic™.

### 3.2 Circuit Simulation, Analysis, and Synthesis

SPICE level analysis of RF circuits has always presented special problems. The paper “Nonlinear Circuit Simulation in the Frequency Domain” by Kundert (see page 383) extended ‘Harmonic Balance’ to large circuits and embedded it in SPICE, enabling RF analysis that were previously impractical. While this particular technique is not used in Cadence today, the idea of algorithms specialized for the task of RF design of ICs began with this paper. This led to the current product Spectre-RF™. The history of this technique (and FastHenry, below) is covered in much more detail in the review by Kundert and White.

As frequencies rise, inductance becomes more important in both analog and digital design. The paper “Efficient Techniques for Inductance Extraction of Complex 3-D Geometries” by Kamon, Tsuk, Smithhisler and White (see page 403) introduced FastHenry, a true 3-D inductance solver. Within Cadence, this approach is used directly to model RF packages, and indirectly to calibrate less accurate but faster approximations used in the analysis of digital circuits[2].

Analog synthesis has always lagged far behind digital synthesis. An early attempt was “Automatic Synthesis of OPAMPS on Analytic Circuit Models” by Koh, Sequin, and Gray (see page 313). This work would find solutions to user-defined analytic approximations. This was rapidly followed by “Analog Circuit Synthesis for Performance in OASYS” by Harjani, Rutenbar, and Carley (see page 325), an approach that could handle non-linear designs as well. These techniques are reflected in the few analog synthesis tools available today, such as NeoLinear.

### 3.3 Physical Design

The paper “TILOS: A Posynomial Programming Approach to Transistor Sizing” by Fishburn and Dunlop (see page 295) introduced a practical algorithm for transistor sizing on large circuits. They showed that the problem was convex and hence a local optimum was a global optimum, and gave an algorithm that quickly converged to the local optimum. The algorithm is still in use since it is

Pareto-optimal - all faster algorithms give worse results, and only slightly better results come from much slower approaches.

In the 1980s, most of the commercial placement engines relied on simulated annealing, such as described by Swartz and Sechen[3]. However, these methods were too slow for increasingly large netlists. The paper “GORDIAN: A New Global Optimization/Rectangle Dissection Method for Cell Placement” by Kleinhans, Sigl, and Johannes (see page 499) was the first of the quadratic based approaches that produced quality results comparable to TimberWolf with better run times. It was capable of handling very large designs with reasonable performance, and was in turn the foundation of many further improvements. These include linear wire length optimization[4], detailed placement by network flow algorithms[5], and inclusion of additional objectives[6]. Approaches derived from this work are included in many Cadence products, in particular QPlace<sup>TM</sup>, the block and standard cell placer, and PKS<sup>TM</sup>, the combined synthesis/placement product.

The paper “Exact Zero Skew” by Tsay (see page 509) introduced the idea that zero skew could be obtained without an exactly balanced tree, by connecting two leaves, picking the zero skew tap point, and then performing this recursively up a hierarchy. This was much more practical than previous solutions such as H trees since it could handle arbitrary distributions of non-uniform flip-flops and still give zero skew. Furthermore it lead directly to bounded and/or associative skew approaches that were more useful yet[7]. These ideas have made their way into all the Cadence products that generate clock trees, including CTGEN, CTPKS, and Silicon Ensemble<sup>TM</sup>.

Layout extraction gives device level netlists, but many operations, such as noise analysis and cell characterization, wish to understand the gate level behavior of a circuit. The conversion from transistor level netlists to gate netlists was initially performed by pattern matching, but this always suffered from incomplete pattern libraries. A better but more challenging approach is to derive the gate level behavior from the circuit. This was pioneered by Bryant in “Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis” (see page 337) and his earlier work on COSMOS. These ideas are used in Cadence noise analysis products such as Pacific and Celtic, and the cell characterization included in the delay analysis tool SignalStorm<sup>TM</sup> and the IR drop analysis tool VoltageStorm<sup>TM</sup>.

### 3.4 Logic Verification

The paper “Automating the Diagnosis and the Rectification of Design Errors with PRIAM” by Madre, Coudert, and Billon (see page 17) was a precursor of the “Modified Framework...” paper below. Although the techniques of this paper

are not used directly today, in retrospect it paved the way to more advanced and practical applications.

The paper “A Modified Framework for the Formal Verification of Sequential Circuits” by Couder and Madre (see page 39 ) is the most important ICCAD paper in the formal verification space. It is the first published work, following its presentation at CAV’89[8], of a model checking method for state machines that is based on binary decision diagrams (BDDs). Using BDDs for model checking is largely held responsible for moving model checking from the academic domain to the commercial domain. Today, it shows up in many products such as equivalence checkers and formal tools. By using BDDs for model checking, the size of problem that could be handled automatically was dramatically increased, to a size that made the practice commercially viable. In an historical sense, the descendants of this work include Cadence products such as the formal verifier FormalCheck<sup>TM</sup> and other products currently in development.

The paper “Dynamic Variable Ordering for Ordered Binary Decision Diagrams” by Rudell (see page 51) was the next big contribution. Once BDDs had shown their value, they also showed an annoying sensitivity to variable order. Rudell showed that the order could be optimized dynamically. This work laid the base for all commercial model checkers, as without dynamic reordering, the tools would not be practical.

The paper “GRASP - A New Search Algorithm for Satisfiability” by Silva and Sakallah (see page 73) was one of the early big improvements in SAT solvers, opening the way to many other heuristic improvements – it was one of the early demonstrations of the value of heuristics; today SAT solvers are ubiquitous in industry. In verification, today they offer an alternative to BDDs for certain types of model checking.

### 3.5 Synthesis

The paper “Multiple-Level Logic Optimization System” by Brayton, Djenens, Krishna, Ma, McGeer, Pei, Phillips, Rudell, Segal, Wang, Yung, and Sangiovanni-Vincentelli (see page 191) is the basis for SIS which is the basis for logic optimization in the industry.

The paper “A Method for Concurrent Decomposition and Factorization of Boolean Expressions” by Vasudevamurthy and Rajsiki (see page 227) is the 2-cube kernel extraction. It made the kernel extraction from (above) feasible and far more practical for complex designs. It became part of standard SIS, and we also use it in our synthesis tools BuildGates<sup>TM</sup> and PKS. Also, the low-power logic optimization in these products uses this 2-cube kernel extraction.

Also concerning power reduction, the paper “Hyper-LP: A Design System for Power Minimization using Architectural Transformations” by Chandrakasan, Potkonjak, Rabaey, and Brodersen (see page 117), was one of the first papers

to address power minimization, a crucial field today. Almost all power sensitive synthesis, including the current Cadence offerings, follows from this work in spirit if not in detail.

## 4. Summary

ICCAD is much more than a purely academic conference. As shown in this paper, and similar papers from other companies, industry has contributed a great deal to ICCAD and received a wealth of good ideas in return. Almost all commercial CAD products, from Cadence and others, incorporate ideas that were first reported at ICCAD. Since almost all chips are designed using one or more CAD tools, practically every chip built today incorporates some ideas from the conference. Since these chips are used in almost all modern electronics, and modern electronics are everywhere, this means that every day a good fraction of the world's population uses the practical fruit of ICCAD. Not many conferences can make that claim!

## Acknowledgments

The author would like to thank Krishna Belkhale, Hermanus Arts, Vinod Kariat, Joel Phillips, Ken Kundert, and Bob Kurshan for their cogent explanations of how the original research as reported at ICCAD is reflected in our current products.

## References

- [1] Shepard, K.L.; Narayanan, V., Noise in deep submicron digital design, ICCAD-96, Pages: 524 -531
- [2] Shepard, K.L.; Zhong Tian, Return-limited inductances: a practical approach to on-chip inductance extraction *IEEE Trans. on CAD*, Volume 19, Issue 4, April 2000, Pages: 425 -436
- [3] Swartz, W.; Sechen, C., New algorithms for the placement and routing of macro cells, ICCAD-90, Pages: 336 -339
- [4] Sigl, G.; Doll, K.; Johannes, F.M., Analytical placement: a linear or a quadratic objective function? 28th ACM/IEEE Design Automation Conference, 1991, Pages: 427 -432
- [5] Doll, K.; Johannes, F.M.; Sigl, G. Accurate net models for placement improvement by network flow methods, ICCAD-92, Pages: 594 -597
- [6] Eisenmann, H.; Johannes, F.M. , Generic global placement and floorplanning Proceedings of the Design Automation Conference, 1998, Pages: 269 -274
- [7] Cong, J.; Kahng, A.B.; Koh, C.-K.; Albert Tsao, C.-W, Bounded-skew clock and Steiner routing under Elmore delay ICCAD-95, Pages: 66 -71
- [8] O. Coudert, C. Berthet, and J. Madre., Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, CAV 89: Automatic Verification Methods for Finite-state Systems, Lecture Notes in Computer Science 407, pages 365- 373. Springer-Verlag, 1989.

# ICCAD AND FUJITSU CAD ACTIVITIES

Hiromu Hayashi

*Fujitsu Laboratories Ltd*

## Abstract

This article describes the impact of ICCAD on Fujitsu, along with Fujitsu's contribution to ICCAD.

## 1. Introduction

Fujitsu Ltd., Fujitsu Laboratories Ltd. (FLL), Fujitsu Laboratories of America Inc. (FLA), and other subsidiary companies related with VLSI design and CAD development congratulate ICCAD for 20 years of innovation in design automation.

CAD is the key technology for VLSI design and therefore has been one of Fujitsu's focused research areas. Started in the 1970s, Fujitsu CAD research now covers a wide spectrum of activities including high level design, verification, simulation, logic synthesis, floorplanning, placement, routing, and test. Fujitsu Laboratories of America, Inc. (FLA), headquartered in the heart of Silicon Valley, was established in 1993. It triggered close collaboration between CAD researchers in US and Japan. The CAD research group is one of the most important departments in FLA. The research essential for system-on-a-chip in deep sub-micron (DSM) era is conducted and practical tools based on these research results are developed, both in US and Japan.

ICCAD is the best technical conference in design automation. Fujitsu has been contributing to ICCAD since 1987, when our first ICCAD paper appeared. Our researchers have served as speakers, session chairs, moderators, organizers, and members of technical program committees on several occasions. In addition to providing them with opportunities to present new research on basic VLSI CAD technologies, ICCAD has also been an invaluable source of ideas, technology, inspiration, and feedback to our research and development teams. This article presents the impact of ICCAD on Fujitsu CAD and also Fujitsu's contribution to ICCAD.

## 2. Verification

Twenty years ago, the word "verification technology" meant simulation technology. Some acceleration techniques such as dedicated hardware for simulation [7] were proposed. Although not widely accepted from an academic viewpoint, these have been providing effective verification solutions as emulators for a long time.

On the contrary, formal verification techniques were only in the realm of pure academic research. A big turning point came in the mid 1980s with the introduction of the Binary Decision Diagrams (BDDs). The potential of BDDs was first presented at ICCAD in 1988 [19]. The approaches on page 29 and page 65 were consolidated into equivalence checking of two large combinational circuits. These two seminal pieces of work form the basis of almost all existing equivalence checking products, including Fujitsu's. Current commercial equivalence checking tools adopt a multi-engine approach, which uses ATPG, linear programming, SAT(see page 73) as well as BDDs. FLA originally developed this multi-engine approach [25].

Model checking also benefited from BDDs [21]. The paper on page 39 is an important contribution in the area of implicit state traversal using BDDs and forms the basis of symbolic model checking. It was followed by a series of more practical applications [10, 9].

BDDs brought with them several interesting research problems, the most important being that of variable (re)ordering. The BDD size depends strongly on the variable order. Techniques in the paper on page 51 and [16, 8] were proposed to address this problem. To handle the size explosion, we also proposed a new representation: Partitioned Ordered BDD or POBDD [2], along with a POBDD-based reachability technique for state machine traversal [1].

Thus the pioneering accomplishments on BDDs and model checking technologies presented at ICCAD brought out several successful research results at Fujitsu. These also led to Fujitsu's in-house equivalence checkers and model checkers, which are widely used today by designers in Fujitsu.

### 3. Logic Synthesis

In mid 1980s logic synthesis focused on multi-level logic circuits. This was made possible largely because of the powerful logic synthesis algorithms and framework provided by "MIS" on page 191. A researcher could quickly implement and evaluate his/her ideas in this framework. We made our in-house logic synthesis tool based on MIS. ICCAD provided several key logic synthesis technologies, which were incorporated in our synthesis tool. MIS showed that restructuring techniques are effective for logic minimization. The kernel extraction technique on page 227 enabled fast and powerful logic decomposition.

Using our logic synthesis tool, we have shown that permissible functions can be represented compactly with BDDs and used effectively for multi-level logic optimization [31]. This work led to remarkable progress in the research on don't cares and BDDs [32, 22]. We have also studied technology mapping for FPGAs [20] and for ASICs [30].

In the age of deep sub-micron technology, timing issues have acquired great importance. [17] provided a useful framework for performance optimization.

We improved upon this framework so that it could handle large industrial circuits [23, 33]. We also made significant contribution to the complexity theory of several delay optimization problems [26, 27]. We showed that certain forms of global fanout optimization and gate resizing problems are NP-complete. We proposed two efficient techniques for delay optimization via gate sizing: an LP-based algorithm [34] and an optimum pseudo-polynomial time algorithm for tree circuits under different rise and fall cell delays [3].

#### 4. Physical Layout

Placement is a key step in physical layout design to reduce the chip size and to improve routability. GORDIAN on page 499 demonstrated that partitioning-based placement can cope with the increasing chip size. The placement tool based on bipartitioning, Loose and Stable Net Removal [15], was developed for ASIC design in FLL. The papers on page 479 and page 535 proposed a systematic way to optimize slicing-tree based floorplan and block packing respectively. These opened up doors to realizing an automatic floorplan generator.

Routability was one of the most important objectives for routing systems in early 1990s. Touch and Cross routing algorithm [18] was proposed for better routability. A massively parallel routing hardware engine consisting of 16K processors was designed to speed up the algorithm for PCB designs. As for ASICs, a routing tool based on Touch and Cross was implemented in early 1990s.

Timing closure has been a hot topic for designers since late 1990s. A timing-driven layout project was launched by FLL and FLA in 1995. It resulted in a new design paradigm, in which logic optimization is interleaved with placement/partitioning refinement and hierarchical global routing. This paradigm was integrated into Fujitsu's ASIC design flow in 1999. As part of this research and development, new layout-aware algorithms for important, layout-friendly logic transforms such as gate resizing, net buffering, generalized demorgan transform, pin permutation, and gate decomposition were proposed [24, 28]. We started a follow-up project in 1999 to develop a more tightly integrated performance-driven layout framework, which enabled several tools to plug-in and communicate with each other through a shared database. A common delay calculator, logic synthesizer, timing-driven placement tool and a global router have been integrated into the framework. ICCAD contributed several key technologies for these tools. For instance, for the logic synthesizer, the two-level minimization paper on page 205 and the multi-level minimization papers on pages 191 & 227 provided the foundation.

The quality of clock tree synthesis is also a key issue for circuit performance. The paper on page 509 proposed the basic idea to control clock skew, which still forms the basis of most of the existing clock tree synthesis tools.

## 5. Test

Fujitsu has been active in testing research since 1975, starting from board-level testing for mainframes. D-algorithm, PODEM, and FAN were improved to make them applicable to large LSIs and board-level designs. To handle the growing complexity of designs, algorithm improvement and parallel processing have been applied. We introduced new test schemes such as RAM Function Test (which carried out data extension of the template pattern for every RAM by ATPG) and Dynamic Function Test (which paid attention to transition faults). We now describe the recent research results in design for testability (DFT), sequential ATPG, and high-level test generation.

DFT is a key technology that facilitates higher fault coverage and lower test cost. In 1995, we introduced the novel technique of cost-free scan [4], which reduced the area overhead of flip-flops by establishing the scan path through combinational logic and replacing scan flip-flops with non-scan flip flops, whenever possible. LSI size is approaching 10 million gates and the performance of LSI-Tester has become a bottleneck. Recently, we invented BAST (BIST Aided Scan Test), which succeeded in compressing the amount of test data (by a factor of 10) and the test time. We also formulated several efficient DFT schemes for low-overhead testing [11, 14, 13]. These techniques resulted in 20-30% savings in test overhead over conventional techniques.

Sequential ATPG still remains a challenging problem. We came up with a technique to significantly improve the performance of diagnostic ATPG for sequential circuits using dynamic fault collapsing [29]. A novel method, binary time-frame expansion, was proposed [5], in which the behavior of a circuit for  $t$  time frames, where  $t \leq n$ , is modeled by unrolling the circuit  $2^0, 2^1, 2^2, \dots, 2^{(\log n - 1)}$  times and combining them. This method outperforms the conventional time-frame expansion by several orders of magnitudes on many designs.

Test pattern generation at higher levels of abstraction is often more tractable than at the logic level because of smaller number of primitives. Efficient RTL ATPG techniques were proposed and integrated into an in-house tool [12], which is 2 to 3 orders of magnitude faster than the logic-level ATPG tools. An efficient methodology was proposed to generate test programs using the instruction-set description of a processor [6]. Experimental results show a substantial CPU time reduction for processor validation.

## 6. Conclusion

System-level design, verification, and ultra DSM problems have become significantly important. Hierarchical design methodologies for very large circuits are necessary but not sufficient. We still have several unresolved problems in

VLSI CAD. Fujitsu would like to continue investigating these challenging problems and contribute to further success of ICCAD.

## Acknowledgments

In preparing this article, the contributions of Nobuaki Kawato, Kaoru Kawamura, Tsuneo Nakata, Yutaka Tamiya, Toshiyuki Shibuya, Takahisa Hiraide, Takashi Mochiyama, and Rajeev Murgai are gratefully acknowledged.

## References

- [1] A. Narayan, A. Isles, J. Jain, R. K. Brayton, and A. L. Sangiovanni-Vincentelli (1997). Reachability Analysis using Partitioned-ROBDDs. In *ICCAD*.
- [2] A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli (1996). Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions. In *ICCAD*.
- [3] A. Oliveira and R. Murgai (2000). An Exact Gate Assignment Algorithm For Tree Circuits Under Rise and Fall Delays. In *ICCAD*.
- [4] C. C. Lin, M. Lee, M. Marek-Sadowska, and K. C. Chen (1995). Cost-Free Scan: A Low-Overhead Scan Path Design Methodology. In *ICCAD*, pages 528–533.
- [5] F. Fallah (2002). Binary Time-Frame Expansion. In *ICCAD*.
- [6] F. Fallah and K. Takayama (2001). A New Functional Test Program Generation Methodology. In *ICCD*.
- [7] F. Hirose, M. Ishii, J. Niituma, T. Shido, N. Kawato, H. Hamamura, K. Uchida, and H. Yamada (1987). Simulation Processor SP. In *ICCAD*, pages 484–487.
- [8] H. Higuchi and F. Somenzi (1999). Lazy Group Sifting for Efficient Symbolic State Traversal of FSMs. In *ICCAD*, pages 45–49.
- [9] H. Iwashita and T. Nakata (1997). Forward Model Checking Techniques Oriented to Buggy Designs. In *ICCAD*, pages 400–405.
- [10] H. Iwashita, T. Nakata, and F. Hirose (1996). CTL Model Checking Based on Forward State Traversal. In *ICCAD*, pages 82–87.
- [11] I. Ghosh, A. Raghunathan, and N. K. Jha (1998). A Design for Testability Technique for RTL Circuits Using Control/Data Flow Extraction. *IEEE Transactions on Computer-Aided Design*.
- [12] I. Ghosh and M. Fujita (2001). Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams. *IEEE Transactions on Computer-Aided Design*.
- [13] I. Ghosh, N. K. Jha, and S. Bhawmik (2000). A BIST Scheme for RTL Controller/Data Paths Based on Symbolic Testability Analysis. *IEEE Transactions on Computer-Aided Design*.
- [14] I. Ghosh, N. K. Jha, and S. Dey (1999). A Low Overhead Design for Testability and Test Generation Technique for Core-based Systems. *IEEE Transactions on Computer-Aided Design*.
- [15] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu (1997). Large Scale Circuit Partitioning with Loose/Stable Net Removal and Signal Flow Based Clustering. In *ICCAD*, pages 441–447.

- [16] J. Jain, W. Adams, and M. Fujita (1998). Sampling Schemes for Computing OBDD Variable Orderings. In *ICCAD*.
- [17] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli (1988). Timing Optimization of Combinational Logic. In *ICCAD*, pages 282–285.
- [18] K. Kawamura, T. Shindo, T. Shibuya, H. Miwatari, and Y. Ohki (1990). Touch and Cross Router. In *ICCAD*, pages 56–61.
- [19] M. Fujita, H. Fujisawa, and N. Kawato (1988). Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams. In *ICCAD*, pages 2–5.
- [20] M. Fujita and Y. Matsunaga (1991). Multi-Level Logic Minimization Based on Minimal Support and its Application to the Minimization of Look-Up Table Type FPGAs. In *ICCAD*, pages 560–563.
- [21] M. Fujita, Y. Matsunaga, and T. Kakuda (1990). Automatic and Semi-Automatic Verification of Switch-Level Circuits with Temporal Logic and Binary Decision Diagrams. In *ICCAD*, pages 38–43.
- [22] M. Fujita, Y. Tamiya, Y. Kukimoto, and K. C. Chen (1991). Application of Boolean Unification to Combinational Logic Synthesis. In *ICCAD*, pages 510–513.
- [23] R. Aggarwal, R. Murgai, and M. Fujita (1997). Speeding Up Technology-Independent Timing Optimization by Network Partitioning. In *ICCAD*, pages 83–90.
- [24] R. Carragher, R. Murgai, S. Chakraborty, T. Shibuya, Y. Kanazawa, M. Prasad, A. Srivastava, N. Vemuri, and H. Yoshida (2001). Layout-driven Logic Optimization. In *DATE Designers' Forum*.
- [25] R. Mukherjee, J. Jain, K. Takayama, M. Fujita, J. Abraham, and D. Fussell (1997). FLOVER: Filtering Oriented Combinational Verification Approach. In *IEEE/ACM International Workshop on Logic & Synthesis*.
- [26] R. Murgai (1999a). On The Global Fanout Optimization Problem. In *ICCAD*, pages 511–515.
- [27] R. Murgai (1999b). Performance Optimization Under Rise and Fall Parameters. In *ICCAD*, pages 185–190.
- [28] R. Murgai (2000). Layout-driven Area-constrained Timing Optimization by Net Buffering. In *ICCAD*, pages 379–386.
- [29] V. Boppana and W. K. Fuchs (1998). Dynamic Fault Collapsing and Diagnostic Test Pattern Generation for Sequential Circuits. In *ICCAD*.
- [30] Y. Matsunaga (1998). On Accelerating Pattern Matching for Technology Mapping. In *ICCAD*, pages 118–123.
- [31] Y. Matsunaga and M. Fujita (1989). Multi-Level Logic Optimization Using Binary Decision Diagrams. In *ICCAD*, pages 556–559.
- [32] Y. Matsunaga, M. Fujita, and T. Kakuda (1990). Multi-Level Logic Minimization Across Latch Boundaries. In *ICCAD*, pages 406–409.
- [33] Y. Tamiya (1999). Performance Optimization Using Separator Sets. In *ICCAD*, pages 191–194.
- [34] Y. Tamiya, Y. Matsunaga, and M. Fujita (1994). LP based Cell Selection with Constraints of Timing, Area, and Power Consumption. In *ICCAD*, pages 378–381.

