

DIGITAL INTEGRATED CIRCUITS

A DESIGN PERSPECTIVE
SECOND EDITION



MOSAID Technologies Inc.
EX 2003 - IPR2024-00599
MediaTek, Inc. v. MOSAID
Technologies Inc.
Page 1 of 8

JAN M. RABAEY
ANANTHA CHANDRAKASAN
BORIVOJE NIKOLIĆ

PRENTICE HALL ELECTRONICS AND VLSI SERIES
CHARLES G. SODINI, SERIES EDITOR

Library of Congress Cataloging-in-Publication Data on file.

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Publisher: *Tom Robbins*
Editorial Assistant: *Eric Van Ostenbridge*
Vice President and Director of Production and Manufacturing, ESM: *David W. Riccardi*
Executive Managing Editor: *Vince O'Brien*
Managing Editor: *David A. George*
Production Editor: *Daniel Sandin*
Director of Creative Services: *Paul Belfanti*
Creative Director: *Carole Anson*
Art and Cover Director: *Jayne Conte*
Art Editor: *Greg Dulles*
Manufacturing Manager: *Trudy Piscioti*
Manufacturing Buyer: *Lisa McDowell*
Marketing Manager: *Holly Stark*

About the Cover: Detail of "Wet Orange," by Joan Mitchell (American, 1925–1992). Oil on canvas, 112 × 245 in. (284.5 × 622.3 cm). Carnegie Museum of Art, Pittsburgh, PA. Gift of Kaufmann's Department Store and the National Endowment for the Arts, 74.11. Photograph by Peter Harholdt, 1995.



© 2003, 1996 by Pearson Education, Inc.
Pearson Education, Inc.
Upper Saddle River, NJ 07458

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher shall not be liable in any event for incidental and consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-597444-5

Pearson Education Ltd., *London*
Pearson Education Australia Pty, Ltd., *Sydney*
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd., *Hong Kong*
Pearson Education Canada Inc., *Toronto*
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education—Japan, *Tokyo*
Pearson Education Malaysia, Pte. Ltd.
Pearson Education Inc., *Upper Saddle River, New Jersey*

MOSAID Technologies Inc.
EX 2003 - IPR2024-00599
MediaTek, Inc. v. MOSAID
Technologies Inc.
Page 2 of 8

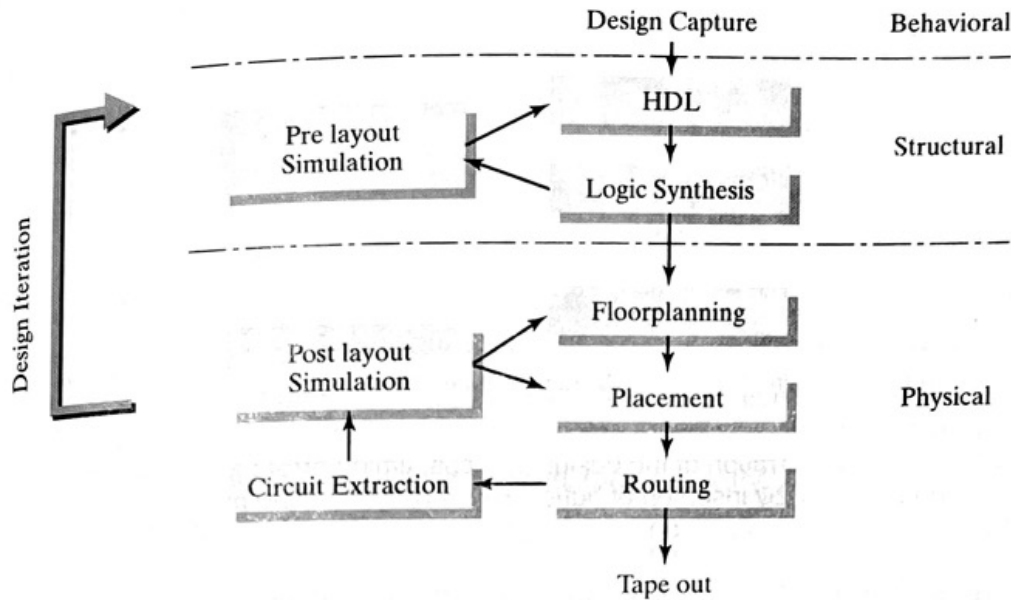


Figure 8-16 The Semicustom (or ASIC) design flow.

5. **Placement.** The precise positioning of the cells is decided.
6. **Routing.** The interconnections between the cells and blocks are wired.
7. **Extraction.** A model of the chip is generated from the actual physical layout, including the precise device sizes, devices parasitics, and the capacitance and resistance of the wires.
8. **Postlayout Simulation and Verification.** The functionality and performance of the chip is verified in the presence of the layout parasitics. If the design is found to be lacking, iterations on the floorplanning, placement, and routing might be necessary. Very often, this might not solve the problem, and another round of the structural design phase might be necessary.
9. **Tape Out.** Once the design is found to be meeting all design goals and functions, a binary file is generated containing all the information needed for mask generation. This file is then sent out to the ASIC vendor or foundry. This important moment in the life of a chip is called *tape out*.

While the design flow just described has served us well for many years, it was found to be severely lacking once technology reached the 0.25- μm CMOS boundary. With design technology proceeding into the deep submicron region, layout parasitics—especially from the interconnect—are playing an increasingly important role. The prediction models used by the logic and structural synthesis tools have a hard time providing accurate estimates for these parasitics. The chances that the generated design meets the timing constraints at the first try are thus very small (Figure 8-17a). The designer (or design team) is then forced to go through a number of costly iterations of synthesis followed by layout generation until an acceptable artwork that meets the timing constraints is obtained (Figure 8-17b and c). Each of these iterations may take several days—just routing a complex chip can take a week on the most advanced computers! The

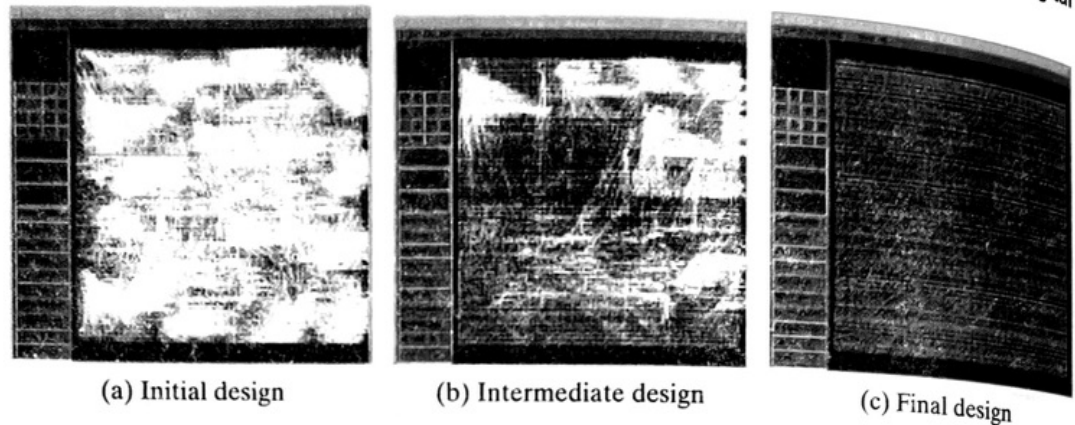


Figure 8-17 The timing closure process. The white lines indicate nets with timing violations. In each iteration of the design process, timing errors are removed by optimizing the logic, by insertion of buffers, by constraining the placement, or by streamlining the routing until an error-free design is obtained [Avanti01].

number of needed iterations continues to grow with the scaling of technology. This problem, called *timing closure*, made it obvious that new solutions and a change in design methodology were required.

The common answer is to create a tighter integration between the logical and physical design processes. If the logic synthesis tool, for example, also performs some part of the placement—or directs the placement—more precise estimates of the layout parameters can be obtained. Figure 8-18 shows an example of a design environment that merges RTL synthesis with first-order placement and routing. The resulting netlist is then fed into an optimization tool that performs the detailed placement and routing, while guaranteeing the timing constraints are met. While this approach has shown to be quite successful in reducing the number of design iter-