# ICCAD'S IMPACT IN IBM

John A. Darringer, Leon Stok and Louise H. Trevillyan

*IBM T.J. Watson Research Center*

*Yorktown Heights, NY, USA*

## 1. Introduction

Over the last twenty years, ICCAD has been a major source of innovative ideas and valuable technical interactions for IBM, as well as a showcase for the many IBM advances in design automation. The quality of papers and presentations has been unparalleled, as is exemplified by the selected papers in this book. As a company that depends on advanced design automation tools for its products, and as an innovator in the design automation arena [7], IBM congratulates ICCAD on its 20th Anniversary and applauds the authors, tutorial presenters, and program and executive committees that have made ICCAD the best technical conference in design automation.

Figure 1 highlights some of the major advances in design automation over the last 30 years, from IBM's point of view. Many tools originated in IBM to support its early use of advanced technology, while many were developed outside IBM and reported at ICCAD. In each of the four disciplines, we list the major areas of innovation on the first row, and the IBM and industry tools that benefited and contributed to this on the second and third row. A more detailed overview of this IBM perspective can be found in [7].

In the following pages, we look at those areas and highlight the papers that have had an impact on IBM. Certainly, many of the papers in this book fall into this category, as do many other excellent papers which have appeared in the proceedings over the years.

## 2. Verification

Verification gates the delivery of IBM's products. It is of overwhelming importance and consumes the largest portion of our product development effort.

The use of equivalence checking is an important complement to simulation. Since the late 1970's, the ability to prove the equivalence of RTL design and lower-level implementations has allowed IBM to use efficient register-transfer-level simulation instead of much more costly gate-level simulation. Although equivalence checking in IBM [2] predated ICCAD, the use of reduced ordered binary decision diagrams (BDDs), has supplanted our earlier methods. Variable ordering is especially important in forming usable BDDs and the Rudell paper [30] was a fundamental work in describing efficient and effective methods for good variable orderings.

Synthesis	PLA	Local Transforms		Global Flow	Redundancy Removal	Incremental Synthesis	Placement Driven Synthesis
Circuit Analysis	MINI	LSS		BooleDozer			PDS
	Espresso			MIS	Design Compiler	FPGA	
1970	75	80	85	90	95	00	
Bipolar GateArray 704 gates/Chip	5K	CMOS GateArray 60K		Standard Cell 300K 1M		Soc 3M 10M 24M	40M
Physical Design	Cut based	Annealing	Multi-Level Partitioning	Quadratic	Force Directed	Physical Synt	
Formal Verification	MCplace Cplace			Xq	PDS Verity Rulebase		
	Boolean Compare SAS				Design Verifier	Model Checking FormalCheck	

Figure 1. 30 years of algorithm and tool development.

The definition of “cut points” in a logic network [33] was key to practical use on large industrial designs, since it allowed computationally expensive equivalence problems to be reduced to smaller ones.

Checking equivalence between RTL and transistor-level net-lists has been especially valuable for custom microprocessor designs [3]. Bryant [36] used four-valued symbolic analysis to perform such an extraction. This work, together with related efforts, has made formal equivalence checking a major contribution to industry that has helped contain the cost of verification.

Model checking has also become a valuable adjunct to simulation. IBM’s RuleBase system [12] started with the work of McMillan [11] and has developed into a robust system with satisfiability solvers such as GRASP [32]. It is now frequently used on large complex designs to identify subtle errors early in the design process.

### 3. Logic Design, Timing and Test

In the 1960’s and 1970’s, transistor and gate networks were designed “by hand” and then simulated to confirm functional and timing correctness. The

convergence of logic synthesis, equivalence checking and static timing analysis, first used by IBM in the early 1980's [5], [2], [4], eliminated much tedious work and opened the door for dramatic designer productivity gains.

One of the most influential logic synthesis systems continues to be MIS [19], the multilevel optimization system developed at IBM and U. C. Berkeley. Fundamental innovations in optimization methods such as weak division, technology mapping and rectangle covering for factoring came out of MIS and provided a "gold standard" for optimization work by a generation of researchers and commercial tool developers.

While MIS used algebraic optimization methods, the first production logic synthesis system, LSS [5], relied more on "local transformations" to generate production-quality designs of product chips with manageable runtimes. IBM followed LSS with BooleDozer [8], which incorporated ideas from MIS, and added many other innovations, such as an extended form of the global flow analysis techniques [21] that were pioneered in LSS. In another extension, Brand [31] used a test-based approach to verification, that focused on determining where changes have occurred, allowing the unchanged part of the design to remain stable. This was especially helpful for handling engineering changes in the synthesis process. Still further advances were based on the work of Rajski [22], which combined the decomposition and factorization steps into a single algorithm, and Watanabe [23], which did the same with decomposition and technology mapping.

Retiming can effectively optimize logic by moving latches in a circuit to balance timing and minimize cycle time. Leiserson and Saxe [6] established the basic principles for retiming, and Shenoy and Rudell [34] made it practical and efficient to apply to large-scale networks. In IBM, we have achieved significant design optimization using this method, but it has not been widely adopted because it interferes with our verification methodology.

In the 1970's, IBM pioneered the development of PLAs and supporting tools such as MINI [13]. Since then there has been much research in this important area [20]. PLA circuits and tools are still used in some high-performance designs.

Overall, logic synthesis has certainly had the greatest impact in the electronics industry. It has allowed raising the level of design above the gate level and enabled much higher designer productivity. We all look forward to the next advance to the system level.

#### 4. Physical Design

The Gordian system [26] sparked research in the area of cell-placement for very large chips. It introduced the principle of treating all cells simultaneously throughout placement. Algorithms based on this principle have been very ben-

eficial for large-scale ASIC designs in IBM. Min-cut algorithms are being used extensively for iterative improvements. Yang and Wong [29] show that network flow algorithms can be efficiently applied to large-scale partitioning problems.

In more hierarchical designs, approaches based on rectangle packing have gained importance. The sequence-pair representation [27] is fundamental to most of these approaches. Simulated Annealing was developed in IBM and has been widely applied to improve designs incrementally and make them meet complex optimization criteria [10]. As an example, annealing is used in an hierarchical design-planning system for early floorplanning [25]. Clock-tree generation is an integral step in physical design today. Many of IBM's clock-tree generation tools are based on the zero-skew principles laid out by Tsay [24].

The improved capacity of physical design tools has allowed increasingly larger designs to be created. Physical verification tool capabilities need to be kept in line with these larger design sizes. Goalie [28] introduced a fundamental region analysis algorithm and "union-find" data structures that have allowed many of the operations intrinsic to design-rule checking and circuit extraction to be carried out very efficiently.

## 5. Circuit Design

Automated circuit tuning is extremely important for IBM's microprocessor design groups. The posynomial approach to transistor sizing [35] marked the beginning of the automated circuit tuning era. Simplified delay models enabled the use of fast algorithms to solve the sizing problem. JiffyTune [37] allowed for circuit tuning in the presence of complex delay models. JiffyTune is a dynamic circuit tuning tool and has been used on many custom circuits in IBM. One of the key elements is the fast circuit simulator, SPECS, [38], which efficiently provides time-domain sensitivities. To use dynamic tuning, it is necessary to specify all input patterns which tuning is to consider. As a result, most IBM designers have moved to static circuit tuning, which is based on the same sensitivity calculation engine, SPECS, and a static timing analyzer EinsTLT [43]. The progress from manual circuit design to automated transistor-level tuning is truly remarkable and a major productivity improvement.

## 6. Analysis

Analysis tools are essential in any design project to guide designer decisions and to provide the foundation of later automation. Over the last 20 years there has been an increasing focus on interconnect analysis. IBM's early use of high-performance packaging for its servers required considerable pioneering work in interconnect modeling, including inductance [9]. Today, many of these techniques have been adapted for chip designs.

O'Brien and Savarino [18] were among the first to recognize the need for reduced models of interconnect networks and developed their Pi models for specific networks. The AWE work of Rohrer and Pileggi [14] provided a more formal and general framework that has seen many significant enhancements, including the PRIMA work [17]. These model-reduction methods are incorporated in IBM's timing and noise analysis tools. The "Fast Henry" work of Kamon, et. al. [16] served as the performance benchmark for our extraction tools.

While design in IBM is primarily digital, the development of SiGe technology has led to a growth in RF and analog design and the need for effective tools. Fortunately, many of the ICCAD advances in this area, such as Kundert's harmonic balance work [15] have found their way into commercial tools that we use internally and support for our customers.

## 7. System Design

System design in IBM typically refers to the creation of very large multiprocessor complexes with shared memory and banks of peripherals. IBM's EDS was the first design system to attempt to extend automation beyond chips to packages, cards, boards and cables [7]. Today, complex systems are being developed on single chips and the industry needs a new set of system-level tools to help designers make critical tradeoffs early in the design process.

The IMEC work on DSP compilers called attention to the importance of optimizing software for emerging DSPs and greatly influenced commercial DSP compilers that are essential for today's embedded systems. Jacome [42] gave an excellent example of an optimization algorithm that can be developed given the appropriate framework for considering tradeoffs

As "systems on a chip" become more complex and chips become larger, still more automation will be required to manage system design. The work by Carloni, et. al. [41] described a correct-by-construction method that could help in organizing such chips in the future.

Power dissipation has emerged as a critical concern for designers and power optimization is needed at all stages of design. The Hyper-LP work [39] provided an early focus on power and described an broad set of architectural and logic transformations for reducing power. These transformations have been widely used and rediscovered in later tools. Malik et. al. [40] introduced the notion of "power cost of software" and provided a simple, but important, model for optimization and considering design tradeoffs.

## 8. Conclusions

ICCAD has served as an important beacon of future EDA advances for 20 years. Many critical ideas were first presented there. But there are many chal-

lenges ahead and more breakthroughs are needed. We look forward to further advances being reported at future ICCAD conferences.

## Acknowledgments

The comments here are those of the authors, but we have been greatly helped by input from Abe Elfadel, Peter Feldmann and Chandu Viswesvariah.

## References

- [1] F.E. Allen and J. Cocke, "A Program Data Flow Analysis Procedure", *CACM*, vol. 19, no. 3, pp. 137-147, March, 1976.
- [2] G. L. Smith, R. J. Bahnsen, H. Halliwell, "Boolean Comparison of Hardware and Flowcharts", *IBM J. Res. Develop.*, Vol. 26, January 1982, pp. 106-116.
- [3] A. Kuehlman et al., "Verity - A formal verification program for custom CMOS circuits", *IBM J. Res. Develop.*, Vol. 39, No. 1/2, Jan/March 1995, pp. 149-165.
- [4] R. B. Hitchcock, G.I. Smith, D.D. Cheng, "Timing Analysis of Computer Hardware", *IBM Journal RD*, v26, No 1, pp. 100 - 105, January 1982
- [5] J. A. Darringer, W. H. Joyner, "A New Look at Logic Synthesis", *Proc. 17th Design Automation Conf.*, June 1980, pp. 543-549.
- [6] C. Leiserson and J. Saxe, "Optimizing synchronous systems", *J. of VLSI and Computer Systems*, vol. 1, pp. 41-67, 1983.
- [7] J. A. Darringer, et al., "EDA in IBM: Past, Present and Future", *IEEE Trans. on CAD*, Vol. 19, No. 12, Dec. 2000.
- [8] L. Stok, D.S. Kung, D. Brand, A.D. Drumm, A.J. Sullivan, L.N. Reddy, N. Hieter, D.J. Geiger, H.H. Chao, and P.J. Osler, "BooleDozer: Logic Synthesis for ASICs", *IBM Journal of Research and Development*, vol. 40, no. 4, pp. 407-430 (July 1996).
- [9] A.E. Ruehli and P.A. Brennan, "Efficient capacitance calculations for three-dimensional multiconductor systems," *IEEE Trans. Microwave Theory Tech.*, Vol. MTT-21, Feb. 1973, pp. 76-82.
- [10] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, Volume 220, Number 4598. May 13, 1983
- [11] K. L. McMillan, "Symbolic Model Checking", ISBN: 0-7923-9380-5, Kluwer, 1993
- [12] I. Beer et al., "RuleBase: an Industry-oriented Formal Verification Tool", *Proc. of the 33rd Design Automation Conf.* 1996, pp. 655-660.
- [13] S.J. Hong, R.G. Cain, D.L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization", *IBM Journal of Research and Development*, vol 18, No 5, pp. 443-458, Sept 1974.
- [14] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis", *IEEE Trans. Computer Aided Design*, 9: 352-366, April 1990.
- [15] K. S. Kundert, A. Sangiovanni-Vincentelli, "Nonlinear Circuit Simulation in the Frequency Domain," in *International Conference on Computer Aided Design*, pp. 240–242, 1985.
- [16] M. Kamon, M. J. Tsuk, C. Smithhisler, T. White, "Efficient techniques for inductance extraction of complex 3-D geometries," in *International Conference on Computer Aided Design*, pp. 438–441, 1992.

- [17] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodeling algorithm," in *International Conference on Computer Aided Design*, pp. 1–9, 1997.
- [18] P. R. O'Brien and T. L. Savarino, "Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation," in *International Conference on Computer Aided Design*, pp. 512–515, 1989.
- [19] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Oei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, and A. Sangiovanni-Vincentelli, "Multiple-level logic optimization system," in *International Conference on Computer Aided Design*, pp. 356–359, 1986.
- [20] R. Rudell and A. Sangiovanni-Vincentelli, "Exact minimization of multiple-valued functions for pla optimization," in *International Conference on Computer Aided Design*, pp. 352–355, 1986.
- [21] L. Berman and L. Trevillyan, "Improved logic optimization using global flow analysis," in *International Conference on Computer Aided Design*, pp. 102–105, 1988.
- [22] J. Vasudevamurthy and J. Rajski, "A method for concurrent decomposition and factorization of boolean expressions," in *International Conference on Computer Aided Design*, pp. 510–513, 1990.
- [23] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," in *International Conference on Computer Aided Design*, 1995.
- [24] R.-S. Tsay, "Exact zero skew," in *International Conference on Computer Aided Design*, pp. 336–339, 1991.
- [25] R. Otten, L. van Ginneken, "Floorplan Design Using Simulated Annealing," in *International Conference on Computer Aided Design*, pp. 96–98, 1984.
- [26] J. M. Kleinhans, G. Sigl, F. M. Johannes, "GORDIAN: A New Global Optimization/Rectangle Dissection Method for Cell Placement," in *International Conference on Computer Aided Design*, pp. 506–509, 1988.
- [27] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "Rectangle-Packing-Based Module Placement," in *International Conference on Computer Aided Design*, 1995.
- [28] T. Szymanski, C. Van Wyk, "GOALIE: A Space-Efficient System for VLSI Artwork," in *International Conference on Computer Aided Design*, pp. 278–280, 1984.
- [29] "Efficient network flow based min-cut balanced partitioning," H. Yang, D. F. Wong in *International Conference on Computer Aided Design*, 1992.
- [30] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *International Conference on Computer Aided Design*, pp. 42–46, 1993.
- [31] D. Brand, "Verification of large synthesized designs," in *International Conference on Computer Aided Design*, pp. 534–537, 1993.
- [32] J. P. Silva and K. A. Sakallah, "GRASP - a new search algorithm for satisfiability," in *International Conference on Computer Aided Design*, pp. 220–227, 1996.
- [33] L. Berman and L. Trevillyan, "Functional comparison of logic designs for vlsi circuits," in *International Conference on Computer Aided Design*, pp. 456–459, 1989.
- [34] N. Shenoy, R. Rudell, "Efficient implementation of retiming," in *International Conference on Computer Aided Design*, 1994.
- [35] J. P. Fishburn, A. E. Dunlop, "TILOS: A Posynomial programming approach to transistor sizing , " in *International Conference on Computer Aided Design*, pp. 326–328, 1985.

- [36] R. E. Bryant, "Extraction of gate level models from transistor circuits by four-valued symbolic analysis," in *International Conference on Computer Aided Design*, pp. 350–353, 1991.
- [37] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, and C. Visweswarah, "Optimization of custom MOS circuits by transistor sizing," in *International Conference on Computer Aided Design*, 1996.
- [38] C. Visweswarah, R. A. Rohrer, SPECS2: an integrated circuit timing simulator in *International Conference on Computer Aided Design*, pp. 94–97, 1987.
- [39] A. P. Chandrakasan, M. Potkonjak, J. Rabaey, R. W. Brodersen, "Hyper-LP: A Design System for Power Minimization using Architectural Transformations," in *International Conference on Computer Aided Design*, pp. 300–303, 1992.
- [40] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," in *International Conference on Computer Aided Design*, 1994.
- [41] L. Carloni, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, "A Method for correct by construction latency insensitive design," in *International Conference on Computer Aided Design*, pp. 309–315, 1999.
- [42] M. F. Jacome, G. de Veciana, V. Lapinskii, "Exploring Performance Tradeoffs for Clustered VLIW ASIPs," in *International Conference on Computer Aided Design*, 2000.
- [43] V.B. Rao, J.P. Soreff, T.B. Brodnax and R.E. Mains, "EinsTLT, transistor level timing with EinsTimer", in: *Proc. TAU*, December 1999.

# MAGMA AND ICCAD

Michel Berkelaar  
*Magma Design Automation*

## 1. Introduction

Magma Design Automation, founded in 1997, sells a complete IC design system (from RTL to GDSII in one tool) under the name Blast Fusion<sup>TM</sup>. This tool contains RTL HDL (VHDL or Verilog) entry with RTL synthesis and data path generation, floorplanning, netlist level optimization, placement and routing, clock optimization, as well as a multitude of analysis engines. Among these the most important are: an incremental static timing analyzer, parasitic extraction, crosstalk noise analysis, power analysis, rail (voltage drop) analysis.

This tool is based on two fundamental concepts: The Magma Unified Data Model and the FixedTiming ® Technology. The Magma Unified Data Model concept means that Magma's design flow consists of a single executable, which stores all design information in one in-memory database. As a result, all design steps have constant access to all data. Synthesis and optimization steps have access to the placement and routing information, for example. There is no need for data exchange through external files during the entire design flow, which also means that no data is lost. This allows a design flow in which the design is incrementally refined, and in which traditionally sequential operations (such as logic synthesis and placement) can be interleaved.

The FixedTiming Technology is fundamental to the way in which the timing of a CMOS circuit is calculated during a large part of the design flow in Magma's tools. It uses concepts from research known as *constant delay* and *logical effort*, on which fundamental papers have been published in ICCAD in the past. Please refer to the section on this topic below.

## 2. Why is ICCAD important to Magma?

ICCAD is important to Magma for a number of reasons. First and foremost, it clearly attracts the best papers written by academic and industrial researchers on the subject of Computer Aided Design for Integrated Circuits. There are other conferences on this topic, but ICCAD always stands out for the high technical quality of the program. These papers often contain new ideas that can inspire industrial researchers. If they turn out to be practical in an industrial setting, they can make their way into industrial tools. In general, however, academic solutions need to be heavily adapted, as real-world circuits are much bigger and more complex in terms of structure (dozens of (generated) clocks, false-path and multi-cycle constraints, etc.) than those used in the results section of most papers. For this reason, many papers that show great results on existing

academic benchmark sets turn out to be completely impractical for the design of real ICs.

During the ICCAD meeting itself, always strategically placed in the heart of Silicon Valley, every year a large number of the important researchers and industrial representatives for the topic gather, making it the prime meeting place to stay in touch with colleagues from all over the world. The world's most promising Ph.D. students give presentations at ICCAD, making it also a very attractive (and intensively used) hunting ground for new employees.

### 3. Papers important for the algorithms inside Magma's tools

In the following sections, we will look at which basic ideas and associated papers or books have been used in Magma's tool suite. In all cases, one should realize that the actual implementations are heavily engineered to make them efficient and robust enough, as well as deliver the right level of quality.

Many basic ideas predate ICCAD, so the most fundamental reference is not an ICCAD reference. In other cases, fundamental ideas were first published in other conferences, journals or books, but they are still mentioned here. In those cases, often ICCAD papers (sometimes many) do exist that take these ideas further.

### 4. FixedTiming Technology

For timing analysis and optimization during much of the flow, Magma uses the concepts of *constant delay* and *logical effort* as developed in [12] and [21]. These ideas are fundamental to the efficiency and accuracy of Magma's timing optimization solution, and allow the optimization of huge circuits flat. If not for the inspiration derived from these ideas, Magma might not have been founded as a company.

### 5. Formal Verification

Magma does not market a formal verification tool, but an internal equivalence checker implementation exists for Quality Assurance checking. For an EDA company, this is a very important task. EDA customers do not take verification errors lightly. [2, 3, 17, 4] are the most fundamental inspirers of our implementation. A good overview of this subject can be found in [15].

### 6. RTL synthesis

For optimizations at the RTL level, the introduction of *Data Flow Graphs* or *DFGs* as an input-language independent intermediate representation that com-

bines data and control flow [10] was very important. This concept is applied in many RTL front-end tools, and Magma's front end is no exception.

## 7. Timing Analysis

For timing analysis during optimization, Magma's tools rely on the fundamental concept of *static timing analysis*. [14] was probably the first paper to introduce this idea, and a good overview of this topic can be found in [16].

## 8. Logic Synthesis

For logic synthesis, the basic idea of *optimization by algebraic transformations on Boolean expressions* as developed by researchers at IBM and later in the University of California, Berkeley was fundamental to efficient and effective implementations. At the University of California, Berkeley, the famous MIS package was built using these ideas (see page 191 of this book), which served as a reference implementation that inspired many industrial logic synthesis tools. A more comprehensive description of the ideas used in MIS can be found in [5].

## 9. Clock tree synthesis

For automatic synthesis of clock trees two basic principles are used in Magma's flow. The first is the construction of a clock tree with minimal skew, of which the implementation is based on H-trees as introduced in [7]. We also use the more advanced notion of *useful skew*, where non-zero clock skew is introduced on purpose to optimize delay. Our implementation was inspired by [8].

## 10. BDDs

*Binary Decision Diagrams*, better known under the acronym *BDDs*, form a basic data structure to store Boolean functions. Their use can be found in several places in Magma's tools. Formal verification and technology mapping are two good examples. Fundamental were both the first papers to employ BDDs to represent Boolean functions [6], as well as the idea of efficient dynamic variable ordering to control their size (paper on page 51 of this book).

## 11. Place and Route

Over the past decades, ICCAD has been a very useful catalyst for the development of back-end (Place and Route) technology. The nature of back-end tools makes it hard to single out individual papers that have changed the back-end landscape. Good place and route technology is the result of experience and know-how. It is much less the result of a single algorithm, rather a carefully tuned flow along a few conventional algorithms. Magma employs several dif-

ferent placement algorithms in its flow. The basic ideas from [11] are used for initial placement. Ideas from [13] are used for incremental placement and the paper on page 479 of this book ([20] gives a comprehensive overview) is used for detailed placement.