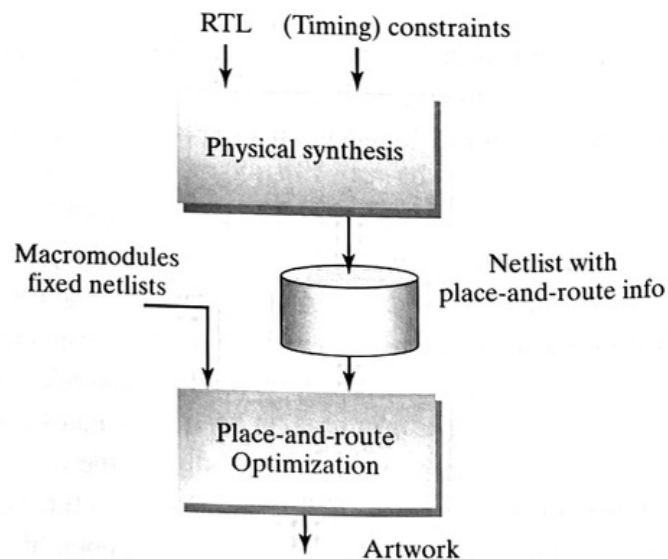


Figure 8-18 Integrated synthesis place-and-route reduces the number of iterations to reach timing closure in deep submicron.

ations, it throws quite a challenge at the design-tool developers. With the number of parasitic effects increasing with every round of technology scaling, the design optimization process that must take all this into account becomes exponentially complex as well. As a result, other approaches might be required as well. In the coming chapters, we will highlight "design solutions" that can help to alleviate some of these problems. An example is the use of regular and predictable structures, both at the logical and the physical level.

8.5 Array-Based Implementation Approaches

While design automation can help reduce the design time, it does not address the time spent in the manufacturing process. All of the design methodologies discussed thus far require a complete run through the fabrication process. This can take from three weeks to several months, and it can substantially delay the introduction of a product. Additionally, with ever-increasing mask costs, a dedicated process run is expensive, and product economics must determine if this is a viable route.

Consequently, a number of alternative implementation approaches have been devised that do not require a complete run through the manufacturing process, or they avoid dedicated processing completely. These approaches have the advantage of having a lower NRE (nonrecurring expense) and are, therefore, more attractive for small series. This comes at the expense of lower performance, lower integration density, or higher power dissipation.

8.5.1 Prediffused (or Mask-Programmable) Arrays

In this approach, batches of wafers containing arrays of primitive cells or transistors are manufactured by the vendors and stored. All the fabrication steps needed to make transistors are standardized and executed without regard to the final application.

To transform these uncommitted wafers into an actual design, only the desired interconnections have to be added, determining the overall function of the chip with only a few metallization steps. These layers can be designed and applied to the premanufactured wafers much more rapidly, reducing the turnaround time to a week or less.

This approach is often called the *gate-array* or the *sea-of-gates* approach, depending on the style of the prediffused wafer. To illustrate the concept, consider the gate-array primitive cell shown in Figure 8-19a. It comprises four NMOS and four PMOS transistors, polysilicon gate connections, and a power and ground rail. There are two possible contact points per diffusion area and two potential connection points for the polysilicon strips. We can turn this cell, which does not implement any logic function so far, into a real circuit by adding some extra wires on the metal layer and contact holes. This is illustrated in Figure 8-19b, where the cell is turned into a four-input NOR gate.

The original *gate-array* approach² places the cells in rows separated by wiring channels, as shown in Figure 8-20a. The overall look is similar to the traditional standard-cell technique. With the advent of extra metallization layers, the routing channels can be eliminated, and routing can

²This approach is often called the *channeled* gate array.

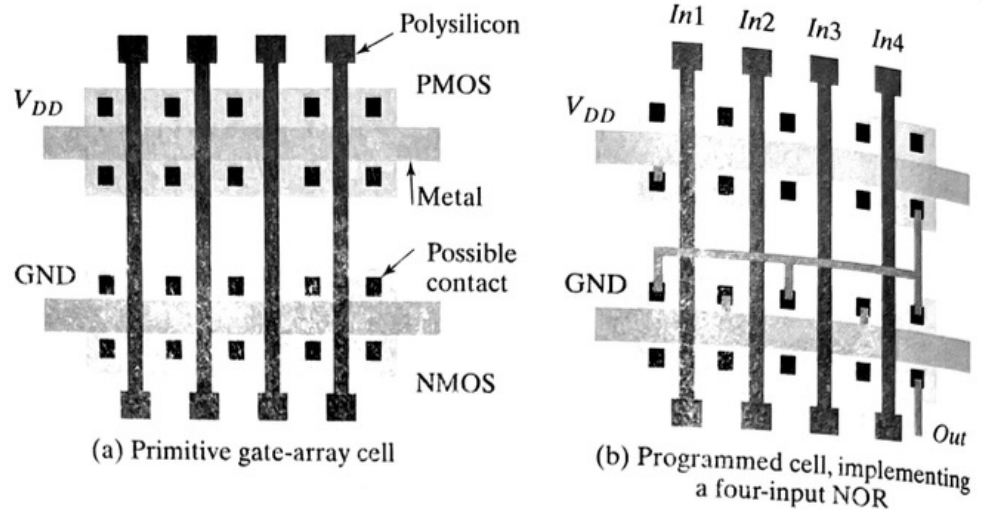


Figure 8-19 An example of the gate-array approach.

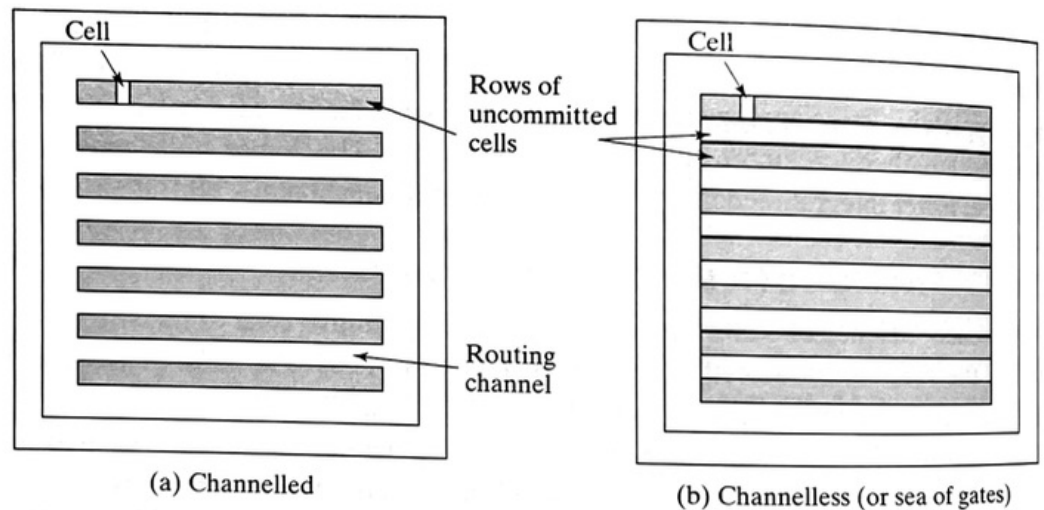


Figure 8-20 Gate-array architectures.

be performed on top of the primitive cells—occasionally leaving a cell unused. This channelless architecture, also called *sea of gates* (Figure 8-20b), yields an increased density, and makes it possible to achieve integration levels of millions of gates on a single die. Another advantage of the sea-of-gates approach is that it customizes the contact layer between metal-1 and diffusion and/or polysilicon, in contrast to the standard gate-array approach where the contacts are predefined (see Figure 8-19a). This extra flexibility leads to a further reduction in cell size.

The primary challenge when designing a gate-array (or sea-of-gates) template is to determine the composition of the primitive cell and the size of the individual transistors. A sufficient number of wiring tracks must be provided to minimize the number of cells wasted to interconnect. The cell should be chosen so that the prefabricated transistors can be utilized to a maximal extent over a wide range of designs. For example, the configuration of Figure 8-19 is well suited for the realization of four-input gates, but wastes devices when implementing two-input gates.

Verification, on the other hand, attempts to extract the system parameters directly from the circuit description. For instance, the critical path of an adder can be recognized from an inspection of the circuit diagram or a model of it. This approach has the advantage that the result is independent of the choice of excitation vectors and is supposedly foolproof. On the other hand, it relies on a number of implicit assumptions regarding design techniques and methodologies. For the example of the adder, determination of the propagation delay requires an understanding of the logic operation of the composing circuitry—for example, dynamic or static logic—and a definition of the term *propagation delay*. As a second example, it is necessary to identify the register elements first before the maximum clock speed of a synchronous circuit can be determined. The resulting tools are therefore restricted in scope and handle only a limited class of circuit styles, such as single-phase synchronous design. In this insert, we analyze a number of the popular verification techniques.