The most common routing algorithms are variations on Dijkstra's vertex expansion algorithm [9].

Placement and routing is not possible without the efficient storage and retrieval of enormous amounts of 2-dimensional objects, and KD-trees are a fundamental data structure used for this. They were introduced by Jon Bentley in [1].

## 12. Power analysis

For power analysis in CMOS circuits, it is very important to know the *switching activities* in the circuit. [18] introduced the first practical algorithm to estimate these activities, and a very similar algorithm is one of the options a Magma user has to estimate these activities.

## 13. Conclusion

From the above it is clear that ICCAD has been very important to Magma in many respects. We certainly hope and expect it will be the prime meeting place for the EDA R&D community for years to come.

## References

- [1] Bentley, J.L., "Multidimensional Binary Search Trees Used for Associative Searching", Communications of the ACM, Vol. 18, 9, pp. 509-517, September 1975.
- [2] Berman, C.L, "On Logic Comparison", Proc. 18th ACM/IEEE Design Automation Conf., pp. 854-861, 1981
- [3] Berman, C.L. and L.H. Trevillyan, "Functional Comparison of Logic Designs for VLSI Circuits", Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 456-459, 1989.
- [4] Brand, D., "Verification of Large Synthesized Circuits", Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 534-537, 1993.
- [5] Brayton, R.K., C.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.
- [6] Bryant, R.E., "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. on Computers, vol. C-35, no. 8, pp. 677-691, August 1986.
- [7] Cong, J., A. Kahng, and G. Robins, "Matching-based methods for high-performance clock routing." IEEE transactions on CAD of ICs, pp. 1157-1169, vol. 12, no. 8, August 1993.
- [8] Deokar, R.B., and S. S. Sapatnekar, "A Graph-theoretic Approach to Clock SkewOptimization", IEEE International Symposium on Circuits and Systems (ISCAS) 1994, Vol. 1, pp. 407-410
- [9] Dijkstra, E.W., "A note on two problems in connexion with graphs", Numerische Mathematik, vol. 1, pp. 269-271, 1959.

- [10] Eijndhoven, J.T.J. van, and L. Stok, "A Data Flow Exchange Standard", Proc. of the European Conference on Design Automation, pp. 193-199, 1992.
- [11] Eisenmann, H.; Johannes, F.M., "Generic global placement and floorplanning", Design Automation Conference, 1998. Proceedings, 1998, pp. 269 -274.
- [12] Grodstein, J., E. Lehmann, H. Harkness, B. Grundmann and Y. Watanabe, "A delay Model for Logic Synthesis of Continuously-Sized Networks", Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 458-462, 1995.
- [13] Kernighan, W and S. Lin, "An efficient heuristic procedure for partitioning graphs", Bell Systems Technical Journal, vol. 49. pp. 291-307, 1970.
- [14] Kirckpatrick, T.I. and N.R. Clark, "PERT as an aid to logic design", IBM Journal of Research and Development, 10(2):135-141, March 1966.
- [15] Kuehlmann, A. and C.A.J. van Eijk, "Combinational and Sequential Equivalence Checking", chapter 14 of Logic Synthesis and Verification, S. Hassoun and T. Sasao, Editors, Kluwer Academic Publishers 2002.
- [16] Kukimoto, Y., M. Berkelaar and K. Sakallah, "Static Timing Analysis", chapter 14 of Logic Synthesis and Verification, S. Hassoun and T. Sasao, Editors, Kluwer Academic Publishers 2002.
- [17] Kunz, W., "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 538-543, 1993.
- [18] Najm, F.N., "Transition Density, a Stochastic Measure of Activity in Digital Circuits", Proc. of the 28th IEE/ACM Design Automation Conference, pp. 644-649, 1991
- [19] Otten, R.H.J.M., L.P.P.P. van Ginneken, "Floorplan design using simulated annealing", Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 96-98, 1984.
- [20] Otten, R.H.J.M., L.P.P.P. van Ginneken, "The annealing algorithm", Kluwer International Series in Engineering and Computer Science 72, 1989, pp. 1-224. ISBN 0-7923-9022-9
- [21] Sutherland, I., B. Sproull and D. Harris, "Logical Effort", Morgan Kaufmann Publishers Inc., 1999.

# **DESIGNERS FACE CRITICAL CHALLENGES AND DISCONTINUITIES OF ANALOG/MIXED SIGNAL DESIGN AND PHYSICAL VERIFICATION**

Walden C. Rhines, Chairman and Chief Executive Officer

*Mentor Graphics Corp., Wilsonville, Oregon*

## **Introduction:**

Mentor Graphics Corporation is recognized as a mainstay in the electronic design automation (EDA) industry, having developed leading-edge products that enable the design of electronic products for more than 20 years. Mentor's breakthrough research in EDA ranges from high-speed board design to resolution enhancement technologies for subwavelength manufacturability.

Mentor's strategic focus in analog/mixed-signal and physical verification and resolution enhancement technologies for system-on-chip (SoC) designs has been positively influenced by the advanced research conducted by contributors to ICCAD, as well as our experience with the real-time, real-work problems faced by IC designers in today's world of complex engineering.

This editorial authored by Walden C. Rhines, chairman and chief executive officer of Mentor Graphics Corp., Wilsonville, Ore. highlights examples of the most critical challenges and discontinuities addressed at ICCAD and that are currently challenging IC designers and EDA vendors alike.

## **Challenges and Discontinuities in Analog/Mixed Signal Design**

The digital chip has dominated industries from automotive to aerospace. But the recent market push in high-bandwidth communications technologies has led to a sharp increase in the use of analog/mixed signal (AMS) chips. Demand for AMS chips is expected to grow by 25 percent in the next few years, which means that engineers will need to shift their technology paradigms to meet the challenges ahead.

To become truly competitive in this fast-paced segment, designers will be required to make the transition to top-down design techniques that take into account the needs of digital and analog designers alike from the beginning of the design. These behavioral modeling methodologies were the topic of a six-person tutorial presented at ICCAD in 1999 [1]. The growing design complexities make it imperative for designers to use the newest analog HDLs for top-down analysis while retaining the bottom-up analysis of the transistor level. The discontinuity

is that analog and digital designers both need to move rapidly to adopt new methods in order to accurately create and test new designs.

For example, many engineers are trying to implement analog features in low-cost digital CMOS chips. Typically, analog and digital subsystems are created separately, do not interact until IC layout and remain untested until fabrication. Any faulty interaction found at this point can result in expensive production delays and possibly in lost market opportunities. Fortunately, there are now tools available that support behavior modeling and standard analog modeling languages as well as mixed signal verification. Key solutions filling the gap include behavioral model libraries, analog HDL languages and design simulators.

Behavioral libraries mimic the behavior of a device and can be implemented at several levels of abstraction. Examples range from a simple op amp to a complex multipole zero op amp. Each model also offers dozens of parameters to enable virtually unlimited customization. If a model is not available to describe a proprietary design, a designer can use an analog HDL such as VHDL-AMS to create new models or write custom code. The development of an HDL reverse engineering tool-set is also a possibility for design data reuse [2].

The current trend toward joint partnerships and purchased intellectual property makes it important to use a simulator that accepts all standard HDLs, including Verilog, Verilog-AMS, VHDL, VHDL-AMS, SPICE and C-level models. Language-independent simulators allow designers to reuse major portions of the analog or digital test bench in the full-chip verification. The models created for this design and refined for the verification are also ideal starting models for the next generation of the product. The models also improve and mature along with each generation. Adapting a model that is simple in design can be the key to quick and complete simulation in a mixed-signal simulator such as Mentor Graphics Eldo software [3].

Leading-edge companies have begun to make the transition from traditional analog tools, but because it is often more difficult to train people than it is to create new software, the industry as a whole has been slow to adopt new generation tools. Both the learning curve and the initial capital investment can be a stretch. However, as the AMS market continues to grow, individual design engineers who have the foresight to increase their skills and the businesses that have the vision to plan for the future will outperform their competitors.

There are many strong business reasons that should compel analog and digital designers working in the AMS space to make the switch to top-down design using the newest developments in HDLs. Ever-increasing chip complexity and time-to-market pressures continue to stretch our current SOC development techniques. System-level design, especially mixed signal design and partitioning, is a major cause of schedule delays. The size of today's analog designs makes it impossible for an engineer to go straight from specification to transistor-level

design. Utilizing top-down design with the new analog HDLs allows designers to increase their productivity.

Designers facing the prospect of complex mixed signal chips should consider their own willingness to make radical shifts when necessary as they plan for the future. With continued acceptance and use of the new AMS tools, we will soon find that we have successfully traversed this discontinuity in analog/mixed signal design and moved on to the next challenge. Even better, a whole set of engineers will be even more indispensable to their employers because they have mastered another design challenge.

## Challenges and Discontinuities in Physical Verification

Several years ago, design engineers were able to simulate and verify designs against a specific set of known realities. About four years ago, however, physical verification hit a major roadblock for those working on very large designs. As feature size continued to shrink, the requirement for more complex design rules grew. The tools in place at the time could not do the job. Designers who thought a job was almost complete were in a bind because they could not finish the verification. This was a discontinuity that design automation companies had to respond to almost without warning, and almost overnight.

The move to deep-submicron work revolutionized everything. New factors had to be considered. As chips became more complex, polygon count exploded exponentially. This called for the creation of a fundamentally new architecture that was hierarchically based so that terabytes of information for physical layout could be compressed down to at least gigabytes. Designers had to move to parallel processing so that they could effectively scale and complete the task without using more memory in the process [4]. Further, as we delved deeper into the subwavelength arena at 0.18 micron and below, we began to play tricks on modern physics.

The EDA industry responded well. Modern photolithography uses wavelengths of light that are larger than the smallest IC feature size to define features with the use of resolution enhancement technologies (RET), including optical proximity correction, phase shift mask, scattering bars and off-axis illumination. As we move into the danger zone at 0.15 micron and below, semiconductor companies will combine different types of RET at different points in the process to extend the life of the lithography equipment as well as to ensure adequate yields. Although microlithography researchers around the world have long been addressing the issues [5], no one can yet predict where we will be in the coming years, although there is the likelihood of adding extreme ultraviolet and fluorine lasers to the RET mix.

One common misconception in the market today is that these factors will create the need to replace the entire EDA tool suite. Several companies suggested that designers would need to replace everything from synthesis to verification overnight and use their new tools because timing closure could never be achieved otherwise. Instead, designers have chosen an evolutionary approach. New generations of physical synthesis tools now make a more efficient use of gates and use buffer insertion to fix some timing issues. These products are changing the way synthesis is done, throwing away 20 to 30 percent of the gates that are unnecessary, looking at placement and bringing in static-timing analysis.

Physical verification proves to us that no one can truly predict the future. Something that looks like a complete technological dead-end at one point, such as subwavelength manufacturing, proves to have revolutionary solutions, while areas where we expect major technological transformation end up functioning fairly smoothly with a few small adjustments.

A few years ago, no one could have predicted how or if we could manage designs below the subwavelength of light, but here we are with tremendous new tools that allow designers to keep breaking down barriers.

## References

- [1] G. Gielen, J. Holmes, K. Lampert, P. Miliozzi, F. O. Eynde, and R. Rutenbar, Tutorial 1: Mixed Signal ASIC Design: CAD, Methodology, Case Studies, International Conference on Computer Aided Design, 1999.
- [2] G. Lehmann, K. Muller-Glaser, and B. Wunder, "Basic Concepts for an HDL Reverse Engineering Tool-Set," International Conference on Computer Aided Design, 1996
- [3] J. Huertas, E. Peralias, and A. Rueda, "Statistical Behavior Modeling and Characterization of A/D Converters," International Conference on Computer Aided Design, 1995
- [4] K. Belkhale, and P. Banerjee, "A Parallel Algorithm for Hierarchical Circuit Extraction," International Conference on Computer Aided Design, 1990
- [5] A. Strojwas, "Design-Manufacturing Interface for 0.13 micron and Below," International Conference on Computer Aided Design, 2000

# NEC AND ICCAD - EDA PARTNERS IN SUCCESS

P. Ashar, S. Chakradhar, A. Gupta, J. Henkel and A. Raghunathan  
*NEC Laboratories America, Inc., Princeton, NJ, USA*

K. Wakabayashi  
*NEC Laboratories, NEC Corporation, Tokyo, Japan*

## 1. Introduction

NEC is a premier semiconductor company with a tradition of strong internal technology and innovation in EDA. NEC has been an active contributor to developments in the EDA community for more than 20 years. In particular, NEC researchers have contributed significantly as organizers, reviewers and program committee members at ICCAD. Notably, Dr. Satoshi Goto of NEC was the Program Chair and General Chair of ICCAD in 1990 and 1991, respectively. NEC has pro-actively developed in-house tools, published papers in premier EDA conferences and journals, and assisted EDA vendors with technology and funds to develop EDA tools. Like other companies with semiconductor operations, NEC has also been a beneficiary of innovations showcased at premier international technical conferences like ICCAD. These innovations have consistently fuelled semiconductor design methodologies world wide across diverse industry segments - whether at the system level (personal computers, hand-held devices etc.) or at the device level (processors, memory, logic, MEMS, RF devices etc.).

## 2. System Design and Test

**System-level design:** NEC is addressing the challenges of SoC design by developing a C-based system design flow that assists system designers in behavioral system specification and simulation, system architecture template definition, behavior-to-architecture (component and communication) mapping, system-level performance and power estimation [1, 2, 3], and automatic optimization of the system architecture through tuning of architectural template parameters [4, 5, 6, 7, 8, 9, 10, 11]. NEC developed the first comprehensive system-level design methodology for on-chip communication. A fast performance analysis technique for bus-based system-on-chip communication architectures was published in 1999 ICCAD [12]. This was followed by a comprehensive methodology for design of communication architectures for system-on-chips [13]. This work led to a Best Paper Award at the ACM/IEEE Design

Automation Conference in the year 2000. NEC also created the ACE-2 initiative to reduce turnaround time for ASIC SoC designs by up to two-thirds, from an average of 450 engineering months to fewer than 150 months. To realize the aggressive goals of ACE-2, NEC teamed with the world's leading electronic design automation (EDA) companies to define a new design methodology for quick and accurate SoC design. We have also developed technologies that automate custom architecture performance analysis [14, 15, 16, 17]. These techniques facilitate the use of extensible processors in SoCs.

**Advanced processing architectures:** Due to limited CPU and battery resources, wireless hand-held devices are unable to provide real-time security processing support. The MOSES project develops a programmable, mobile security processing system that combines a novel hardware architecture with a tamper-proof, flexible software architecture to achieve security protocol acceleration that cannot be achieved by conventional cryptographic algorithm hardware accelerators [18, 19, 15, 20]. The code compression project CoCo compresses instruction code off-line, places compressed code in SystemLSI's memory and decompress the code on the fly, during system run-time [21, 22, 23, 24]. NEC continues to pioneer the discovery of new architectures for on-chip communication and network switching fabrics [27, 28, 29] that have received critical industry acclaim (for example, EE Times featured coverage of FLEXBAR technique [30, 31]). NEC has also developed MP98, a high-performance and low-power microprocessor technology for smart information terminals [32, 33]. NEC has pioneered the development of a unique, dynamically re-configurable processing architecture that finds wide applications in multimedia processing [34, 35].

**High-level and RTL design:** NEC's high-level hardware design flow featured the use of C-based hardware description language [36], well before the recent emergence of languages such as SystemC, providing designers with clear advantages in terms of ease of specification and simulation time compared to traditional HDL-based flows. The CYBER C-based design system [37, 36] has been in use internally in NEC for both ASIC and re-configurable (FPGA) designs for over a decade. It all began with a ground-breaking paper published in the 1989 ICCAD [38]. This was followed by numerous papers [39, 40] that underscored NEC's commitment and conviction to move to higher levels of abstraction. Subsequently, a comprehensive performance-driven HLS flow for control-flow intensive designs was developed [41, 44, 46, 45], which introduced several new concepts and significantly improved the state-of-the-art in high-level synthesis. NEC pioneered a new paradigm for energy and performance optimization. The new paradigm was based on optimizing for the common-case computations in a behavior [42]. This work won the Best Paper Award at the 1999 ACM/IEEE Design Automation Conference. NEC also recognized the fact that behavioral descriptions often consist of multiple con-

current processes, and developed techniques for performance analysis [47] and optimized synthesis [48] of multi-process behaviors.

**Low Power Design:** NEC's RTL power optimization and estimation techniques were one of the first attempts to address power issues at the RTL level [49, 50, 51, 52, 42, 52, 53]. These techniques inspired the research community to focus on RTL power-optimization techniques. NEC has continuously driven the efforts on higher level power optimization techniques [54, 55, 56].

Rather than minimizing power consumption of single component in a system, NEC recognized that the power consumption of a complex SOC can only be minimized when the interdependencies of system components are taken into consideration [4, 5, 2, 57]. NEC also recognized the importance of hw/sw partitioning for low power [6] and the impact of operating system's power consumption [58, 59]. The shift to higher level optimization techniques continued with optimizing the system for low power consumption, first published at ICCAD [8] and later refined into a suite of highly efficient methods [7, 9, 10, 11, 60].

**Test and Design for Testability:** NEC researchers pioneered the concept of scan in the 1960's, and they continue to use it extensively. NEC's test tool suite TRANGEN still enjoys superior performance over commercial EDA offerings. The tool suite is powered by innovations that were first proposed in a 1988 ICCAD paper that advocated use of quadratic programming and SAT-based techniques [73] for test generation as an alternative to the traditional path-sensitization based methods. This technique was championed and further refined [74, 75, 76, 77, 78, 79] in subsequent years by NEC researchers, culminating in the development of TRANGEN tool suite that is still in wide use in NEC for over eight years. NEC researchers proposed static [80, 81] and dynamic [82, 83] test set compaction algorithms that continue to be effective for large designs. NEC's manufacturing test road map includes investigation of design for testability techniques other than full-scan. Over the years, NEC researchers have developed partial scan [84, 85] and synthesis for testability techniques [86, 87, 88, 89, 90] that are in use today in the development of high-speed supercomputers. The partial scan work received the Best Paper Award at the 1994 ACM/IEEE Design Automation Conference [85]. Research in delay testing [91, 92, 93] has culminated in the development of in-house delay testing tool suite that is in use for high-speed designs. NEC researchers have used DFT early in the design process to ensure test coverage and to reduce test development time [94, 95, 96, 97, 81, 98, 93]. NEC has been actively investigating BIST for SOC designs, with particular emphasis on low-overhead [99, 100], low-power consumption (during test application) techniques [101, 102, 103].

### 3. Logic and Physical Design

Seminal papers published at ICCAD allowed design abstraction to transition from schematic entry to automated two-level/multi-level logic synthesis. This allowed designers to produce gate-level netlists that pushed physical design tools to their limits. Papers on MIS, ESPRESSO-EXACT, and global-flow (pages 191, 205 and 217) have been the most influential and a large number of publications (pages 227 and 249) continued the work. Many papers at ICCAD, including the paper on page 235, made significant contributions in targeting FPGAs. Techniques proposed in these papers are probably widely implemented in industry tools for FPGAs. NEC's logic synthesis system Varchsyn was superior to commercial EDA vendor tools for a long time. Research emphasis was on performance optimization of logic circuits using techniques like false-path elimination [61] and novel variations of retiming [62, 63, 64, 65], and logic simulation [66, 67]. Today, NEC has adopted industry-standard logic design tools with limited internal research and development investments in logic design.

ICCAD papers on physical design like the GORDIAN paper (page 499) on quadratic programming techniques for placement and the floorplan design paper (page 479) on application of probabilistic algorithms for physical design have significantly influenced automation in physical design. As timing-driven placement became important, papers like "Exact Zero Skew" on page 509 significantly advanced the state of the art. NEC invested considerable amount of resources in developing superior, proprietary physical design capability. Seminal contributions in channel routing [68, 69] in the early eighties are still in wide use today. For the past two decades, NEC has been developing leading-edge physical design technology and contributing significantly to the placement, global routing and floorplanning phases [70, 71]. A notable contribution on floorplanning [72] recently won the prestigious award for the Best Paper in IEEE Transactions on CAD, 2001.

### 4. Verification

The ability to design is clearly predicated on the ability to verify. Without significant advances in functional verification, increasing the complexity of designs may become a pipe-dream. The EDA community has demonstrated in the last two decades that mathematical techniques can be effective in solving large-scale verification problems that arise in practice.

Technologies developed for logic netlist verification and the verification of gate-level netlists against their RT-level specifications have almost done away with the need for gate-level simulation. It is our belief that this advance alone has saved the semiconductor industry countless dollars and many embarrassments.

Numerous papers on the development of Binary Decision Diagram (BDD) based (e.g. dynamic BDD-variable reordering paper on page 51 that demonstrates the locality of variable order exchange in a BDD) and ATPG/SAT-based techniques (e.g., GRASP paper on page 73 highlighting the benefits of conflict-analysis based non-chronological backtracking and learning) for answering satisfiability questions on logic expressions have of course played major roles in this advance. What was probably more crucial in making these technologies tractable in real life were the kind of ideas proposed in two early papers on pages 29 and 65 demonstrating ways to take advantage of partitioning and circuit similarity.

The next hurdle for the EDA community is to augment simulation with formal methods in the verification of temporal functionality. ICCAD in 1990 had three key papers in the same session on the practical aspects of the application of symbolic methods for state space traversal, one of which is included in this book on page 39. These papers have led to numerous follow on papers on BDD and SAT based techniques that have significantly advanced the ability to check temporal behavior of large designs, to the extent that today design blocks with about a million gates can be analyzed exhaustively for a few hundred clock cycles in a reasonable time at NEC.

Deployment of formal techniques for verification continues to be a major strategic focus at NEC. It is NEC's belief that access to the best verification technology is of vital importance to NEC semiconductor design business. NEC internally introduced formal equivalence checking of netlists using BDD-based techniques long before such tools became commonly available from EDA vendors. NEC's BDD-based ZERO equivalence checking system developed by Akira Mukaiyama was somewhat of a pioneer in that respect. As functional verification has grown in importance, NEC researchers have explored many of its aspects in the last decade and contributed to the state of the knowledge and application (e.g., [104], [105], [106], [107], [108], [109] ). In the recent past, researchers at NEC have developed novel world-leading technologies for analyzing temporal behavior of industry-scale designs that are being deployed within NEC. It is expected that such technologies will be instrumental in significantly bringing down the development time for complex new chips from the current multiple years.