Electrical Verification

Given the transistor schematics of a digital design, it is possible to verify whether a number of basic rules are satisfied. Some examples of typical rules help illustrate this concept, and many others can be derived from previous chapters:

- The number of inversions between two C²MOS gates should be even.
- In a pseudo-NMOS gate, a well-defined ratio between the PMOS pull-up and NMOS pull-down devices is necessary to guarantee a sufficient noise-margin low NM_L .
- To ensure that rise and fall times of the signal waveforms stay within limits, minimum bounds can be set on the sizes of the driver transistors as a function of the fan-out.
- The maximum amount of charge sharing in a dynamic design should be such that the noise-margin high is not violated.

Pure common sense can help define a large set of rules to which a design should always adhere. Applying them requires an in-depth understanding of the circuit structure. An electrical verifier, therefore, starts with the identification of well-known substructures in the overall circuit schematic. Typical templates are simple logic gates, pass transistors, and registers. The verifier traverses the resulting network on a rule-per-rule basis. Since electrical rules tend to be specific to a particular design style, they should be able to be easily modified. For example, *rule-based expert systems* allow for an easy updating of the rule base [DeMan85]. The individual rules can be complex and even invoke a circuit simulator for a small subsection of the network to verify whether a given condition is met. In summary, electrical verification is a helpful tool, and it can dramatically reduce the risk of malfunction.

Timing Verification

As circuits become more complex, it is increasingly difficult to define exactly which paths through the network are critical with respect to timing. One solution is to run extensive SPICE simulations that may take a long time to finish. Even then, this does not guarantee that the iden-

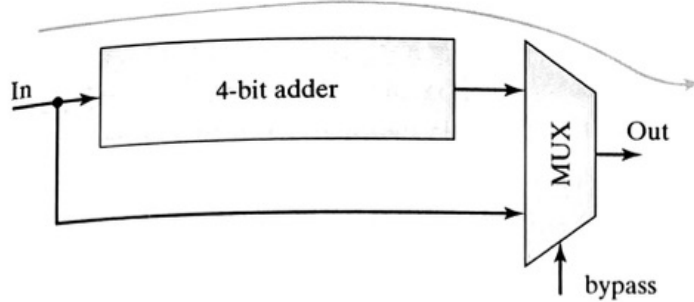


Figure G-1 Example of false path in timing verification.

identified critical path is the worst case, since the delay path is a function of the applied signal patterns. A timing verifier traverses the electrical network and rank-orders the various paths, based on delay. This delay can be determined in a number of ways. One approach is to build an *RC*-model of the network and compute bounds on the delay of the resulting passive network. To obtain more accurate results, many timing verifiers first extract the details of the longest path(s), based on the *RC*-model, and perform circuit simulation on the reduced circuit in order to obtain a better estimation. Examples of early timing verifiers are the Crystal [Ousterhout83] and TV [Jouppi84] systems.

One problem that hampered many earlier systems is that they identified *false paths*—that is, critical paths that can never be exercised during normal circuit operation. For example, a false path exists in the carry-bypass adder discussed in Chapter 11 (also shown in simplified form in Figure G-1). From a simple analysis of only the circuit topology, one would surmise that the critical path of the circuit passes through the adder and multiplexer modules as illustrated by the arrow. A closer look at the circuit operation reveals that such a path is not feasible. All individual adder bits must be in the propagate mode for *In* to propagate through the complete adder. However, the *bypass* signal is asserted under these conditions, and the bottom path through the multiplexer is selected instead. The actual critical path is thus shorter than what would be predicted from the first-order analysis. Detecting false timing paths is not easy, since it requires an understanding of the logic functionality of the network. Newer timing verifiers are remarkably successful in accomplishing this and have become one of the more important design aids for the high-performance circuit designer [Devadas91].

Example G.1 Example of Timing Verification

The output of a static timing verifier, the PathMill tool from Synopsys [PathMill], is shown in Figure G-2. The input to the verification process is a transistor netlist, but gate- and block-level models can be included as well. The analysis considers capacitive and resistive parasitics that are obtained from the transistor schematic or layout extraction.

The output of the timing analysis is an ordered list of critical timing paths. For the example of Figure G-2, the longest path extends from node *B* (falling edge) to node *Y* (falling edge) over nodes *S1*, *S2*, and *X3*. The predicted delay is 1.14 ns. Other paths close to