## References

- [1] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, and A. Sangiovanni-Vincentelli, "A case study on modeling shared memory access effects during performance analysis of hw/sw systems," in *Proceedings of the 6<sup>th</sup> IEEE International Workshop on Hardware/Software Codesign*, pp. 117–121, March 1998.
- [2] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "An efficient co-simulation based power estimation framework for system-on-chip design," in *Proceedings of the IEEE DATE*

- 2000, pp. 27–34, March 2000.
- [3] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, “Co-simulation based power estimation for system-on-chip design,” *IEEE Transactions on VLSI Systems*, vol. 10, pp. 253–266, June 2002.
  - [4] Y. Li and J. Henkel, “A framework for estimating and minimizing energy dissipation of embedded hw/sw systems,” *IEEE/ACM 35th. Design Automation Conference (DAC'98) 1998*, pp. 188–193, 1998.
  - [5] J. Henkel and Y. Li, “Avalanche: An environment for design space exploration and optimization of low power embedded systems,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2003.
  - [6] J. Henkel, “A low power hardware/software partitioning approach for core-based embedded systems,” *IEEE/ACM 36th. Design Automation Conference (DAC'99)*, pp. 122–127, 1999.
  - [7] T. Givargis, F. Vahid, and J. Henkel, “System-level exploration for pareto-optimal configurations in parameterized system designs,” *Proc. of IEEE/ACM International Conf. on CAD (ICCAD'01)*, pp. 25–30, 2001.
  - [8] T. Givargis, J. Henkel, and F. Vahid, “Interface and cache power exploration for core-based embedded systems design,” *IEEE/ACM International Conf. on CAD (ICCAD99)*, pp. 270–273, 1999.
  - [9] T. Givargis, F. Vahid, and J. Henkel, “Fast cache and bus power estimation for parameterized system-on-a-chip design,” *IEEE/ACM Conference on Design Automation and Test in Europe Conference (DATE'00)*, 2000.
  - [10] T. Givargis, F. Vahid, and J. Henkel, “Evaluating power consumption of parameterized cache and bus architectures in system-on-a-chip designs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9, No. 4, pp. 500–508, 2001.
  - [11] T. Givargis, F. Vahid, and J. Henkel, “System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2003.
  - [12] K. Lahiri, A. Raghunathan, and S. Dey, “Fast performance analysis of bus-based system-on-chip communication architectures,” in *Proc. Int. Conf. Computer-Aided Design*, 1999.
  - [13] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey, “Communication architecture tuners: A methodology for the design of high-performance communication architectures for system-on-chips,” in *Proc. of Design Automation Conference*, June 2000.
  - [14] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, “Automatic instruction selection for extensible processors,” in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, 2002.
  - [15] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, “System Design Methodologies for a Wireless Security Processing Platform,” in *Proc. ACM/IEEE Design Automation Conf.*, pp. 777–782, June 2002.
  - [16] N. Cheung, S. Parameswaran, and J. Henkel, “Rapid configuration and instruction selection for an asip: A case study,” in *Proceedings of the Design Automation and Test in Europe Conference*, 2003.
  - [17] Y. Fei, S. Ravi, A. Raghunathan, and N. K. Jha, “Energy estimation for extensible processors,” in *Proceedings of the Design Automation and Test in Europe Conference*, 2003.
  - [18] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana, “Algorithm exploration for efficient public-key security processing on wireless handsets,” in *Proc. Design, Automation, and Test in Europe (DATE) Designers Forum*, pp. 42–46, March 2002.

- [19] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana, "Optimizing Public-Key Encryption for Wireless Clients," in *Proc. IEEE Int. Conf. Communications*, pp. 1050–1056, May 2002.
- [20] S. Ravi, A. Raghunathan, and N. Potlapally, "Securing wireless data: system architecture challenges," in *Proceedings of the 15th international symposium on System Synthesis*, pp. 195–200, 2002.
- [21] H. Lekatsas, J. Henkel, and W. Wolf, "Code compression for low power embedded systems design," *IEEE/ACM 37th. Design Automation Conference (DAC'00)*, pp. 294–299, 2000.
- [22] H. Lekatsas, J. Henkel, and W. Wolf, "A decompression architecture for low power embedded systems," *IEEE/ACM Proc. of International Conference on Computer Design (ICCD'00)*, pp. 571–574, 2000.
- [23] H. Lekatsas, J. Henkel, and V. Jakkula, "Design of an one-cycle decompression hardware for performance increase in embedded systems," *Proc. of IEEE/ACM 38th. Design Automation Conference (DAC'02)*, 2002.
- [24] H. Lekatsas, J. Henkel, and V. Jakkula, "1-cycle code decompression circuitry for performance increase of XTENSA-1040-based embedded systems," *Custom Integrated Circuits Conference (CICC'02)*, 2002.
- [25] J. Henkel and H. Lekatsas, "A2BC: Adaptive address bus coding for low power deep sub-micron designs," *Proc. of IEEE/ACM 38th. Design Automation Conference (DAC'01)*, pp. 744–749, 2001.
- [26] T. Lv, J. Henkel, H. Lekatsas, and W. Wolf, "An adaptive dictionary encoding scheme for soc data buses," *Design Automation and Test in Europe Conference 2002 (DATE'02)*, pp. 1059–1064, 2002.
- [27] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "LOTTERYBUS: A novel high-performance communication architecture for system-on-chips," in *Proc. ACM/IEEE Design Automation Conference*, 2001.
- [28] J. Chang, S. Ravi, and A. Raghunathan, "FLEXBAR: a crossbar switching fabric with improved performance and utilization," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 405–408, 2002.
- [29] K. Anjo and et al, "NECoBUS: A high-end SOC bus with a portable and low-latency wrapper-based interface mechanism," in *Proceedings of CICC*, pp. 315–318, May 2002.
- [30] G. Robinson, "Innovative ideas leap design hurdles at CICC," in *EE Times*, May, 16th 2002.
- [31] J. Chang, S. Ravi, and A. Raghunathan, "Flexible I/O speeds data switch," in *EE Times*, May, 13th 2002.
- [32] N. Nishi and et al, "A 1 GIPS 1 W single-chip tightly coupled four-way multiprocessor with architectural support for multiple control flow execution," in *Proceedings of ISSCC*, pp. 418–419, 2000.
- [33] S. Matsushita and et al, "Merlot: A single chip tightly coupled four-way multi-thread processor," in *Cool Chips III*, 2000.
- [34] T. Fujii, K. Furuta, M. Motomura, K. Wakabayashi, and M. Yamashina, "Spatial-temporal mapping of real applications on a dynamically reconfigurable logic engine (DRLE) LSI," in *Proceedings of CICC*, pp. 151–154, May 2000.
- [35] K. Anjo, T. Fujii, K. Furuta, Y. Hirota, H. Ito, M. Mizuno, M. Motomura, Y. Nakazawa, M. Nomura, K. Wakabayashi, and M. Yamashina, "A dynamically reconfigurable logic engine with a multi-context/multi-mode unified-cell architecture," in *Proceedings of ISSCC*, pp. 364–365, 1999.

- [36] K. Wakabayashi, "C-based high-level synthesis system, "CYBER"-design experience-," *NEC Research and Development*, vol. 41, pp. 264–268, July 2000.
- [37] K. Wakabayashi and H. Tanaka, "Global scheduling independent of control dependencies based on condition vectors," in *Proc. of Design Automation Conference*, pp. 112–115, June 1991.
- [38] K. Wakabayashi and T. Yoshimura, "A resource sharing and control synthesis method for conditional branches," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 62–65, 1989.
- [39] K. Wakabayashi and T. Okamoto, "C-based SOC design flow and EDA tools: An ASIC and system vendor perspective," *IEEE Transactions on Computer-Aided Design*, vol. 19, pp. 1507–1522, December 2000.
- [40] K. Wakabayashi, "C-based synthesis experiences with behavior synthesizer CYBER," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 390–393, 1999.
- [41] S. Ravi, G. Lakshminarayana, and N. K. Jha, "Removal of memory access bottlenecks for scheduling control-flow intensive behavioral descriptions," in *Proceedings of the IEEE/ACM International conference on Computer-aided design*, pp. 577–584, 1998.
- [42] G. Lakshminarayana, A. Raghunathan, K. S. Khouri, N. K. Jha, and S. Dey, "Common Case Computation: A high-level power optimization technique," in *Proc. Design Automation Conf.*, pp. 56–61, June 1999.
- [43] G. Lakshminarayana and N. K. Jha, "FACT: A framework for applying throughput and power optimizing transformations to control-flow intensive behavioral descriptions," *IEEE Transactions on Computer-Aided Design*, vol. 18, pp. 1577–1594, November 1999.
- [44] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proceedings of Design Automation Conference*, pp. 491–496, June 1994.
- [45] S. Bhattacharya, S. Dey, and F. Brglez, "Provably correct high-level timing analysis without path sensitization," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 736–742, November 1994.
- [46] S. Bhattacharya, S. Dey, and F. Brglez, "Clock period optimization during resource sharing and assignment," in *Proceedings of Design Automation Conference*, pp. 195–200, June 1994.
- [47] S. Dey and S. Bommu, "Performance analysis of a system of communicating processes," in *Proc. Int. Conf. Computer-Aided Design*, pp. 590–597, November 1997.
- [48] W. Wang, A. Raghunathan, N. K. Jha, and S. Dey, "High-level synthesis of multi-process behavioral descriptions," in *Proceedings of the International Conference on VLSI Design*, January 2003.
- [49] A. Raghunathan, S. Dey, and N. K. Jha, "Glitch analysis and reduction in register-transfer-level power optimization," in *Proc. Design Automation Conf.*, pp. 331–336, June 1996.
- [50] A. Raghunathan, S. Dey, N. K. Jha, and K. Wakabayashi, "Controller re-specification to minimize switching activity in controller/data path circuits," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 301–304, August 1996.
- [51] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer level estimation techniques for switching activity and power consumption," in *Proc. Int. Conf. Computer-Aided Design*, pp. 158–165, November 1996.
- [52] A. Raghunathan, S. Dey, and N. K. Jha, "Power management techniques for control-flow intensive designs," in *Proc. Int. Conf. VLSI Design*, pp. 429–434, 1997.

- [53] S. Ravi, A. Raghunathan, and S. T. Chakradhar, "Efficient RTL Power Estimation for Large Designs," in *Proc. Int. Conf. VLSI Design*, 2003.
- [54] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. on Computer-Aided Design*, vol. 14, pp. 12–31, January 1995.
- [55] G. Lakshminarayana, A. Raghunathan, N. K. Jha, and S. Dey, "A power management methodology for high-level synthesis," in *Proc. Int. Conf. VLSI Design*, pp. 24–29, January 1998.
- [56] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.
- [57] V. Raghunathan, S. Ravi, G. Lakshminarayana, and A. Raghunathan, "Transient power management through high level synthesis," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 545–552, 2001.
- [58] R. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha, "Power Analysis of Embedded Operating Systems," in *Proc. Design Automation Conf.*, June 2000.
- [59] T. K. Tan, A. Raghunathan, and N. K. Jha, "Energy macromodeling of embedded operating systems," in *Proc. International Conference on Computer Design (ICCD)*, October 2002.
- [60] S. Parameswaran and J. Henkel, "I-copes: Fast instruction code placement for embedded systems to improve performance and energy efficiency," *Proc. of IEEE/ACM International Conf. on CAD (ICCAD'01)*, pp. 635–641, 2001.
- [61] P. Ashar, S. Dey, and S. Malik, "Exploiting multi-cycle false paths in performance optimization of sequential circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-aided design*, 1992.
- [62] S. Dey, M. Potkonjak, and S. G. Rothweiler, "Performance optimization of sequential circuits by eliminating retiming bottlenecks," in *Proceedings of the IEEE/ACM International Conference on Computer-aided design*, 1992.
- [63] S. T. Chakradhar, S. Dey, M. Potkonjak, and S. G. Rothweiler, "Sequential circuit delay optimization using global path delays," in *Proceedings of the 30th Design Automation Conference*, 1993.
- [64] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical path minimization using retiming and algebraic speed-up," in *Proceedings of the 30th Design Automation Conference*, 1993.
- [65] S. T. Chakradhar, "Optimum retiming of large sequential circuits," in *Proceedings of the 8th International Conference on VLSI Design*, pp. 135–141, 1995.
- [66] S. T. Chakradhar and V. D. Agrawal, "Logic simulation and parallel processing," in *Proc. Int. Conf. on CAD (ICCAD)*, pp. 496–499, Nov. 1990.
- [67] V. D. Agrawal and S. T. Chakradhar, "Performance analysis of synchronized iterative algorithms on multiprocessor systems," *IEEE Trans. Parallel and Distr. Syst.*, vol. 3, pp. 739–746, Nov. 1992.
- [68] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," in *IEEE Transactions on Computer Aided Design*, vol. Vol. CAD-1, pp. 25–35, 1982.
- [69] T. Yoshimura, "An efficient channel router," in *Proceedings of the 21st Design Automation Conference*, pp. 38–44, 1984.
- [70] M. Edahiro and T. Yoshimura, "New placement and global routing algorithms for standard cell layouts," in *Proceedings of the 27th Design Automation Conference*, pp. 642–645, 1990.

- [71] T. Okamoto, M. Ishikawa, and T. Fujita, "A new feed-through assignment algorithm based on a flow model," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 775–778, 1993.
- [72] P. N. Guo, T. Takahashi, C. K. Cheng, and T. Yoshimura, "Floorplanning using a tree representation," in *IEEE Transactions on Computer Aided Design*, vol. Vol. 20, pp. 281–289, 2001.
- [73] S. T. Chakradhar, M. L. Bushnell, and V. D. Agrawal, "Automatic test generation using neural networks," in *Proc. Int. Conf. on Computer-Aided Design*, (Santa Clara, CA), pp. 416–419, Nov. 1988.
- [74] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell, "Automatic test generation using quadratic 0-1 programming," in *Proc. 27th ACM/IEEE Des. Autom. Conf.*, (Orlando, FL), pp. 654–659, June 1990.
- [75] S. T. Chakradhar and V. D. Agrawal, "A transitive closure based algorithm for test generation," in *Proc. 28th Design Automation Conf.*, pp. 353–358, June 1991.
- [76] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A transitive closure algorithm for test generation," *IEEE Trans. CAD*, vol. 12, pp. 1015–1028, July 1993.
- [77] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG theorems for identifying untestable faults in sequential circuits," *IEEE Trans. CAD*, vol. 14, pp. 1155–1160, Sept. 1995.
- [78] A. Balakrishnan and S. T. Chakradhar, "Software transformations for sequential test generation," in *Asian Test Symposium*, pp. 266–272, November 1995.
- [79] S. T. Chakradhar, S. G. Rothweiler, and V. D. Agrawal, "Redundancy removal and test generation for circuits with non-boolean primitives," *IEEE Trans. CAD*, vol. 16, pp. 1370–1377, Nov. 1997.
- [80] S. K. Bommu, K. B. Doreswamy, and S. T. Chakradhar, "Static compaction using overlapped restoration and segment pruning," in *Proc. of ICCAD*, pp. 140–146, November 1998.
- [81] S. Ravi, G. Lakshminarayana, and N. K. Jha, "Reducing test application time in high-level test generation," in *Proceedings of the international test conference*, pp. 829–838, 2000.
- [82] A. Raghunathan and S. T. Chakradhar, "Acceleration techniques for dynamic vector compaction," in *Proc. of ICCAD*, November 1995.
- [83] S. T. Chakradhar and A. Raghunathan, "Bottleneck removal algorithm for dynamic compaction in sequential circuits," *IEEE Transactions on Computer Aided Design*, vol. 16, pp. 1157–1172, October 1997.
- [84] S. T. Chakradhar, A. Balakrishnan, and V. D. Agrawal, "An exact algorithm for selecting partial scan flip-flops," in *Proc. 31st Design Automation Conf.*, pp. 81–86, June 1994.
- [85] S. T. Chakradhar and S. Dey, "Resynthesis and Retiming for Optimum Partial Scan," in *Proc. of the 31st ACM/IEEE Design Automation Conf.*, pp. 87–93, June 1994.
- [86] S. Kanjilal, S. T. Chakradhar, and V. D. Agrawal, "A synthesis approach to design for testability," in *Proc. International Test Conf.*, pp. 754–763, Oct. 1993.
- [87] S. Kanjilal, S. T. Chakradhar, and V. D. Agrawal, "Test function embedding algorithms with application to interconnected finite state machines," *IEEE Trans. CAD*, vol. 14, pp. 1115–1127, Sept. 1995.
- [88] S. Kanjilal, S. T. Chakradhar, and V. D. Agrawal, "A partition and resynthesis approach to testable design of large circuits," *IEEE Trans. CAD*, vol. 14, pp. 1268–1276, Oct. 1995.

- [89] S. T. Chakradhar, S. Banerjee, R. K. Roy, and D. K. Pradhan, "Synthesis of initializable asynchronous circuits," *IEEE Transactions on VLSI Systems*, April 1996.
- [90] S. T. Chakradhar, S. Kanjilal, and V. D. Agrawal, "Finite State Machine with Fault Tolerant Test Function," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 562–567, June 1992.
- [91] S. T. Chakradhar, M. Iyer, and V. D. Agrawal, "Energy models for delay testing," *IEEE Trans. CAD*, vol. 14, pp. 728–739, June 1995.
- [92] A. Krstić, K.-T. Cheng, and S. T. Chakradhar, "Primitive delay faults: Identification, testing and design for testability," *IEEE Transactions on Computer-Aided Design*, vol. 18, June 1999.
- [93] M. Lajolo, "Bus guardians: An effective solution for online detection and correction of faults affecting system-on-chip buses," *IEEE Transactions on VLSI Systems*, vol. 9, pp. 974–982, December 2001.
- [94] S. Bhattacharya, F. Brglez, and S. Dey, "Transformations and resynthesis for testability of rt-level control-data path specifications," *IEEE Trans. on VLSI Systems*, vol. vol. 1, September 1993.
- [95] S. Dey, M. Potkonjak, and R. Roy, "Exploiting hardware sharing in high level synthesis for partial scan optimization," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 20 – 25, November 1993.
- [96] M. Potkonjak, S. Dey, and R. Roy, "Considering testability at behavioral level: Use of transformations for partial scan cost minimization under timing and area constraints," *IEEE Transactions on Computer-Aided Design*, vol. 14, pp. 531–546, May 1995.
- [97] M. Potkonjak, S. Dey, and R. Roy, "Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Transactions on Computer-Aided Design*, vol. 14, pp. 1141–1154, September 1995.
- [98] M. Lajolo, L. Lavagno, M. Rebaudengo, M. S. Reorda, and M. Violante, "Behavioral test vector generation for system-on-chip designs," in *Proceedings of the IEEE High Level Design Validation and Test Workshop*, pp. 21–26, November 2000.
- [99] S. Ravi, G. Lakshminarayana, and N. K. Jha, "A framework for testing core-based systems-on-a-chip," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 385–390, 1999.
- [100] S. Wang, "Low hardware overhead scan based 3-weight weighted random bist," in *Proceedings of ITC*, pp. 868–877, 2001.
- [101] S. Wang, "Generation of low power dissipation and high fault coverage patterns for scan-based bist," in *Proceedings of ITC*, pp. 834–843, 2002.
- [102] S. Wang and S. K. Gupta, "DS-LFSR: A BIST TPG for low switching activity," *IEEE Transactions on Computer-Aided Design*, vol. Vol. 21, pp. 842–851, July 2002.
- [103] S. Wang and S. K. Gupta, "An automatic test pattern generator for minimizing switching activity during scan testing activity," *IEEE Transactions on Computer-Aided Design*, vol. Vol. 21, pp. 954–968, Aug. 2002.
- [104] P. Ashar and M. Cheong, "Efficient breadth-first manipulation of binary decision diagrams," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 622–627, 1994.
- [105] M. Lajolo, L. Lavagno, M. Rebaudengo, M. S. Reorda, and M. Violante, "Automatic test bench generation for simulation-based validation," in *Proceedings of the 8<sup>th</sup> IEEE International Workshop on Hardware/Software Codesign*, pp. 136–140, May 2000.

- [106] M. Lajolo, L. Lavagno, M. Rebaudengo, M. S. Reorda, and M. Violante, "System-level test bench generation in a co-design framework," in *Proceedings of the IEEE European Test Workshop 2000*, pp. 69–73, May 2000.
- [107] P. Ashar and S. Malik, "Fast functional simulation using branching programs," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 408–412, 1995.
- [108] A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik, "Partition-based decision heuristics for image computation using SAT and BDDs," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 286–292, 2001.
- [109] S. Cadambi, C. S. Mulpuri, and P. Ashar, "A Fast, Inexpensive and Scalable Hardware Acceleration Technique for Functional Simulation," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 570–575, June 2002.

# THE STRONG MUTUAL IMPACT BETWEEN PHILIPS RESEARCH AND ICCAD

Emile Aarts and Frans Theeuwen

*Philips Research Laboratories Eindhoven  
Eindhoven, The Netherlands*

## **Abstract**

We briefly discuss the influence of the International Conference on Computer-Aided Design (ICCAD) on the research program carried out at Philips Research over the past twenty years. We highlight some of the developments in the areas of simulated annealing, media-processor design, logic synthesis, and formal verification.

## **1. Introduction**

During its 20 years of existence the International Conference on Computer-Aided Design (ICCAD) has provided Philips Research scientists with an international stage to exchange ideas and results with the research community. The conference not only enabled our early access to new results in the field, but it also gave our scientists the opportunity to present and discuss their own work. This two-way flow of information is characteristic of the role ICCAD has played within Philips Research. Over the years, about fifty Philips Research scientists attended ICCAD, and fifteen papers were contributed on a range of subjects, including mixed-signal IC design, simulated annealing, logic synthesis, high-level synthesis, device modeling, and simulation.

The CAD effort within Philips Research over the past twenty years has been substantial. In the late 1970s, the early efforts on circuit simulation, layout, and testing resulted in the establishment of a CAD research department staffed with about twenty research scientists. In the early 1990, the growth of these developments led to the start of a self-financing activity within Philips called Electronic Design and Tools. The role of this department is to develop CAD tools and act as a consulting department for the Semiconductor and Components product divisions within Philips. The introduction of efficient and effective tools for low abstraction level design tasks, such as circuit simulation and layout synthesis, shifted the CAD effort within Philips Research towards higher abstraction levels such as formal verification and high-level synthesis. This development complies with the general trend within the domain. It also has led to the integration of CAD research with research on embedded systems design, diminishing the position of CAD as an independent field of research.

Below, we present four best practices illustrating different aspects of the role ICCAD has played over the past twenty years within Philips Research. The first two discuss contributions from Philips Research to ICCAD; the other two elaborate on the impact of research results presented at ICCAD on CAD developments within Philips.

## 2. Simulated annealing

Almost immediately after Kirkpatrick, Gelatt and Vecchi [6] published their seminal paper on simulated annealing, the CAD community discovered the approach as a very useful tool to handle hard optimization problems in IC design. For many problems in routing, placement, and floorplanning simulated annealing proved to be more effective than existing approaches. Problems in gate-matrix and standard-cell placement called for new effective solution methods that could deal with the rapidly growing instance sizes that resulted from the ongoing miniaturization in semiconductor technology. Despite its effectiveness, simulated annealing suffered from the disadvantage of being very time-consuming, which was inherently due to the probabilistic nature of the approach. Consequently, many researchers took up the challenge to improve the algorithm's efficiency by introducing advanced cooling schedules that could guarantee fast convergence to near-optimal solutions. These efforts have led to many improvements, which were presented at ICCAD in the mid 1980's.

The classical cooling schedule of Kirkpatrick, Gelatt and Vecchi [6] uses a cooling parameter, also called temperature, which controls the probability of accepting newly generated solutions. The lowering of this control parameter during the algorithm's execution has been the subject of investigation for many researchers. Otten and Van Ginneken (see paper on page 479) were the first to propose to replace the simple geometric lowering of the control parameter by an advanced lowering function making use of the concept of "quasi equilibrium" and of concepts borrowed from statistical physics. Aarts and Van Laarhoven [1] used the quasi equilibrium concept to develop a schedule that converged provably in polynomial time. Huang, Romeo and Sangiovanni-Vincentelli [4] proposed a schedule using an exponentially fast reduction of the control parameter. With the introduction of the extremely fast schedule by Lam and Delosme [8] a point in the development of ever faster cooling schedules was reached where the complexity of the calculations needed to estimate new control parameter values could no longer be compensated by the gain in speed of the algorithm's convergence. So, consequently, the interest in cooling schedule research diminished, and ICCAD was no longer the stage where new cooling schedule results were presented.

Simulated annealing has grown mature to become a standard optimization method in many CAD tools. An excellent example is the TimberWolf placement

and routing package that was developed by Sechen and Sangiovanni-Vincentelli [12]. Within Philips several simulated annealing based tools have been used for a long time to cope with problems in IC design. Currently many tools dealing with low abstraction level problems such as routing and placement have been replaced with commercial layout packages, some of which still use simulated annealing. In several cases, where there are no commercial packages available, simulated annealing is often the preferred optimization tools because of its fast and flexible application in non-standard situations.

### 3. Media-processor design

Over the years Philips has invested quite some effort in the development of tools that support the design of Digital Signal Processors (DSPs) for multi-media applications. The effort was based on the belief that true silicon compilation could be achieved in certain application-specific domains such as DSP design. Early work at Philips Research dates back to the design of parametric filters for audio processing applications in the mid 1980s. Typical of the audio applications was the fact that requirements for both flexibility and performance could be easily met in a single design. In video applications this was no longer the case because of the extremely high computational demands imposed by the tight throughput requirements. This introduced the problem of trading off flexibility and performance.

One approach to this problem was the VSP chip with the corresponding tool set introduced by Essink, Aarts, Van Dongen, Van Gerwen, Korst, and Visser [3]. The VSP was an advanced programmable DSP consisting of several VLIW processing units that could run in parallel. The corresponding tool set provided the user with an advanced programming environment that could be used to (re)program the chip. The flexibility of the VSP was certainly a big advantage, but the intricacy of the programming environment and the large number of VSPs that were needed for real-life video applications hampered commercial success. Nevertheless, one can view the VSP as a predecessor of the Philips programmable processor generation that includes the Trimedia and Nexperia chips.

Another way to trade-off between flexibility and performance was to use tools that could substantially reduce the design time of custom ICs for video processing. The Phideo design methodology introduced by Van Meerbergen, Lippens, Verhaegh, and Van der Werf [11], was an example of such an approach developed at Philips Research. The approach used a collection of design tools that translate a high-level functional specification of a video algorithm, expressed in the Phideo Input Format, into VHDL descriptions of a set of processing units, memories, address generators and controllers. Two Phideo elements deserve special attention. Firstly, the approach made use of the mathematical model that expressed the repeated execution of tasks, which is typical for video ap-

plications, in terms of periodic operations. Verhaegh, Lippens, Aarts, Korst, Van der Werf, and Van Meerbergen [13] discuss the problem of finding appropriate periodic schedules for one dimension of repetition. Verhaegh, Aarts, and Van Gorp [14] discuss the corresponding periodic scheduling problem for multi-dimensional periodic operations. Secondly, the approach applied improved optimization techniques to find solutions to the well-known retiming problem introduced by Leiserson and Saxe [10]. Van der Werf, Peek, Aarts, Van Meerbergen, Lippens, and Verhaegh [15] discuss this problem in the general setting of area optimization of multi-functional processing units. The Phideo approach has proved to be a successful design methodology and it was used to design several industrial ICs. The best known example is the Melzonic: an IC for motion-compensated field-rate upconversion by Lippens, de Loore, de Haan, Eeckhout, Huijgen, Loning, McSweeney, Verstraelen, Pham, Kettenis [9] and an IC for MPEG2 video encoding by Kleihorst, van der Werf, Bruls, Verhaegh, Waterlander [7].

#### 4. Logic synthesis

The logic synthesis effort at Philips Research has many roots in research that was first published at ICCAD. In the late 1980s, Philips Research started several research projects on logic synthesis. The aim of these projects was to develop software tools that could automatically translate functional specifications of controllers into circuits consisting of logic gates. Most of these tools applied language transformation techniques that generated PLA tables that were optimized and matched onto a standard cell library. One of the first tools was PHIFACT, introduced by Crowet, Davio, Dierieck, Durieu, Louis, and Ykman-Couvreur [2], which was based on the techniques for multi-level logic synthesis, introduced by Brayton, Detjens, Krishna, Ma, McGeer, Pei, Philips, Rudell, Segal, Wang, Yung, Sangiovanni-Vincentelli (see page 191) and Rudell, Sangiovanni Vincentelli (see page 205).

OMA (Optimizer and MAtcher) was another logic synthesis tool employing concepts similar to those described by Brayton, Detjens, Krishna, Ma, McGeer, Pei, Philips, Rudell, Segal, Wang, Yung, Sangiovanni-Vincentelli (see page 191) and Rudell, Sangiovanni Vincentelli (see page 205). The tool used ELLA as the input language, and can be viewed as the first available ELLA synthesizer. Subsequent tooling efforts led to the development of a Verilog synthesizer called VSyn, and to the start of the development of a VHDL version of a logic synthesis tool, which, however, has not been completed. Several IC's were designed using the logic synthesis tools mentioned above. Especially, in the design of circuits for telecom application the tools have been proved very successful. An example is a chip that could control the digital PRXD telephone switching station.

Currently, these tools are no longer used within Philips. As in the domain of routing and placement they have been replaced by commercially available software packages. This is a general trend in circuit design that holds true for many tools that have been developed within Philips Research. Initially, research groups create implementations of techniques that are of high importance for the company, but as soon as commercial tools become available they are replaced. The advantage of the commercial tools is their improved reliability and maintenance serviced by the vendors. The role of the in-house created tools is however significant because they enable electronics companies to reduce design times of ICs, thus enabling fast introduction of new products.

## 5. Formal verification

Although most techniques used in formal verification date back to the early 1980s, it was only in the early 1990's when the first industrial formal verification tools were introduced Philips. YATC, which was developed in 1993 at Philips Research, was one of the first working tools, and is widely used throughout Philips ever since. YATC as a tool is mainly used for equivalence checking. The objective of equivalence checking is to compare two designs, possibly at different levels of abstraction. Examples of such level comparisons are: RTL-to-RTL, RTL-to-GATE, and GATE-to-GATE. Equivalence checking provides a way to attain full correctness with respect to the functional equivalence of all steps down from the "golden" RTL model, i.e., no bugs are introduced in the synthesis process, during addition of clock trees, scan chain insertion, engineering change orders, etc. The YATC approach was based on techniques described by Berman and Trevillyan (see page 29) and work performed by Rudell described (see page 51), YATC was developed in close cooperation with research groups of the Eindhoven University of Technology, who contributed to the project with their vast knowledge on binary decision diagrams [5]. The YATC tool was built, to a large extent, on the knowledge about formal verification techniques that was publicly available in open literature. However special attention was paid to heuristics that would make the tool more suitable for relevant industrial designs. We believe that this task of tailoring academic knowledge to relevant industrial use is a role that can only be carried out with success by industrial research groups, because they have direct access to real-life circuit designs, whose use is mandatory to judge the techniques on their true merits, but which are often not available to academic research groups for several reasons. Below we discuss two remarkable success stories for formal verification within Philips.

In 1997 circuit designers ran into a serious design problem, which took them several weeks of extensive simulations in the backend to find out what caused the problem. The actual formal verification of the block where the error occurred

took just 15 minutes. It turned out that the logic synthesis tool had swapped two bits deep inside the module.

The second example relates to the chip set of the Philips car navigation system CARIN that was developed by Philips during the late 1990s. YATC was used to verify the controller part of the system, and this was done with great success because the tool found several serious errors. The designers used YATC because they were convinced that automatic formal verification tools could be of great use in obtaining correct designs. In this respect they were the first within the Philips design community to put faith and trust in such automatic tools. Currently, YATC is widely accepted and it has become part of the standard design flow for digital chips within Philips.

## 6. Conclusion

Computer Aided Design is an extremely challenging field for industrial research, and ICCAD has provided a splendid international discussion forum for more than twenty years. We can only hope that the conference will maintain its prime position, and that it will be able to serve this most important role for as long as computer aided design remains a scientific research topic.

## References

- [1] Aarts, E.H.L, and P.J.M. van Laarhoven [1985], A new polynomial-time cooling schedule, Proc. IEEE Int. Conf. on Computer-Aided Design, Santa Clara, 206-208.
- [2] Crowet, F., M. Davio, C. Dierieck, J. Durieu, G. Louis, and C. YkmanS-Couvreur [1990], PHIFACT-a Boolean preprocessor for multi-level logic synthesis, Proceedings of the IEEE Int. Conference on Computer-Aided Design, Santa Clara, 506-509.
- [3] Essink, G., E. Aarts, R. van Dongen, P. van Gerwen, J. Korst, and K. Vissers [1991], Scheduling in programmable video signal processors, Proceedings of the IEEE Int. Conference on Computer-Aided Design, Santa Clara, 284-287.
- [4] Huang, M.D., F. Romeo, and A. Sangiovanni-Vincentelli [1986], An efficient general cooling schedule for simulated annealing, Proceedings of the IEEE International Conference on Computer-Aided Design, 381-384.
- [5] Eijk, C.A.J. van, Janssen, G.L.J.M. [1995], Exploiting structural similarities in a BDD-based verification method, Proceedings TPCD 94, ed. T. Kropf and R. Kumar, vol 901 of lecture notes im Comp.Science, Springer Verlag
- [6] Kirkpatrick S., Gelatt C., Vecchi, [1983] Optimization by simulated annealing, Science 220
- [7] Kleihorst R.P., A van der Werf, W.H.A. Bruls, W.F.J. Verhaegh, E. Waterlander, MPEG2 video encoding in consumer electronics [1997], J. VLSI Signal Processing, vol 17, no 2-3, pp. 241-253.
- [8] Lam, J., and J.-M. Delosme [1986], Logic minimization using simulated annealing, Proc. IEEE Int. Conf. on Computer-Aided Design, Santa Clara, 348-351.
- [9] Lippens. P.E.R., B. de Loore, G. de Haan, P. Eeckhout, H. Huijgen, A. Loning, B. McSweeney, M. Verstraelen, B. Pham, J. Kettenis, [1996], A video signal processor for motion-

- compensated field-rate upconversion in consumer television, IEEE J. Solid-State Circuits, vol 31. pp 1762-1769
- [10] Leiserson, C.E., and J.B. Saxe [1991], Retiming Synchronous Circuitry, Algorithmica 6, 5-35.
  - [11] Meerbergen J.L. van, P.E.R. Lippens, W.F.J. Verhaegh, A. van der Werf [1995], PHIDEO:High-level synthesis for high throughput applications, Journal of VLSI Signal Processing, vol 9, pp. 89-104.
  - [12] Sechen, C., and A.L. Sangiovanni-Vincentelli [1985], The TimberWolf Placement and Routing package, IEEE Journal of Solid State Circuits 20, 510-522.
  - [13] Verhaegh, W.F.J., P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, A. van der Werf, and J.L. van Meerbergen [1992], Efficiency improvements for force-directed scheduling, Proceedings of the IEEE Int. Conference on Computer-Aided Design, Santa Clara, 286-291.
  - [14] Verhaegh, W.F.J. E.H.L. Aarts, and P.C.N. van Gorp [1998], Period assignment in multi-dimensional periodic scheduling, Proceedings of the IEEE Int. Conference on Computer-Aided Design, San Jose, 585-592.
  - [15] Werf, A. van der, M.J.H. Peek, E.H.L. Aarts, J.L. van Meerbergen, P.E.R. Lippens, and W.F.J. Verhaegh [1992], Area optimization of multi-functional processing units, Proceedings of the IEEE Int. Conference on Computer-Aided Design, Santa Clara, 292-299

# CONTRIBUTIONS FROM THE “BEST OF ICCAD” TO SYNOPSYS

Raul Camposano, Ahsan Bootehsaz, Debasish Chowdhury,  
Brent Gregory, Jim Kukula, Narendra Shenoy and Tom Williams

*Synopsys, Inc.,*

*Mountain View, CA, USA*

## Abstract

This paper highlights some examples of contributions from the selected ICCAD papers to Synopsys. We do not attempt to be exhaustive. Given the breadth of the topics addressed at ICCAD and the number of products offered by Synopsys this would be very difficult. We also try not to make any value judgments on the importance of a topic: In terms of (current) economic relevance markets speak for themselves, predicting future relevance accurately is impossible, and judgment on the technical relevance is subjective. So the given examples merely reflect the papers and products the authors are most familiar with. We hope however, that they do illustrate what it takes to create practical design technology (tools) starting from outstanding ideas.

Synopsys creates leading electronic design automation (EDA) tools for the global electronics market. The company delivers advanced design technologies and solutions to developers of complex integrated circuits, electronic systems and systems on a chip. Synopsys also provides consulting and support services to simplify the overall IC design process and accelerate time to market for its customers. Synopsys participates vigorously in all EDA markets addressed by the ICCAD paper selection: analysis, system, logic, circuit, physical, functional verification, timing, and test. Synopsys is among the largest EDA companies with a total market share of over 30%.

Logic Synthesis was Synopsys' first area of focus and continues to be one of our main interests. Our Design Compiler™ and Physical Compiler™ product family compile Verilog and VHDL into optimized, technology mapped and (in the case of Physical Compiler™ ) placed netlists, and they have been widely adopted for design. These products have incorporated many ideas from the ICCAD paper list, e.g. [1, 2, 3, 4, 5, 6]. Boolean logic optimization / minimization techniques such as the ones implemented by Brayton et.al. [1] and Rudell [2] became a central component of Synopsys' logic synthesis tools starting from the first release. These techniques led to automatically generated circuits that could rival those generated by hand, leading to widespread adoption of this productivity improving technology. From the start, improved clock speed and chip area were concerns of every designer using logic synthesis. The global flow techniques described by Berman et. al [7] led to large improvements over what was possible with the earlier generation logic synthesis tools. This helped

broaden the appeal of this new technology to the most performance sensitive designers. Ordered BDDs are extensively used in logic synthesis (as well as many other EDA applications) and the work by Rudell [5] was certainly a milestone in this area. One less known example is the work of Shenoy and Rudell [6] which documented for the first time that retiming could be used on circuits of practical interest. Until then, the concept of retiming was largely of theoretical interest and the impact on the design community was marginal. This work revisited the problem from the viewpoint of determining an efficient implementation (although the theoretical complexity is slightly worse than the original algorithm by Leiserson and Saxe). This effort resulted in the "Behavioral Retiming" concept introduced in Behavioral Synthesis (Behavioral Compiler<sup>TM</sup>) and Logic Synthesis (Design Compiler<sup>TM</sup>). Although designers are usually hesitant to allow tools to change register boundaries, the benefits of this approach (speed and area) have led to its successful application particularly to pipelined data path circuitry where initial state is not a concern.

Over the last five years Static Timing Analysis (STA) has become part of the standard design flow and is used extensively for timing "sign-off" (between the designer and the IC manufacturer). Computing the delay of interconnects is key in STA, and the O'Brien and Savarino's ideas [8] in this area influenced our product PrimeTime<sup>TM</sup>. Another necessary step to reduce the amount of delays dealt with to a practical size is model order reduction, the contributions by Odabasioglu et. al [10] and by Feldman and Freund [11] were certainly very influential in this field. Through the late eighties and early nineties, circuit design with latches became a popular style. It enabled designers to "borrow" computation time from the preceding and following stages during the active period of the latch. However such a design style is hard for timing verification, as paths can sneak through multiple levels of latches. Sakallah et al. provided a nice framework to do analysis of such circuitry. The problem of timing verification at the latch boundaries was resolved to be of polynomial complexity by the work of Szymanski and Shenoy [9]. The major restriction is that all clocks are required to operate at the same frequency. The authors showed that it is important to initialize the departure times at the latches to the earliest launch times at the start of the iteration for correct results. Although the restriction on the clock frequency makes the algorithm less relevant in the industrial environment with multiple clock frequencies, almost all timing analysis algorithms have the iterative scheme and initialization process presented by the authors.

Formal verification appears today in the Synopsys product line principally in the Formality<sup>TM</sup> equivalence checking tool and in the FormalVera RTL verification tool. The BDD technology of Rudell [5], the SAT technology of Marques Silva and Sakallah [12], the detection and exploitation of internal equivalences described in the paper of Berman and Trevillyan [13] and the paper of Brand [14], and the diagnosis method of Madre, Coudert and Billon [15], all of

these are building blocks that a state of the art industrial equivalence checker like Formality must use to meet customer needs. FormalVera is a semi-formal functional verification tool that aims to improve the effectiveness of RTL verification by supporting coverage-driven test generation and formal property verification. Some of the techniques used in FormalVera for checking functional correctness of digital designs with respect to specifications can be traced back to Coudert and Madre [16], along with the fundamental BDD technology of [5].

In addition, Marques Silva and Sakallah's algorithm to solve Satisfiability [12] also influenced FormalVera. Beyond these products with explicit formal verification function, other Synopsys products in the synthesis and verification areas incorporate the same BDD and SAT technology to analyze, solve, and transform Boolean functions.

The VCS™ Verilog simulator provides the performance, capacity and built-in coverage metrics required for verifying multi-million gates SoC designs. In addition, VCS7.0 has introduced native support of assertions, test bench features, observed coverage and a direct, fast link to C applications (DirectC, CycleC). VCS7.0 supports Verilog, VHDL, mixed-HDL (MX) and mixed-signal simulation for complex SoC designs. We see this trend of integrating simulation with formal approaches as one of the key directions in functional verification today, which has been strongly influenced by the formal analysis and assertion/property specification [16]. VERA®, the test bench automation tool from Synopsys, addresses the support and automation of generating simulation vectors. VERA® is based on the non-proprietary, open standard OpenVera™ hardware verification language. VERA® uses BDD techniques [5] and BDD-based constraint solver technology.

The papers by Strojwas et.al. [17] and by Antreich and Graeb [18] provide insight to improve the overall yield of a fabrication process and the overall robustness of the design. In today's fabrication processes variations typically exceed the cases contemplated by these papers, and failure data collection to help diagnostics are increasingly popular. Synopsys' ATPG tool, TetraMAX®, coupled with automatic test equipment or a tester collects such data and offers diagnostics information to isolate the problem areas in failure analysis. TetraMAX® also includes a delay test package, which allows the user to determine what kind of delay path test he or she requires. It uses a robust path delay that covers a pre-specified critical functional paths or a transition delay test which will target all slow to rise and slow to fall delay faults in the entire network. Kundu et.al [19] address delay test, dealing with how to find a delay test for a path that is not invalidated by transient signals caused by arbitrary circuit delays and/or timing skews in input changes and looking at the full coverage of such tests relative to stuck open and stuck on transistors.

Finally, in the area of transistor-level design, we would like to mention two contributions. Bryant's techniques for extraction of gate level models from tran-

sistor circuits [20] certainly influenced our transistor-level static timing analysis tool PathMill™. AMPS™, our tool to optimize circuits by transistor sizing , offers similar functionality than JiffyTune described by Conn et.al. [21].

In summary, we can cite numerous ideas published in the “Best of ICCAD” contributions to commercial EDA tools and our product line in particular. ICCAD’s long tradition includes papers with ideas that were put to work almost immediately; others were published long before their use in commercial products and for some their time has yet to come.

## References

- [1] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Oei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, and A. Sangiovanni-Vincentelli, “Multiple-level logic optimization system,” in *International Conference on Computer Aided Design*, pp. 356–359, 1986.
- [2] R. Rudell and A. Sangiovanni-Vincentelli, “Exact minimization of multiple-valued functions for pla optimization,” in *International Conference on Computer Aided Design*, pp. 352–355, 1986.
- [3] J. Vasudevamurthy and J. Rajski, “A method for concurrent decomposition and factorization of boolean expressions,” in *International Conference on Computer Aided Design*, pp. 510–513, 1990.
- [4] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, “Logic decomposition during technology mapping,” in *International Conference on Computer Aided Design*, 1995.
- [5] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” in *International Conference on Computer Aided Design*, pp. 42–46, 1993.
- [6] N. Shenoy, R. Rudell, “Efficient implementation of retiming,” in *International Conference on Computer Aided Design*, 1994.
- [7] L. Berman and L. Trevillyan, “Improved logic optimization using global flow analysis,” in *International Conference on Computer Aided Design*, pp. 102–105, 1988.
- [8] P. R. O’Brien and T. L. Savarino, “Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation,” in *International Conference on Computer Aided Design*, pp. 512–515, 1989.
- [9] T. G. Szymanski, N. Shenoy, “Verifying Clock Schedules,” in *International Conference on Computer Aided Design*, pp. 124–131, 1992.
- [10] A. Odabasioglu, M. Celik, and L. T. Pileggi, “PRIMA: Passive reduced-order interconnect macromodeling algorithm,” in *International Conference on Computer Aided Design*, pp. 1–9, 1997.
- [11] P. Feldmann and R. W. Freund, “Circuit noise evaluation by pade approximation based model reduction techniques,” in *International Conference on Computer Aided Design*, pp. 132–138, 1997.
- [12] J. P. Silva and K. A. Sakallah, “GRASP - a new search algorithm for satisfiability,” in *International Conference on Computer Aided Design*, pp. 220–227, 1996.
- [13] L. Berman and L. Trevillyan, “Functional comparison of logic designs for vlsi circuits,” in *International Conference on Computer Aided Design*, pp. 456–459, 1989.
- [14] D. Brand, “Verification of large synthesized designs,” in *International Conference on Computer Aided Design*, pp. 534–537, 1993.

- [15] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," in *International Conference on Computer Aided Design*, pp. 30–33, 1989.
- [16] O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits," in *International Conference on Computer Aided Design*, pp. 126–129, 1990.
- [17] A. J. Strojwas, S. R. Nassif, and S. W. Director, "A methodology for worst case design of integrated circuits," in *International Conference on Computer Aided Design*, pp. 152–153, 1983.
- [18] K. J. Antreich, H. E. Graeb, "Circuit optimization driven by worst-case distances," in *International Conference on Computer Aided Design*, pp. 166–169, 1991.
- [19] S. Kundu, S. M. Reddy, N. K. Jha, "On the design of robust multiple fault testable CMOS combinational logic circuits," in *International Conference on Computer Aided Design*, pp. 240–243, 1988.
- [20] R. E. Bryant, "Extraction of gate level models from transistor circuits by four-valued symbolic analysis," in *International Conference on Computer Aided Design*, pp. 350–353, 1991.
- [21] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, and C. Visweswariah, "Optimization of custom MOS circuits by transistor sizing," in *International Conference on Computer Aided Design*, 1996.

# ICCAD AND XILINX

Rajeev Jayaraman

*Xilinx Inc.*

*San Jose, CA, USA*

## Abstract

This paper examines the unique impact of Computer-Aided Design on Xilinx and on Field Programmable Gate Arrays. A brief history and description of FPGA implementation software is given. Some specific CAD algorithms that have influenced FPGA implementation software are listed along with their contributions.

## 1. Introduction

Field Programmable Gate Arrays (FPGAs) have revolutionized digital design and are popular because of their programmability, fast time-to-market, and low fixed costs. Founded in 1984 to pioneer a revolutionary new technology, the field programmable gate array (FPGA), Xilinx fulfills more than half the world's demand for FPGAs. Today, Xilinx develops, manufactures, and markets a broad line of advanced FPGAs and software design tools.

As developers of advanced custom ICs, Xilinx employs a variety of advanced CAD tools for the design of FPGA chips. As a developer of software design tools for FPGAs, Xilinx also provides advanced CAD tools for users to implement their designs on the programmable fabric of FPGAs. Xilinx is, therefore, both a consumer and a developer of advanced CAD tools. CAD is, therefore, a very critical factor in the success of Xilinx and FPGAs. Innovation has always been at the centerpiece of Xilinx' culture and we at Xilinx congratulate ICCAD on its 20th anniversary of being the premier conference bringing forth fresh innovative ideas in CAD.

For the first few generations, FPGAs were developed on mature process technologies that lagged the leading edge technologies in performance and area, but had better yields. In the past few years, however, FPGA chips are being implemented in leading process technologies, and are increasingly being used as process drivers for these leading process technologies. FPGAs are being used as process drivers for leading process technologies for several reasons: they are built on mainstream CMOS technology, they are fabricated in large volumes, and they are highly regular and densely laid out chips. Fabricating FPGAs in leading edge technologies requires the use of cutting-edge EDA tools. Several of these cutting edge EDA tools have had their origins in ICCAD papers.

Apart from using EDA tools to design and develop FPGAs, Xilinx also develops FPGA implementation software that lets users design their systems and program the FPGA accordingly. In this paper, we will highlight different CAD papers and techniques that have played a significant part in the development of FPGA implementation software and contributed to the success of Xilinx and FPGAs.

## 2. Impact of CAD on FPGA implementation software

When Xilinx invented FPGAs, there were no EDA tools available for users to implement their designs on FPGAs. Users had to manually implement their design on an FPGA. They would bring up a graphical tool that displayed a model of the FPGA device. They would then proceed to configure specific logic elements and routing resources to implement their design. After configuring different parts of the FPGAs, users would then direct the tool to automatically generate the bitstream necessary to program the device. Even for the relatively small gate densities of FPGAs available at that time, this manual method was exceedingly error prone and tedious.

Automation came in the form of CAD tools that allowed users to enter their design on a standard schematic editor. The netlist generated by the schematic editor was processed by automatic tools that placed and routed these designs, and also generated the corresponding bitstreams. In the beginning, these tools were developed by Xilinx because the FPGA market was small compared to the size of the ASIC market and, consequently, there were no other EDA vendors developing FPGA-specific CAD tools.

Since the first FPGA-CAD tools were originally developed by Xilinx and other FPGA vendors instead of ASIC-EDA vendors, they have evolved differently from the ASIC tools. Consequently, the FPGA implementation flows have also evolved differently from the ASIC flows. As FPGAs became more popular, the implementation flows required to implement user designs became more sophisticated and started to resemble ASIC flows. Consequently, external EDA vendors started supplying some CAD tools for FPGAs. Typically, these tools were small FPGA-related modifications to the tools that they already sold to the ASIC market. Today, however, there are several EDA vendors that supply CAD tools that are designed exclusively for FPGAs.

The history of CAD tools for FPGAs is important to understand the evolution of FPGA-CAD algorithms. Initially, FPGA-CAD algorithms started out by exclusively adapting standard ASIC CAD algorithms for use in FPGAs. However, over time algorithms were developed that were exclusive to FPGAs. There is now a thriving research effort focused exclusively on FPGA-related CAD tools. Interestingly, in this transition from adapting standard ASIC-CAD algorithms to

developing FPGA-specific CAD algorithms, the FPGA-CAD community has developed several CAD algorithms that have not only found applications in ASIC-CAD but have become seminal works for the general CAD community.

It is important to understand the FPGA implementation process before we can illustrate how specific algorithms have influenced FPGAs. A typical FPGA design flow is shown in Figure 2. The FPGA implementation flow can be divided into 3 main phases: design entry, design implementation, and design verification. The design entry phase is identical to the ASIC design flow and consists of design entry tools such as HDL, schematics etc. The design verification phase is also similar to the ASIC flow and consists of verification tools such as simulation tools. Since they are similar to standard ASIC tools, we will not discuss these tools in this paper. This paper discusses only the design implementation phase of the FPGA design flow.

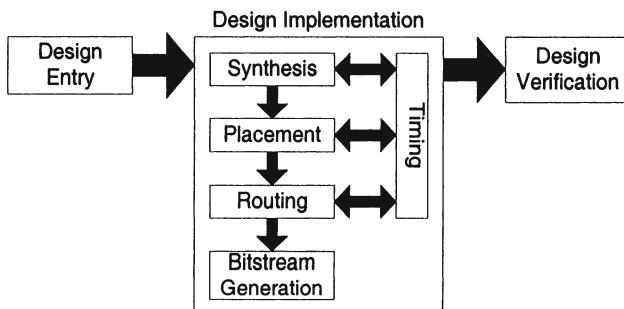


Figure 1. FPGA Design Flow

The FPGA design implementation phase can be divided into synthesis, placement, and routing, followed by bitstream generation. In the logic synthesis phase, an input HDL description is synthesized and mapped into logic elements such as Lookup tables, Flip-flops, IOs, Multiplexors etc. that are the basic building blocks of the target FPGA architecture. The resulting netlist consists of these logic elements connected together to implement the user design, and is used as input to the placement phase consisting of placing these elements on the FPGA sites that implement these logic elements. The placement of the logic elements in the design netlist on the logic element sites on the FPGA dictates the configuration of those sites. After all the logic elements are placed appropriately on sites on the FPGAs, they are connected together in the routing phase. Routing determines how the routing fabric must be configured to achieve the connections that implement the design. Together, the configuration of the logic sites and the routing fabric constitutes the bitstream for the FPGA that can be loaded on the device to implement the user design on the FPGA.

## 2.1 Synthesis and Technology Mapping

Initially, the synthesis and technology mapping for FPGAs was heavily influenced by the Espresso paper by Rudell et al. on page 205 and the MIS paper by Brayton et al. on page 191. One of the earliest algorithms for FPGA specific technology mapping was presented by Francis et al. in [4]. The objective of their work was to perform technology mapping in order to minimize the number of LUTs (area minimization). Their main contribution was the combination of decomposition and covering LUT FPGA technology mapping. They enhanced the algorithm to do delay minimization (reduce the number of logic levels) in the Chortle-d algorithm presented ICCAD 1991 [5]. The FLOW-MAP algorithm by Cong et al. on page 235 optimally solved the LUT-based FPGA technology mapping problem for depth minimization for general Boolean networks. This was a seminal result because it presented a polynomial time algorithm for solving the FPGA-specific (LUT-based) technology mapping problem compared to the general technology mapping problem which is NP-hard.

## 2.2 Placement

The placement problem for FPGAs, as mentioned earlier, consists of placing logic elements onto the configurable logic sites on the FPGAs. The placement problem, therefore, is very similar to the placement problem for ASICs. Consequently, the standard placement algorithms such as min-cut based approaches [7], annealing-based approaches [8] such as TimberWolf [11], and quadratic placement based approaches such as GORDIAN on page 499 and Ritual [12] were employed for FPGA placement. However, due to the complex constraints inherent in the FPGA placement problem, annealing-based approaches were the most popular algorithms for FPGA placement. One of the earliest published works on placement for FPGAs was Betz et al. in [1]. They developed a tool called VPR that did logic element packing, placement, and routing for FPGAs. VPR was used not only as a FPGA-CAD tool but also as a tool that could evaluate FPGA architectures.

## 2.3 Routing

While the FPGA placement problem is very similar to the ASIC placement problem, FPGA routing is very different from ASIC routing. Initial FPGA routing approaches were adaptations of basic maze routing methods [9]. The routing model of an FPGA is represented as a connectivity graph where the nodes of the graph are the routing segments while the edges are the programmable interconnection points. The FPGA routing problem is essentially one of embedding the netlist on the underlying router graph. This routing model is the main reason why FPGA routing differs from ASIC routing: in ASIC routing, a route is ex-

pressed in terms of the underlying rectilinear grid, but in an FPGA, a route is expressed in terms of a path in the underlying routing graph.

Brown et al. [2] developed one of the first routers for FPGAs. The PATHFINDER algorithm by Ebeling et al. [3] presented a novel method for FPGA routing that has since become the most popular FPGA routing algorithm. In this algorithm, nets are routed sequentially under the assumption that all resources are available to every net. After the first iteration, a cost is imposed on the resource corresponding to the demand of nets on the resource. With this new cost, another iteration is started. The costs of the resources are monotonically increased based on the demand for them. This algorithm is especially suited for FPGAs since it can take a global view of the finite routing resources on the FPGA.

Recently, there have been several novel approaches to FPGA routing based on the GRASP algorithm by Silva et al. on page 73. Gi-Joon et al. in [10] formulates the FPGA routing problem as a Boolean Satisfiability Problem and uses the GRASP SAT solver to solve the routing problem.

## 2.4 Contributions of FPGAs to CAD

Most of the work on FPGA CAD has had its roots in ASIC CAD. However, there have been instances where research spawned by work on FPGAs has made significant contribution towards ASIC-CAD. One such example is the work by Frankle on iterative and adaptive timing slack allocation [13] where he proposed a generalization of the limit-bumping algorithm and then showed how lower and upper bounds on connection delays could be used in performance-driven layout improvement. This work was initially done for FPGA routing but is now a highly referenced paper for timing slack allocation in timing-driven placement and routing for both ASICs and FPGAs.

## 3. New Architecture Definition

One of the unique ways in which FPGAs depend on CAD is in the area of FPGA architecture development. New FPGA architectures must be defined and evaluated for cost, routability, and performance. This evaluation is done, typically, with the help of CAD tools. In a typical evaluation of a new FPGA architecture, the different parameters of the FPGA (logic elements, routing resources etc.) are set. Using CAD tools modified to support the new architecture, designs are implemented on the FPGA and analyzed. Based on this analysis, the architecture parameters are modified and optimized to get the best performance on a set of designs on which the architecture is evaluated.

Betz et al. [1] have done pioneering work in this area. With the help of CAD tools, the functionality of the logic elements, the size and area of the logic

elements, and the routing network between the logic elements can be modified and evaluated for cost, performance, and density.

#### 4. Future role of CAD

FPGAs are growing increasingly dense and complex. FPGAs today no longer consist of simply gates and routing. Instead, they have several other system level features such as processors, gigabit transceivers etc. implemented on them. This increasing complexity puts a tremendous onus on CAD tools to efficiently implement user designs onto these large and complex FPGAs.

The performance of the implemented designs and the runtime of the FPGA-CAD tools are two of the cornerstones for the success of FPGAs. Designs implemented on FPGAs can have an inherent performance disadvantage over those implemented on ASICs. Consequently, it is critical for the success of FPGAs to narrow this performance gap. Further, since one of the primary reasons FPGAs are becoming popular is their faster time-to-market, it is also important for FPGA implementation tools to reduce their execution times to allow for faster time-to-solution while operating on increasingly large designs. These aggressive performance and runtime objectives require significant advances in CAD algorithms.

A new area in which FPGAs have become very popular is in the area of reconfigurability. Due to their inherent reconfigurability, FPGAs are ideal vehicles for implementing reconfigurable systems. CAD for reconfigurable systems is a relatively new area that will drive the acceptance of reconfigurable computing into the mainstream.

In the past 20 years, ICCAD has been a fertile publishing avenue for research in CAD. We believe that revolutionary CAD ideas and techniques that will originate from ICCAD in the future will provide the breakthroughs in these areas and help propel FPGAs and Xilinx to greater success.

### References

- [1] V. Betz and J. Rose, *VPR: A New Packing, Placement and Routing Tool for FPGA Research*, Proceedings of the 7th Intl. Workshop on Field-Programmable Logic, London, 1997, pp 213-222
- [2] S. Brown, J.S. Rose, Z. Vranesic, *A Detailed Router for Field Programmable Gate Arrays*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, vol. 11, 1992, pp 620-628
- [3] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, *Placement and Routing Tools for the Triptych FPGA*, IEEE Transactions on VLSI, Dec. 1995, pp 473-482
- [4] R. J. Francis, J. Rose, and K. Chung, *Chorlde: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays*, Proceedings of the 27th ACM/IEEE Design Automation Conference, 1990, pp 613-619

- [5] R. J. Francis, J. Rose, and Z. Vranesic, *Technology Mapping of Lookup Table-Based FPGAs for Performance*, Proceedings of the ICCAD, 1991, pp 568-571
- [6] R. Jayaraman, *Physical Design for FPGAs*, Proceedings of the Intl. Symposium on Physical Design, 2001, pp 214-221
- [7] B. W. Kernighan and S. Lin, *An Efficient Heuristic Procedure for Partitioning Graphs* , The Bell System Technical Journal, vol. 49, 1970, pp 291-307
- [8] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, vol. 220, 1983, pp 671-680
- [9] C. Y. Lee, *An Algorithm for Path Connections and its Applications*, IRE Transactions on Electronic Computers, vol. 5, 1961 pp 408-421
- [10] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, *A Comparative Study of two Boolean Formulations of FPGA Detailed Routing Constraints* , Proceedings of the Intl. Symposium on Physical Design, 2001, pp 222-227,
- [11] C. Sechen, *Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing*, Proceedings of the 25th ACM/IEEE Design Automation Conference, 1988, pp 73-80,
- [12] A. Srinivasan, K. Chaudhary and E. S. Kuh, *RITUAL: Performance Driven Placement Algorithm for Small Cell ICs*, Proceedings of the ICCAD, 1991, pp 48-51
- [13] J. Frankle, *Iterative and Adaptive Slack Allocation for Performance-driven Layout and FPGA routing*, Proceedings of the 29th ACM/IEEE Design Automation Conference, 1992, pp 536-542,
- [14] <http://www.xilinx.com>

# Index

- Acceptance region, 586
- Adder
  - balanced paths in, 121
- Adjoint sensitivity, 347
- Admittance matrix, 435–436
- ADS, 374
- Algebraic decomposition, 183, 260
  - MIS, 197
- Algebraic methods, 181, 183
- Algorithm
  - Arnoldi, 436
  - Bellman-Ford, 603, 620
  - branch and bound, 113, 211, 277
  - Dijkstra, 624, 656
  - GMRES, 406
  - GORDIAN, 501
  - knowledge-based method, 276
  - max-flow min-cut, 241, 468, 521
  - min-cut, 31
  - MPVL, 434, 460
  - NP completeness, 29, 125, 273, 526
  - optimal, 235, 526
  - PVL, 455
  - simulated annealing, 125, 277, 299, 479–480, 545, 636, 648, 676
- Allocation
  - memory, 107
  - resource, 108
- Amplifier, 270, 413
  - noninverting, 390
  - operational, 275, 313, 325, 460
  - traveling wave, 390
- Analysis
  - artwork, 489
  - connectivity, 490
  - corner-case, 557
  - extreme-case, 557
  - GOALIE, 473, 489, 648
  - interconnect, 368
  - layout, 489
  - region, 490
  - static timing, 296, 348, 567, 647, 655, 662, 684
  - switch-level, 279
  - worst-case, 557
- Application Specific Instruction Set Processor, 159
  - See also* ASIP
- ASIP, 100, 159
- AS/X, 352–353, 360
- ATOMICS, 107
- ATPG, 9, 65–66, 181, 555, 640, 642, 665, 667, 685
  - delay fault, 556
  - SAT, 665
  - sequential, 556
- Augmented Lagrangian, 347, 349, 356
- Automatic Test Pattern Generation, 642
  - See also* ATPG
- Backtracking, 10, 73
  - non-chronological, 73, 667
- BDD, 6, 65, 182, 184, 640, 655, 667, 684
  - efficient implementation, 52
  - formal verification, 18, 637
  - ordered, 6, 52
  - package, 182
  - sifting, 57
  - variable ordering, 7, 637
    - dynamic, 7, 53, 655
  - variable swap, 55
  - window permutation algorithm, 56
- BDSYN, 192
- Binary Decision Diagram, 6
  - See also* BDD
- Binding, 162
- Bit-width reduction, 122
- Boolean constraint propagation, 78
- Boolean logic, 683
  - equation solving, 24, 637
  - quantified formula, 23
- Boolean minimization, 207
- Boolean network, 185, 227, 251
  - AND2/INV, 251
    - associative transformation, 257
    - distributive transformation, 258
    - inverter transformation, 257
  - K*-bounded, 236
  - labeling of, 239
  - mapped, 252
- Boolean relation, 185
- Boolean satisfiability, 73

- See also* SAT
- BooleDozer, 181
- Capacitance
  - estimation, 123
  - extraction, 491
  - interconnect, 122–124, 634
  - MOSFET, 297
  - reduction, 121
  - registers, 123
  - switched, 119, 123
- CATHEDRAL II, 96, 107
- Cauchy point, 355
- Charge conservation, 303
- CHESS, 100
- Choice node, 252
- CINDERELLA, 100
- Circuit, 269
  - analog, 275
  - analog/digital, 313
  - analytic models, 313
  - asynchronous, 147, 182
  - channel-connected component, 341
  - combinational, 553
  - cyclic, 186
  - delay-insensitive, 147
  - equations, 303
  - high-Q, 383
  - mesh analysis, 406
  - op amp, 277, 313, 325
  - optimization, 313, 347
  - quasi-delay-insensitive, 147
  - RF, 276, 373
  - simulation, 288, 303, 313, 347, 383
    - RF, 383, 635
    - symbolic, 18
  - sizing, 347, 587, 635, 686
  - synchronous, 552
    - MOS, 295
    - synthesis, 313
    - topology, 270, 276, 313, 501
- Clearance checking, 493
- Clique partitioning, 114
- Clock
  - multi-phase clocking, 597
  - planning, 472
  - schedule, 552
    - optimization, 598
    - verification, 598
  - skew, 473, 509, 636, 641
    - minimal, 655
    - routing, 515
    - useful, 473, 655
    - zero, 473, 515, 636, 648
  - tree, 472, 510, 636, 648, 655, 679
- CoCo Code compression architecture, 664
- Combinational gate, 296
- Communication architecture tuners, 663
- Conflict diagnosis, 10
  - conflict-induced clause, 81
  - non-chronological backtracking, 78
- Conjugate gradient method, 357
- Constraint
  - cluster capacity, 160
  - face, 185
  - interconnect capacity, 161
  - partitioning, 502
  - propagation, 78
- Convex function, 348
- Convex program, 273
  - transistor sizing as, 299, 635
- Convolution, 352–354, 362
- Critical path, 551
  - estimation of, 124
  - impact on power, 120
- Crossing time, 350
- Current
  - bias, 277
  - piecewise approximate, 304
- Custom design, 347
- Cut
  - fundamental cutset, 306, 310
  - height of, 237
  - in a flow network, 242
  - K*-feasible, 237
  - transformation of, 240
  - volume of, 237
- CYBER synthesis system, 664
- Cycle
  - zero weight, 604
- Datapaths, 684
  - clustered, 162
  - parameterization of, 162
- Data precedence, 108
- Decomposition, 227
  - algebraic, 183–184, 260
  - AND2/INV, 251
  - Cholesky, 589
  - gate, 237
  - Roth-Karp, 237
  - using Huffman coding, 238
- Deep Sub-Micron, 143
  - See also* DSM
- Defects, 557
- Delay
  - circuit path, 299
  - combinational block, 299
  - complex gate, 298
  - constant, 654
  - Elmore, 297, 369, 510
  - interconnect, 393
  - optimization, 235
  - optimum, 184
  - testing, 182, 556, 665, 685
  - zero/unit model, 340

- Δ-Mapping, 258
- Design
  - centering, 591
  - for testability, 554
  - See also* DFT
  - hierarchical, 472
  - latency insensitive, 144
  - nominal, 587
  - platform-based, 99
  - programmable, 689
  - re-use, 347–348, 362
  - RTL, 664
  - rule checking, 493
  - space exploration, 161
  - worst-case, 563
- Device
  - characterization, 285
  - measurement, 287
  - modeling, 285, 303
  - sizing, 276
- DFT, 554, 642, 665
- Digital Signal Processor, 677
  - See also* DSP
- Direct sensitivity, 347
- Divisor
  - double-cube, 227
  - kernel, 184–185
  - multiple-cube, 228–229
  - single-cube, 228–229
  - two-cube, 183
- Don't care, 182
  - observability, 185
- DRC, 493, 661
- Driving-point admittance, 396, 634
- DSM, 143, 146
- DSP, 95–96, 101, 107, 130, 649, 677
- Dynamic logic, 186
- Dynamic programming, 184
- Dynamic simulation, 348
- Dynamic tuning, 348
- Elmore delay, 297, 369, 510
- Embedded system, 130, 649, 675
  - application, 98, 159
  - memory, 95
  - optimization, 99
  - power analysis, 99, 129
  - power optimization, 98, 141
  - software, 98
- Enclosure checking, 493
- Equivalence checking, 3, 5, 29, 65, 637, 645, 667, 684
- ESPRESSO, 182, 205
- ESPRESSO Exact, 208
- Estimation
  - capacitance, 123
  - critical path, 124
  - power, 122
- supply voltage, 124
- Event-driven simulation, 303, 305
- Extraction
  - gate-level model, 279
  - MIS, 195
  - model, 270
- Factoring, 233
  - MIS, 197
- False negative, 5, 11, 30
- False path, 199, 348
  - analysis, 552
- FastCap, 367
- FastHenry, 367, 403, 407, 635
- Fault
  - path delay, 556
  - stuck-at, 568
  - transition, 556
  - untestable, 568
- FLEXWARE, 100
- Floorplanning, 470, 648, 676
  - definition, 479
  - representations, 480, 483
- Formal verification, 636, 654, 679, 684
  - BDD, 18
  - diagnosis, 8, 17
  - equivalence checking, 3, 5, 17, 29, 645, 667, 679, 684
  - model checking, 4, 640, 646
  - rectification, 7, 17
- FPGA, 181, 184, 186
  - architecture, 693
  - CAD, 689
  - lookup-table based (LUT), 235
  - placement, 692
  - routing, 692
  - synthesis, 692
  - technology mapping, 235
  - Xilinx, 689
- Frequency domain, 452
- Frequency range, 455
- FSM, 185
  - equivalence, 8
  - power up, 186
- Function
  - convex, 296
  - flexibility, 186
  - multiple-valued, 206
  - multiple-valued input binary-valued output, 205
- Fundamental cutset, 306, 310
- Fundamental loop, 305–306, 310
- Garbage collection, 343
- Gate array, 181, 469
- Gate
  - combinational, 296
  - complex, 237, 296
  - simple, 237
- Geometric operations, 490

- Global flow, 183, 217–218, 647, 683
- GMRES, 406
- GOALIE, 473, 489, 648
- Gordian, 469, 636, 647, 666
  - complexity, 504
  - results, 504
- Gradient
  - based optimization, 348
  - conjugate, 357
  - projected, 355
  - time-domain, 347
- Graph-mapping, 252, 254
- GRASP, 9, 73–74, 634, 646, 667, 693
- Group partial separability, 362
- Hardware-software co-design, 98
- Harmonica, 374, 388
- Harmonic balance, 384
- Hazard
  - dynamic, 121
- High-level design, 664
- HYPER, 97
- HYPER-LP, 117
- IC-CAP, 272
- Inductance
  - extraction, 403, 635
  - partial, 404
- Input patterns, 65, 348
- Instruction level parallelism, 159
- Interconnect
  - analysis, 368, 403, 634
  - delay, 393, 634
  - RLC, 433
- IRSIM, 123, 278
- Iterative solver, 406
- JiffyTune, 271, 347
- Kernel, 228
- Kirchhoff
  - current law, 405
  - voltage law, 406
- KISS, 185
- Krylov methods, 368, 372, 374, 406
- Lagrange multiplier, 356
- Lagrangian
  - augmented, 347, 349, 356
- $\Lambda$ -Mapping, 257
- LANCELOT, 347, 351
- Latch, 296
  - level-sensitive, 597
- Levenberg-Marquardt optimization, 351
- Library, 252
- Limit cycle, 306
- Linear system
  - solution, 357
- Logical effort, 654
- Logic synthesis, 65, 655, 666, 678, 683
  - for testability, 555
  - technology mapping, 344
- Loop
  - fundamental, 305–306, 310
- LSS, 183
- Macromodels, 433
- Manufacturability, 362, 556
- Mapping graph, 184, 252
  - reduced, 253
- Markov chain, 186
- Match, 254
  - covered by, 255
- Matching, 254
- Matrix polynomial, 456
- Maximal independent set heuristic, 212
- McBoole, 205
- Media processor, 677
  - programmable media chip, 677
- Meet-in-the-middle, 107
- Microcode compiler, 107
- Microprocessor
  - MP98, 664
  - PDLX, 154
  - power models, 130
- Microwave simulation, 383, 635
- MIMOLA, 94
- Minimax optimization, 347, 350, 358
- Minimization
  - MINI, 181
  - PLA, 181
  - Quine-McCluskey, 182
  - two-level, 182
- Minmax formulation, 277
- Minos, 351
- MIS, 184, 191, 640
  - algebraic decomposition, 197
  - extraction, 195
  - factoring, 197
  - phase assignment, 197
  - resubstitution, 196
  - simplification, 198
  - timing optimization, 199
- Miter, 11, 66
- Mixed integer continuous problem, 362
- Mixed-signal systems, 275
- MMIC, 373, 383
- MNA, 435
- Model checking, 4, 8, 637, 640, 646
  - bounded, 10
  - explicit, 8
  - symbolic, 8
- Model
  - closed form, 455
  - extraction, 270
  - fitting, 270–272
  - $\pi$ , 370–371
  - reduced-order, 456
- Modified Nodal Analysis, 435
- Moments, 441

- Monolithic Microwave Integrated Circuit, 383  
*See also* MMIC
- MOSES security processing architecture, 664
- MOSFET, 352  
model, 297
- MOTIS, 277
- Multi-frontal methods, 357
- Multiple fault  
path delay, 577, 582  
stuck-at, 579–581  
stuck-open, 579  
testable design, 582
- Multiplication  
strength reduction, 122
- Multiplier  
Lagrange, 300
- Multiport, 433
- Netlist  
as incidence structure, 480  
connectivity matrix, 502  
topology matrix, 501
- Network, 185  
Boolean, 185  
depth, 184  
pruning, 343
- Network flow  
augmenting path, 241  
max-flow min-cut theorem, 241
- NMOS, 565
- Noise, 413, 452  
analysis, 413, 453  
covariance matrix, 425  
flicker, 417, 453  
linear time-varying circuits, 413  
models, 416  
nonlinear circuits, 413  
numerical, 357  
shot, 417, 452  
simulation, 413  
spectral density, 452  
spectrum, 455  
thermal, 417, 452  
white, 452
- OASYS, 271
- Objective function  
quadratic, 502
- OPASYN, 270
- Optimization  
circuit, 347  
delay, 235  
for power, 118  
global, 503  
gradient-based, 348  
group partial separability, 362  
Levenberg-Marquardt, 351  
minimax, 347, 350, 358  
Minos, 351
- numerical methods, 277  
semi-infinite, 362  
sequential, 185  
trust region, 354
- Optimum  
global, 273, 299  
local, 273, 299
- Ordering  
topological, 341
- Orthogonalization of concerns, 145
- Oscillation, 306  
prevention, 306
- Padé approximation, 455
- Padé expansion, 372
- Parameter  
extraction, 285, 288  
optimization, 288
- Parasitic extraction, 403, 491
- Partial match, 262
- Partitioning  
exhaustive slicing optimization, 503  
hardware-software, 99  
min-cut, 521  
network-flow based, 521  
recursive, 503  
shape functions, 504
- Passivity, 373, 435  
proof, 439
- Path  
blocked, 568  
critical, 300  
false, 568
- Pattern, 254  
distributed, 259  
D-pattern, 259  
factored, 259  
F-pattern, 259  
matching, 279
- Performance specification, 586
- PERT, 551
- Phase assignment  
MIS, 197
- PHIDEO, 97
- Physical design, 276, 467, 666
- Piecewise approximation, 304
- Pi-model, 513  
equivalent, 513  
load, 397, 634
- PLA, 205
- Placement, 655  
incremental, 475  
loose and stable removal, 641
- Positional Cube Notation, 208
- Posynomial, 270, 299, 348
- Power, 649  
analysis, 649, 656  
embedded system, 129

- software, 130
- consumption, 118
  - CMOS, 118
  - reducing, 118
    - voltage dependent, 119
  - estimation, 99, 122, 186
    - software, 139
  - measurement, 131
  - optimization, 99, 118, 637, 649
    - embedded system, 141, 665
    - software, 141
  - reduction, 118
- Precedence graph, 109
- Preconditioner
  - block diagonal, 407
  - Schnabel-Eskow, 357
- PRIMA, 373, 438, 635
- Prime implicant, 207
- Process
  - disturbances, 563
  - Lanczos, 459
  - patient, 149
  - stochastic, 452
- Program
  - convex, 299
  - geometric, 274, 277, 299
  - posynomial, 270, 299
- Projected gradient, 355
- Property verification, 637, 684
- Protocol
  - latency insensitive, 149
- PROUD, 469
- PVL, 437
  - algorithm, 455
  - reduction, 456
- Pyramid shaped nand gates, 300
- Quadratic optimization
  - constrained, 502
- Quadratic placement
  - bipartitioning, 499
- Rectangle covering, 185
- Rectangle packing problem, 535
  - sequence-pair, 540
- Recursive learning, 183
- Reduced order model, 635
- Redundancy, 181
  - addition and removal, 183
- Region-oriented scanline, 491
- Register file
  - centralized, 160
  - distributed, 160
- Relay station, 144, 149, 151
- Reset sequence, 186
- Resistance
  - skin effect, 409
  - source-to-drain, 297
- Resistive shielding, 401, 634
- Resubstitution
  - MIS, 196
- Resynthesis, 185
- Retiming, 553, 597, 615, 678, 684
  - area-delay curve, 627
  - Bellman-Ford algorithm, 620
  - constrained min-area, 615
    - period edge pruning, 624
  - min-area, 615
    - minimum cost circulation, 627
  - min-period, 615
    - predecessor heuristic, 621
    - relaxation, 619
  - W and D matrices, 623
- RF
  - circuit, 276, 373
  - design, 93, 276, 367, 649
  - simulation, 373, 383, 635
- RICE, 442
- RLC interconnect, 433
- Routing, 472, 655
  - incremental, 475
  - touch and cross, 641
- SAT, 5, 9, 73, 182, 637, 640, 667
- Satisfiability, 73, 556
  - See also* SAT
- Scheduling, 107, 162
- Schnabel-Eskow preconditioner, 357
- Segment tree structure, 494
- Semi-infinite optimization, 362
- Sensitivity, 347
  - adjoint, 347, 352
  - calculation, 300
  - direct, 347, 352
- Sequence-pair, 540
- Shannon's expansion, 577, 582
- Shell, 144, 152
- Shooting method, 375, 384
- Signal processing
  - power reduction in, 125
- Silicon compiler, 95, 107, 677
- Simplification
  - MIS, 198
- Simulated annealing, 125, 277, 299, 479, 545, 636, 648, 676
  - acceptance probability, 480, 485
  - algorithm, 480
  - entropy, 482
  - finalization, 486
  - initialization, 485
  - moves, 480
  - schedule, 485
  - score function, 481
  - selection probability, 480, 484
  - state space, 480
- Simulation, 313, 685
  - circuit, 303, 347, 383

- dynamic, 348
- event-driven, 303, 305
- fast, 278
- fault, 69
- frequency-domain, 383
- hardware accelerator, 337
- hardware-software co-simulation, 99
- microwave, 383
- RF, 383
- switch-level, 337
- symbolic, 4, 18
- system-level, 455
- ternary, 186
- SIS, 184
- Sizing, 185
- Skin effect, 409
- Slew-rate, 277
- SMO formulation, 597
- SoC, 93, 100, 275
  - test, 555
- Software
  - power analysis, 130
  - power and energy, 131
  - power estimation, 139
  - power optimization, 141
- Solution space
  - P-admissible, 536
  - rectangle packing problem, 536
- Solving linear systems, 357
- Specification
  - performance, 586
- SPECS2, 270
- SPECS, 303, 347, 352
- Spectre, 374, 388
- SpectreRF, 374, 635
- SPFD, 186
- Sphere of influence, 306
- SPICE, 123, 276, 288, 307, 313, 348, 374–375, 383, 413, 443, 451, 556, 565, 660
- State assignment, 185
- Static timing analysis, 299, 348
- Steepest descent, 277
- Stochastic differential equations, 419
- Subnetwork
  - series-connected, 296
- Substitution
  - multiplication to addition, 122
- Symbolic analysis
  - switch-level, 338
- Symbolic model checking, 8
- Synthesis
  - analog, 635
  - asynchronous circuit, 182
  - FPGA, 692
  - interface, 99
  - logic, 637
  - multi-valued circuit, 182
- op amp, 313
- technology independent, 181
- System-level Design, 663
- System on a Chip, 100
  - See also* SoC
- Table models, 303
- Tagged signal model, 149
- TECAP2, 270
- Technology mapping, 181, 184, 255
- Tellegen's theorem, 352
- Temporal logic, 4
- Test, 554, 665
  - built-in self, 554
  - compaction, 665
  - defect based, 556
  - IDDQ, 556
  - invalidation, 576
  - resource partitioning, 554
  - scan chain, 555
  - SoC, 555
  - stuck-at fault, 65, 568
  - synthesis, 554
  - untestable fault, 65, 568
- Testability
  - partial scan, 665
  - robust, 575–576, 579
- Testbench, 685
- Throughput
  - digital computation, 119
- TILOS, 270, 295
  - heuristic, 299
- Timing, 273, 551
  - analysis, 551
    - algorithm, 569
    - block oriented, 552
    - functional, 551
    - static, 296, 348, 567, 647, 655, 662, 684
  - closure, 146
  - driven layout, 641
  - optimization in MIS, 199
  - simulation, 277, 303
- Tolerance body, 586
- Tool
  - AS/X, 352
  - BooleDozer, 181
  - CATHEDRAL II, 96, 107
  - CHESS, 100
  - CINDERELLA, 100
  - Design Compiler, 181, 683
  - ESPRESSO, 182, 205
  - ESPRESSO Exact, 208
  - FASTHENRY, 403
  - FLEXWARE, 100
  - GOALIE, 473, 489, 648
  - GORDIAN, 469, 666
  - Harmonica, 388
  - HYPER-LP, 117

- IC-CAP, 272
- IRSIM, 278
- JiffyTune, 271, 347
- KISS, 185
- LANCELOT, 347, 351
- LSS, 181
- McBoole, 205
- MIMOLA, 94
- MIS, 183, 191
- PRIMA, 373, 438, 635
- PROUD, 469
- RICE, 442
- SIS, 183
- SPECS2, 270
- SPECS, 303, 347, 351
- SPICE, 313, 383, 565
- System Architect Workbench, 94
- TECAP2, 270
- TILOS, 295
- TRANALYSE, 337
- TRANGEN, 665
- UTMOST, 272
- Trade-off
  - area-delay, 185
- TRANALYZE, 271
- TRANGEN, 665
- Transfer function, 454
- Transformation
  - algebraic, 655
  - architectural, 126
  - VLSI circuits, 120
- Transistor
  - bidirectional model, 337
  - channel-connected component, 341
  - extraction, 491
  - sizing, 270, 273, 347, 635, 686
    - equation-based, 273
    - simulation-based, 273
- Transition
  - spurious, 121
- Transmission lines, 383
- Tree/link formulation, 303
- Tree mapping, 184
- Trust region, 347, 354
- Truth table, 182
- Tuning
  - dynamic, 348
  - static, 348
- Ugate, 253
  - factor-free, 262
- User interface, 347, 351, 359
- UTMOST, 272
- Variability, 557
  - within-die, 557
- Variable accuracy, 303
- Variable lifetime, 114
- Vertex coloring, 113
- Very Large Instruction Word, 159
  - See also* VLIW
- VLIW, 96, 100–101, 159
- Voltage
  - estimation of, 124
  - piecewise approximate, 304
  - supply, 126
- Wave-pipelining, 597
- Wire length minimization
  - quadratic, 636
- Worst-case
  - analysis, 557
  - distance, 588
  - parameter set, 588
- Yield, 557, 563
  - optimization, 591
  - parametric, 586

# Author Index

- Aarts, E. H. L., 675  
Antreich, K. J., 585  
Ashar, P., 663  
Berkelaar, M., 653  
Berman, C. L., 29, 217  
Billon, J. P., 17  
Boderson, R. W., 117  
Bootehssaz, A., 683  
Brand, D., 65  
Brayton, R. K., 181, 191  
Bryant, R. E., 3, 337  
Camposano, R., 683  
Carloni, L. P., 143  
Celik, M., 433  
Chakradhar, S. T., 663  
Chandrakasan, A. P., 117  
Chowdhury, D., 683  
Cong, J., 235  
Conn, A. R., 347  
Coudert, O., 17, 39  
Coulman, P. K., 347  
Darringer, J. A., 181, 645  
De Man, H., 93, 107  
Demir, A., 413  
Detjens, E., 191  
De Veciana, G., 159  
Ding, Y., 235  
Director, S. W., 563  
Dunlop, A. E., 295  
Feldmann, P., 451  
Fishburn, J. P., 295  
Freund, R. W., 451  
Fujiyoshi, K., 535  
Getreu, I. E., xi  
Ginneken, L.P.P.P. van, 479  
Golembeski, J. J., xi  
Goossens, G., 107  
Graeb, H. E., 585  
Gregory, B., 683  
Grodstein, J., 249  
Gupta, A., 663  
Haring, R. A., 347  
Harjani, R., 269  
Harkness, H., 249  
Hayashi, H., 639  
Henkel, J., 663  
Jacome, M. F., 159  
Jayaraman, R., 689  
Jha, N., 575  
Johannes, F. M., 499  
Kajitani, Y., 535  
Kamon, M., 403  
Kleinhan, J. M., 499  
Krishna, S., 191  
Kukula, J. H., 3, 683  
Kundert, K. S., 367, 383  
Kundu, S., 575  
Lapinskii, V., 159  
Lehman, E., 249  
Liu, E., 413  
Madre, J. C., 17, 39  
Magarshack, P., 269  
Malik, S., 129  
Marques Silva, J. P., 73  
Mas, G., 269  
Ma, T., 191  
McGeer, P., 191  
McMillan, K. L., 143  
Morrill, G. L., 347  
Murata, H., 535  
Nakatake, S., 535  
Nassif, S. R., 551, 563  
O'Brien, P. R., 393  
Odabasioglu, A., 433  
Otten, R.H.J.M., 479  
Pei, L., 191  
Phillips, N., 191  
Pileggi, L. T., 433  
Potkonjak, M., 117  
Rabaey, J., 93, 107, 117  
Raghunathan, A., 663  
Reddy, S., 575  
Rhines, W. C., 659  
Rohrer, R. A., 303  
Rudell, R., 51, 191, 205, 615  
Rutenbar, R. A., 269  
Sakallah, K. A., 73, 551  
Saldanha, A., 143

- Sangiovanni-Vincentelli, A. L., 143, 191, 205, 383, 413  
Savarino, T. L., 393  
Scheffer, L. K., 633  
Segal, R., 191  
Shenoy, N., 597, 615, 683  
Sigl, G., 499  
Smithhisler, C., 403  
Stok, L., 645  
Strojwas, A. J., 563  
Szymanski, T. G., 489, 597  
Theeuwen, J. F. M., 675  
Tiwari, V., 129  
Trevillyan, L. H., 29, 217, 645  
Tsuk, M. J., 403  
Vandewalle, J., 107  
Van Wyk, C. J., 489  
Visweswariah, C., 303, 347  
Wakabayashi, K., 663  
Walker, D. M. H., 551  
Wang, A., 191  
Watanabe, Y., 249  
Weil, P. B., xi  
White, J. K., 367, 403  
Williams, T., 683  
Wolfe, A., 129  
Wong, D. F., 521  
Yang, H. H., 521  
Yung, R., 191

# Reference Index

- Aagaard, M. D., 9  
Aarts, E. H. L., 97, 676  
Abraham, J. A., 552, 555  
Acuna, E. L., 278  
Aftab, S., 557  
Aggarwal, R., 641  
Agrawal, V., 666  
Agrawal, V. D., 552, 555  
Ahmad, I., 97  
Aitken, R. C., 555  
Akers, S. B., 5–6  
Alain, G., 279–280  
Alatan, L., 369  
Allen, F. E., 647  
Allstot, D. J., 277  
Aloul, F., 693  
Alpert, C. J., 469  
Alt, J., 556  
Anjo, K., 664  
Antreich, K. J., 685, 557  
Ashar, P., 667  
Bahnsen, R. J., 5, 8, 645  
Baker, Jr. G. A., 372  
Baker, K., 556  
Baklashov, M., 555  
Balakrishnan, A., 665  
Balasa, F., 97  
Barbacci, M., 94–95  
Barett, G., 279  
Barnes, J., 557  
Bartlet, K., 181  
Bartley, M., 279  
Bayraktaroglu, I., 555  
Beattie, M., 369  
Beer, I., 646  
Ben-David, S., 646  
Benini, L., 98  
Benkoski, J., 552  
Bentley, J. L., 656  
Berman, C. L., 10, 183, 640, 646–647, 654, 679,  
    683–684  
Berhet, C., 8–9  
Betz, V., 692–693  
Bhattacharya, S., 665  
Biere, A., 10  
Billon, J. P., 7, 637, 684  
Bjesse, P., 10  
Blaauw, D., 369  
Blades, E. E., 276  
Blanton, R. D., 556  
Bolsens, I., 94, 97  
Bommu, S., 665  
Boole, G., 181  
Boots, H. M. J., 272  
Boppana, V., 642  
Borriello, G., 99  
Bose, S., 8  
Boyd, S. P., 277  
Brace, K. S., 7  
Bracken, J. E., 372–373  
Brand, D., 4, 11, 181, 183, 551–552, 640, 647, 654,  
    684  
Braun, G., 101  
Brayton, R. K., 7, 9, 95–96, 181–184, 274, 470,  
    552, 555, 637, 640–641, 647, 655, 678, 683,  
    692  
Brennan, P. A., 648  
Breuer, A. D., 468  
Breuer, M., 555–556  
Brewer, F., 97  
Bricaud, P., 279  
Brodersen, R. W., 98–99, 638, 649  
Brodnax, T. B., 648  
Brown, S., 693  
Bryan, M. J., 555  
Bryant, R. E., 6–7, 182, 271, 279–280, 636, 646,  
    655, 686  
Burch, J. R., 9, 11  
Buric, M., 95  
Burns, S., 693  
Burstein, M., 470  
Bushnell, M. L., 555  
Cain, R. G., 181, 647  
Camposano, R., 96, 468  
Cangellaris, A. C., 371  
Carley, L. R., 271, 275–277, 635  
Carloni, L. P., 102, 634, 649  
Carragher, R., 641

- Carter, W., 4  
 Casavant, A. E., 97  
 Catthoor, F., 94, 97  
 Celetti, M. D., 556  
 Celik, M., 370, 373, 635, 649, 684  
 Cerny, E., 11  
 Chakraborty, T. J., 555  
 Chakradhar, S., 665  
 Chakravarty, S., 556  
 Chandrasekhan, A. P., 98, 638, 649, 665  
 Chandrasekhar, M. S., 5  
 Chang, C., 97, 183  
 Chang, H., 277, 371, 552  
 Chan, T. F., 470  
 Chao, H. H., 181, 647  
 Charbon, E., 276  
 Chatterjee, M., 555  
 Chaudhary, K., 470, 692  
 Chawla, B. R., 277  
 Chen, C., 97  
 Chen, H., 470–471, 552  
 Chen, H. C., 181, 552  
 Chen, K. C., 8, 555  
 Chen, W., 556  
 Cheng, C., 666  
 Cheng, C. K., 468–469, 471  
 Cheng, D., 8  
 Cheng, D. D., 647  
 Cheng, K.-T., 8, 181–183, 555–556  
 Cheung, N., 664  
 Chickermane, V., 555  
 Chiprout, E., 371–372  
 Choi, H., 101  
 Chou, P., 99  
 Christensen, O., 95  
 Chu, C. Y., 279  
 Chung, E.-Y., 98  
 Chung, K., 692  
 Cimatti, A., 10  
 Claesen, L., 96, 552  
 Clarke, E. M., 8–10  
 Clark, N. R., 551  
 Cocke, J., 647  
 Coehlo, C. N., 97  
 Cohen, W., 181  
 Cohn, J. M., 276  
 Cong, J., 184, 636, 641, 655, 692  
 Conn, A. R., 271, 273–274, 278, 648, 686  
 Conradt, K. W., 5  
 Conway, L., 369  
 Cory, W. E., 4  
 Coudert, O., 7–9, 183, 637, 640, 684–685  
 Coulman, P. K., 271, 273–274, 278, 648, 686  
 Cox, P., 557  
 Crowet, F., 678  
 Culliney, J. N., 182  
 D'Abreu, M. A., 97  
 Dagenais, M. R., 553  
 Dai, W. M., 470  
 Darringer, J., 4, 181, 645, 647  
 Dartu, F., 370  
 Daveau, J. M., 99  
 Davidson, E., 645  
 Davis, M., 5  
 Dec, A., 376  
 Decher, P. H., 270  
 DeGeus, A. J., 181, 277  
 DeGrauw, M. G., 276  
 De, K., 555  
 Del Mar Hershenson, M., 277  
 De Man, H., 96–97, 552  
 De Micheli, G., 96–98, 100, 181  
 Demir, A., 375–376  
 Denyer, P., 97  
 Deokar, R. B., 655  
 Deprettere, E., 98  
 Dervenis, J. P., 278  
 Desineni, R., 556  
 Despain, A., 101  
 Detjens, E., 95, 183, 637, 640–641, 647, 683, 692  
 Devadas, S., 5, 181–182, 552, 555–556  
 De Veciana, G., 101, 649  
 Devgan, A., 278  
 Dey, S., 555, 663  
 Dijkstra, E. W., 624, 656  
 Dill, D. L., 9  
 Ding, Y., 184, 692  
 Director, S. W., 557, 685  
 Donath, W. E., 10  
 Drumm, A. D., 181, 647  
 Du, D., 181, 552  
 Dunlop, A. E., 270, 273–274, 636, 648  
 Dutta, R., 470  
 Dutt, N., 97  
 Dwarkanath, K. N., 556  
 Dworak, J., 556  
 Ebeling, C., 553, 693  
 Edahiro, M., 666  
 Eichelberger, E. B., 555  
 Eijk, C. van, 11, 679  
 Eijndhoven, J. T. J. van, 96, 655  
 Eisenmann, H., 656  
 Eisner, C., 646  
 Elfadel, I. M., 278, 372–373  
 Elmasry, M. I., 277  
 Elmore, W. C., 369  
 El-Turky, F., 276  
 Emerson, E. A., 8  
 Entrena, L. A., 181, 183  
 Ernst, R., 98, 100, 103  
 Essink, G., 677  
 Fallah, F., 642  
 Fang, T.-T., 371  
 Fan, M., 470

- Feldmann, P., 279, 372, 374–376, 684  
Felt, E., 557  
Fiduccia, C. M., 468  
Fink, F., 556  
Finlay, I., 97  
Fishburn, J., 270, 273–274, 636, 648  
Fisher, A. L., 8  
Flynn, M. J., 277  
Fraisse, H., 183  
Francis, R. J., 692  
Frankle, J., 693  
Freund, R. W., 372, 375, 684  
Fuchs, K., 556  
Fuchs, W. K., 642  
Fujisawa, H., 7  
Fujita, M., 7, 11, 640, 642  
Fujiyoshi, K., 470, 641, 648  
Fummi, F., 555  
Gala, K., 369  
Gallivan, K., 372  
Ganai, M., 667  
Gao, M., 182  
Gao, P. N., 471  
Garrod, D. J., 276  
Geiger, D. J., 181, 647  
Geilert, M., 556  
Gelatt, C. D., 648, 692  
George, B. J., 279  
George, E. W., 557  
Gerbi, J. V., 181  
Ghanta, S., 552  
Ghosh, I., 642  
Gielen, G., 276–277  
Ginneken, L. P. P. van, 470, 641, 648  
Givargis, T., 663  
Glasser, L. A., 273  
Glover, K., 372  
Goel, D. K., 556  
Goertzel, G., 273  
Goldberg, E., 9  
Goossens, G., 94, 96, 100  
Gopal, N. G., 279  
Goto, S., 663  
Gouget, G., 272  
Goust, T., 279  
Graeb, H., 557, 685  
Graham, R., 471  
Grant, D., 97  
Graves-Morris, P., 372  
Gray, P. R., 270, 275–277, 635  
Gregory, D., 181  
Greiner, A., 279  
Grimme, E., 372  
Grodstein, J. E., 184–185, 647, 654, 683  
Gronthoud, G., 556  
Gulati, R. K., 556  
Gummel, H. K., 277  
Guo, P., 666  
Gupta, A., 667  
Gupta, R., 555  
Gupta, S., 276, 556  
Hachtel, G. D., 95, 182–183, 555  
Hafer, L., 94  
Hagen, L., 468  
Halbwachs, N., 279  
Halliwell, H., 5, 645  
Hammer, P. L., 5  
Haring, R. A., 271, 273–274, 278, 648, 686  
Harjani, R., 271, 275–277, 635  
Harkness, H., 184–185, 647, 683  
Harrigan, E., 371  
Harvey, J. P., 277  
Hasan, M. M., 276  
Hashizume, M., 276  
Hathaway, D. J., 645  
Hauck, S. A., 693  
Haynal, S., 97  
Hedlund, K. S., 273  
Heineken, H., 557  
Hemachandra, L., 95, 182  
Henkel, J., 663  
Herr, N., 557  
Hieter, N., 181, 647  
Higuchi, H., 640  
Hirose, F., 639  
Hitchcock, R. B., 552, 647  
Hocevar, D., 557  
Hoffman, C. M., 184  
Hoffmann, A., 101  
Hojati, R., 9  
Hong, S. J., 181, 647  
Horowitz, M. A., 279, 369–370  
Hoyle, L. P. J., 273  
Hsu, C. P., 469  
Hsu, F. F., 555  
Huang, I., 101  
Huang, J., 469  
Huang, M. D., 676  
Huang, S.-Y., 8  
Hur, S. W., 470  
Hu, T. C., 468  
Hwang, K. S., 97  
Hwang, L. J., 9  
Hwang, S. H., 101  
Ishii, A. T., 553  
Ishikawa, M., 666  
Ishiura, N., 7  
Itakura, T., 376  
Iwashita, H., 640  
Iyengar, V., 551–552  
Jackson, M., 470, 472  
Jacoby, R., 555  
Jacome, M. F., 101, 649  
Jain, A., 279

- Jain, J., 11, 640  
Jain, R., 555  
Jain, S., 279  
Jerraya, A. A., 99  
Jha, N., 664  
Jha, N. K., 96, 98, 554–556, 685  
Jiang, J.-H., 182  
Jiang, Y., 182  
Jiang, Y. M., 556  
Johannes, F. M., 469, 636, 641, 648, 692  
Johansen, D., 95  
Johnson, S., 186  
Jones, M., 371  
Jones, R. B., 9  
Jongeneel, D., 184  
Jouppi, N. P., 552  
Joyner, W., 4, 181, 183, 647  
Juge, A., 272  
Jusuf, G., 277  
Kahng, A. B., 468–469, 636  
Kajitani, Y., 470, 641, 648  
Kambayashi, Y., 182  
Kamon, M., 367–368, 371–372, 635, 649  
Kam, T., 279  
Kanbara, H., 277  
Kanjilal, S., 665  
Kao, W. H., 273  
Kastner, R., 470  
Kawaii, H. Y., 276  
Kawamura, K., 641  
Kawato, N., 4, 7  
Keating, M., 279  
Kelsey, T. P., 556  
Kernighan, B. W., 468, 656, 692  
Kerns, K. J., 373  
Keutzer, K., 181–182, 184, 552, 555  
Khalily, E., 270  
Khare, J., 557  
Khouri, G., 96  
Kim, J., 94  
Kirkpatrick, S., 648, 676, 692  
Kirkpatrick, T. I., 551, 655  
Klaasen, D. B. M., 272  
Kleihorst, R. P., 678  
Kleinhan, J. M., 469, 636, 641, 648, 692  
Koenemann, B., 645  
Koh, C. K., 636  
Koh, H. Y., 270, 275, 277, 635  
Kolson, D., 97  
Korst, J., 97  
Koushanfar, F., 94  
Kozak, P., 277  
Krauter, B., 368–369  
Krishna, S., 95, 183, 637, 640–641, 647, 683, 692  
Krohm, F., 11  
Krstic, A., 556  
Kruiskamp, W., 276  
Kuehlmann, A., 8, 11, 646, 654  
Kuh, E. S., 468–470, 472, 666, 692  
Kukimoto, Y., 184, 655  
Kukula, J. H., 8–9  
Kundert, K. S., 371, 373–376, 634–635, 649  
Kundu, S., 554–556, 685  
Kung, D. S., 181, 647  
Kunz, W., 11, 182–183, 555, 654  
Kurdahi, F. J., 97  
Kyung, C.-M., 101  
Lahiri, K., 663  
Lai, H. C., 182  
Lai, K., 555  
Lajolo, M., 663  
Lakshminarayana, G., 96, 555, 663  
Lalgudi, K. N., 553  
Lam, J., 676  
Landver, A., 646  
Langevelde, R., 272  
Lapinskii, V., 101, 649  
LaPotin, D., 8, 646  
Laurentin, M., 279  
Lavagno, L., 182–183, 663  
Lavin, M., 645  
Lee, C. Y., 6, 467, 692  
Lee, E., 103  
Lee, M. T. C., 555  
Lee, T. H., 277  
Leenaerts, D., 276  
Lehman, E., 184–185, 647, 683  
Leiserson, C. E., 553, 647, 678  
Leive, G., 94  
Lekatsas, H., 664  
Lengauer, T., 468  
Leonard, T., 10  
Leung, B., 277  
Levy, H., 371, 373  
Lieverse, P., 98  
Lillis, J., 470  
Lin, B., 9, 94, 99  
Lin, C. C., 555, 642  
Lin, C. J., 556  
Lin, S., 468, 470, 692  
Lin, T.-M., 369–370  
Ling, D., 371–372  
Liou, J. J., 556  
Lippens, P. E. R., 94, 97, 678  
Li, S., 94  
Li, Y., 100, 182, 663  
Liu, E. W. Y., 375–376  
Liu, S., 557  
Lockyear, B., 553  
Logemann, G., 5  
Long, D., 9, 375–376  
Lousberg, M., 556  
Loveland, D., 5  
MacMillen, D., 371, 373

- Madigan, C., 9, 182  
Madre, J. C., 7–9, 183, 637, 640, 684–685  
Ma, H. K. T., 5, 556  
Mains, R. E., 648  
Majumdar, A., 555  
Malavasi, E., 276  
Malik, S., 7, 9, 99–100, 181–182, 552, 649, 667  
Maly, W., 556–557  
Mao, W., 556  
Maraninchi, F., 279  
Marbot, R., 279  
Marchioro, G. F., 99  
Marculescu, D., 181  
Marculescu, R., 181  
Marek-Sadowska, M., 183, 470, 555  
Marinissen, E. J., 555  
Marple, D., 274  
Maruyama, F., 4  
Marwedel, P., 100  
Ma, T., 95, 183, 637, 640–641, 647, 683, 692  
Matheson, T., 95  
Matson, M., 273  
Matsunaga, Y., 11, 640  
Matsushita, S., 664  
Mattheyses, R. M., 468  
Maulik, P. C., 277  
Mauras, C., 11  
McCluskey, E. J., 181, 555  
Mc Donald, C., 279  
McGeer, P., 95, 181, 183, 552, 556, 637, 640–641, 647, 683, 692  
McMillan, K. L., 9, 102, 634, 646, 649  
McMullen, C., 95, 181–183  
McMurchie, L., 693  
McWilliams, T. M., 552  
Mead, C. A., 369–370  
Mehra, R., 98  
Mehrotra, A., 375–376  
Mehrotra, S., 369  
Melville, R., 374  
Menezes, N., 370  
Mercer, M. R., 555  
Meyer, R. G., 375  
Meyr, H., 101  
Minato, S., 7  
Mokkedem, A., 10  
Molzen Jr., W. W., 278  
Moon, C., 183  
Moon, I.-H., 9  
Mooney, V., 100  
Moore, W., 556  
Morrell, J. K., 645  
Morrill, G. L., 271, 273–274, 278, 648, 686  
Morrison, C., 555  
Moskewicz, M., 9, 182  
Motomura, M., 664  
Mouthaan, T., 276  
Mudge, T. N., 553  
Mukherjee, R., 640  
Murata, H., 470, 641, 648  
Murgai, R., 183, 641  
Muroga, S., 182  
Nabors, K., 367, 371  
Nachtergaele, L., 97  
Nagasamy, V., 94  
Nagel, L., 375, 556  
Nagoya, A., 182  
Najm, F.N., 656  
Nakatake, S., 470–471, 641, 648  
Nakata, T., 640  
Nakhla, M. S., 372  
Nam, G., 693  
Narayan, A., 557, 640  
Narayanan, S., 555  
Narayanan, V., 635  
Nassif, S. R., 557, 685  
Nemirovsky, A., 273  
Neri, A., 375  
Nestrov, Y., 273  
Newton, A. R., 9, 95, 182, 556  
Nguyen, T. V., 279  
Nicolau, A., 97  
Nigh, P., 556  
Nii, K., 276  
Ning, Z., 276  
Nishi, N., 664  
Nohl, A., 101  
Note, S., 97  
Novikov, Y., 9  
Oakamoto, T., 664  
O'Brien, P. R., 278, 369–371, 634–635, 649, 684  
Ochotta, E. S., 277  
Odabasioglu, A., 370, 373, 635, 649, 684  
Odawara, G., 5  
O'Donnell, M. J., 184  
Ofek, H., 10  
Ohta, T., 5  
Ohtsuki, T., 468  
Okamoto, T., 666  
Okumura, M., 376  
Okuzawa, O., 5  
Oliveira, A., 641  
Olukotun, O. A., 553  
Onodera, H., 277  
Opalski, L., 557  
Orailoglu, A., 555  
Orchard-Hays, W., 273  
Ortega, R., 99  
Osler, P. J., 181, 647  
Ostapko, D. L., 181, 647  
Otten, R. H. J. M., 96, 184, 470–471, 641, 648, 656, 676  
Ousterhout, J. K., 553  
Ozev, S., 555

- Pabst, M., 556  
 Pagones, A. J., 278  
 Panda, R., 369  
 Pandini, D., 276  
 Papachirstou, C., 555  
 Papaefthymiou, M. C., 553  
 Parameswaran, S., 664  
 Parker, A. C., 94  
 Park, I.-C., 101  
 Park, N., 97  
 Patel, J. H., 555  
 Patil, S., 556  
 Pederson, D. O., 374  
 Pedram, M., 181, 468  
 Pei, L., 95, 183, 637, 640–641, 647, 683, 692  
 Penfield, Jr. P., 369–370  
 Perremans, S., 552  
 Perry, T. S., 374  
 Phillips, J., 368, 371–372, 376  
 Phillips, N., 95, 183, 637, 640–641, 647, 683, 692  
 Piket-May, M. J., 371  
 Pileggi, L. T., 368–370, 373, 635, 649, 684  
 Pillage, L. T., 279, 372, 649  
 Pilotty, R., 95  
 Pirouz, B. S., 279–280  
 Pitchumani, V., 4  
 Pixley, C., 181  
 Pomeranz, I., 556  
 Potkonjak, M., 94, 97–98, 555, 638, 649, 666  
 Potlapally, N., 664  
 Pradhan, D., 11, 182–183, 555  
 Privitera, J. P., 5  
 Putnam, H., 5  
 Quine, W. V., 181  
 Rabaey, J., 94, 96–98, 638, 649  
 Raghavan, V., 372  
 Raghunathan, A., 98, 663  
 Raghunathan, V., 665  
 Rahmat, K., 645  
 Raimi, R., 10  
 Rajski, J., 183, 637, 640–641, 647, 683  
 Randall, K. H., 553  
 Rao, V. B., 648  
 Ratzlaff, C. L., 279  
 Ravi, K., 9  
 Ravi, S., 555, 664  
 Reddy, L. N., 181, 647  
 Reddy, S. M., 11, 554–556, 685  
 Rhodes, D., 100  
 Richards, D., 184  
 Richter, K., 98, 103  
 Rijmenants, J., 276  
 Rizzoli, V., 375  
 Robinson, G., 664  
 Rocheteau, F., 279  
 Roesner, W., 645  
 Roessel, T., 556  
 Rohrer, R. A., 270, 277–279, 372, 375, 648–649  
 Rose, J., 692–693  
 Rothweiler, S., 665  
 Roth, J. P., 4  
 Rothweiler, S., 666  
 Roychowdhury, J., 374–376  
 Roy, K., 555  
 Roy, R., 555  
 Rubinstein, J., 369–370  
 Rudeanu, S., 5  
 Rudell, R., 7, 95, 181–183, 551, 553, 637, 640–641, 645, 647, 655, 678–679, 683–685, 692  
 Rudnick, E. M., 555  
 Ruehli, A. E., 273, 368, 648  
 Rumin, N. C., 553  
 Rutenbar, R. A., 271, 275–277, 635, 693  
 Saito, T., 4  
 Sakallah, K. A., 9, 182, 552–553, 556, 634, 637, 640, 646, 684, 693  
 Saldanha, A., 102, 181–183, 555, 634, 649  
 Saleh, R. A., 278  
 Saluja, K., 555–556  
 Salz, A. S., 279  
 Sanghavi, J., 183  
 Sangiovanni-Vincentelli, A. L., 5, 7, 9, 95, 102–103, 181–183, 274, 276–277, 373, 375–376, 555–557, 634, 637, 640–641, 647, 649, 663, 683, 692  
 Sankaradass, M., 664  
 Sansen, W. M. C., 276–277  
 Sar-Dessai, V. R., 556  
 Sarrafzadeh, M., 468, 470  
 Sato, M., 470  
 Savarino, T. L., 369–371, 634–635, 649, 684  
 Savoj, H., 182–183  
 Sawada, H., 182  
 Sawkar, P., 184  
 Saxe, J., 553, 647  
 Saxena, J., 555  
 Schanzenbach, E., 645  
 Schliebusch, O., 101  
 Scholten, A. J., 272  
 Schulz, M. H., 556  
 Sciuto, D., 555  
 Sechen, C., 636, 677, 692  
 Segal, R., 95, 183, 637, 640–641, 647, 683, 692  
 Seger, C.-J. H., 9  
 Sentovich, E. M., 183  
 Sequin, C. H., 270, 275, 277, 635  
 Serra, M., 555  
 Shannon, C. E., 181  
 Shenoy, N., 551, 553, 647, 683–684  
 Shepard, K. L., 369, 635  
 Sherwani, A. N., 468  
 Shipley, T. R., 9  
 Shyu, J., 274  
 Siewiorek, D. P., 94

- Sigl, G., 469, 636, 641, 648, 692  
Silva, J. P. M., 9, 182, 552, 556, 634, 637, 640, 646, 684, 693  
Silveira, L. M., 371–372  
Singhal, K., 557  
Singhal, V., 11  
Singh, K. J., 183, 274, 640  
Sinha, A., 98  
Sinha, S., 182  
Sistla, A. P., 8  
Sivaraman, M., 556  
Six, P., 96  
Smith, G., 5, 645, 647  
Smithhisler, C., 367, 635, 649  
Somenzi, F., 9, 640  
Soreff, J. P., 648  
Soukup, J., 468  
Srinivasan, A., 8, 470, 472, 646, 692  
Stabler, E., 4  
Stefanov, T., 98  
Stein, A. D., 279  
Stephan, P. R., 183  
Stoffel, D., 11  
Stok, L., 181, 647  
Strehl, K., 98  
Strenski, P. N., 278  
Strojwas, A. J., 556–557, 685  
Styblinski, M., 557  
Subrahmanyam, P. A., 279  
Sugawara, T., 376  
Sullivan, A. J., 181, 647  
Sun, F., 664  
Sun, J., 99  
Sun, S. Z., 552  
Sutherland, I., 654  
Suyama, K., 376  
Swartz, W., 636  
Szepieniec, A. A., 471  
Szymanski, T., 473, 551, 553–554, 648, 684  
Tabbara, A., 470  
Taflove, A., 371  
Takahashi, T., 471, 666  
Tamaru, K., 277  
Tamesada, T., 276  
Tamiya, Y., 641  
Tan, C. J., 553  
Tanimoto, H., 376  
Teegarden, D. A., 270  
Teich, J., 103  
Telichevesky, R., 375–376  
Tellez, G., 645  
Thiele, L., 98, 103  
Thomas, D. E., 94  
Thomas, V., 371  
Thuk, M. J., 367  
Tiwari, V., 97, 99, 649  
Tomita, M., 5  
Toth, L., 376  
Touati, H. J., 9  
Touba, A. S., 555  
Trevillyan, L. H., 10, 181, 183, 640, 645–647, 683–684  
Tsao, C. W., 636  
Tsay, R. S., 469, 472, 636, 641, 648  
Tsividis, Y., 272  
Tsuk, M. J., 367, 635, 649  
Tuan, T., 94  
Uehara, T., 4  
Underwood, B., 555  
Unger, S., 553  
Upadhyaya, S. J., 555  
Uppaluri, P., 556  
Vahid, F., 663  
VanCleemput, W. M., 4  
Van der Werf, A., 97  
Van der Wolf, P., 94, 98  
Vandewalle, J., 96  
Van Dooren, P., 372  
Vanhooft, J., 97  
Van Meerbergen, J. L., 97, 677  
Van Rootselaar, G., 94  
Van Wyk, C., 473, 648  
Vasudevamurthy, J., 183, 637, 640–641, 647, 683  
Vecchi, M. P., 648, 692  
Vercauteren, S., 99  
Verhaegh, W. F. J., 97, 678  
Vissers, K., 94  
Viswesvariah, C., 270–271, 273–274, 277–279, 648, 686  
Vranesic, Z., 692–693  
Wagner, T., 4  
Wahlen, O., 101  
Wakabayashi, K., 664  
Walker, D. M. H., 556–557  
Wallinga, H., 276  
Walscharts, H. C. C., 277  
Wang, A., 7, 95, 181, 183, 274, 637, 640–641, 647, 683, 692  
Wang, D., 470  
Wang, J., 369  
Wang, M., 470  
Wang, S., 665  
Wang, W., 665  
Watanabe, Y., 184–185, 647, 683  
Webb, R. W., 557  
Weber, L., 375  
Wei, Y. C., 468  
Wemple, I. L., 373  
Werf, A., 678  
Whan, C. B., 278  
Whetsel, L., 555  
White, J. K., 367–368, 371–373, 375–376, 635, 649  
Williams, T. W., 555  
Wolfe, A., 99–100, 649

- Wolf, P. K., 273  
Wolf, W., 94, 100, 664  
Wong, C. K., 468  
Wong, D. F., 468, 648  
Wyatt, Jr. J. L., 369  
Xiong, X. M., 470  
Yaghutiel, H., 276  
Yajima, S., 7  
Yalcin, H., 552  
Yamashita, S., 182  
Yang, A. T., 373  
Yang, H., 468, 648  
Yang, P., 557  
Yang, X., 470  
Yao, B., 471  
Yen, H. C., 552  
Ye, W., 100  
Yoshimura, T., 471, 664, 666  
Young, B., 369  
Youssef, M. N., 470  
Yung, R., 95, 183, 637, 640–641, 647, 683, 692  
Yusim, I., 376  
Zarrineh, K., 555  
Zhang, L., 9, 182  
Zhao, L., 371  
Zhao, Y., 9, 182  
Zhong, T., 369, 635  
Zhuang, Z., 5  
Zhu, Y., 10  
Ziegenbein, D., 98, 103  
Zimmermann, G., 94, 100  
Zimmermann, M., 556  
Zolotov, V., 369  
Zorian, Y., 555