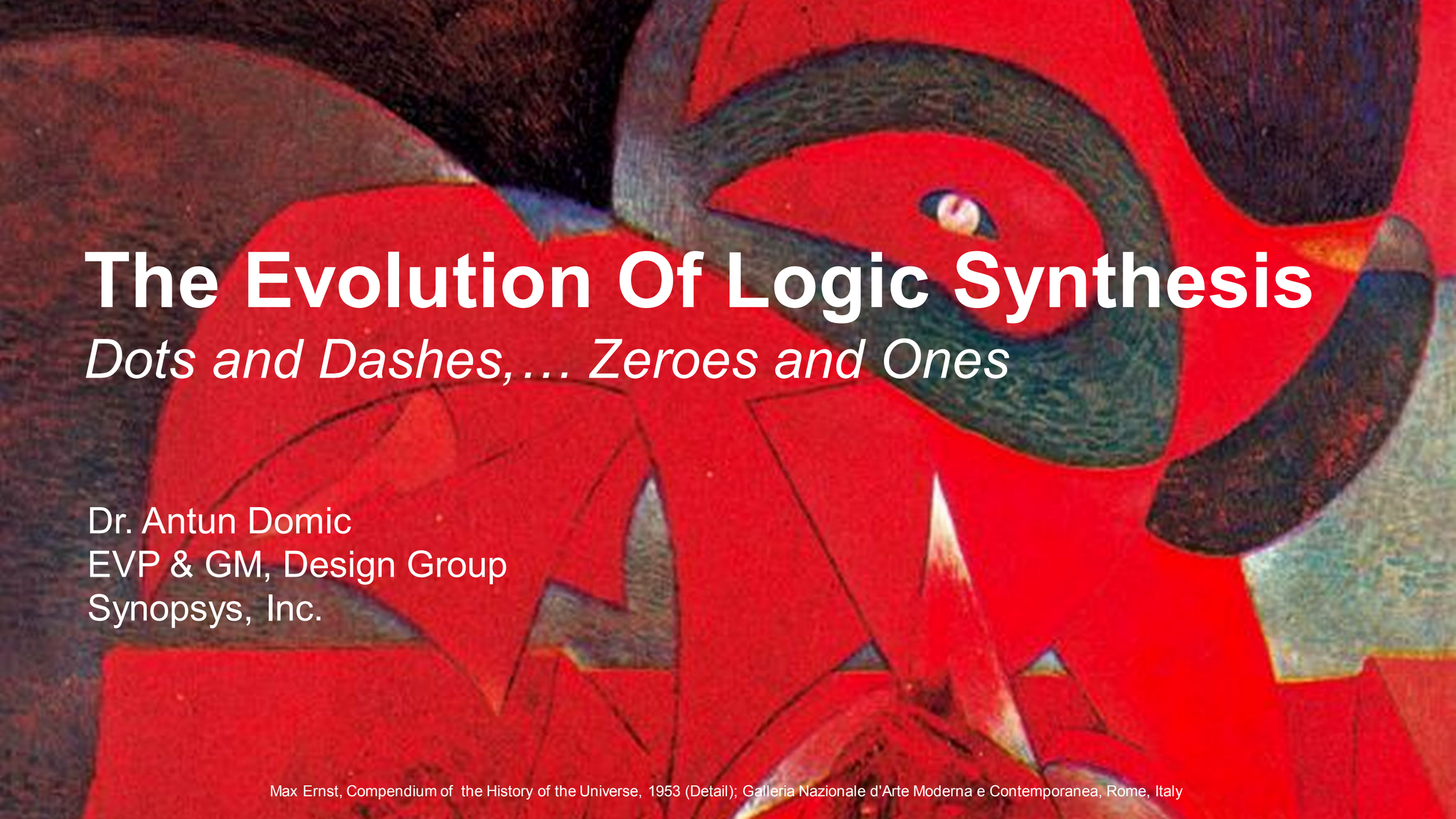


The Evolution Of Logic Synthesis

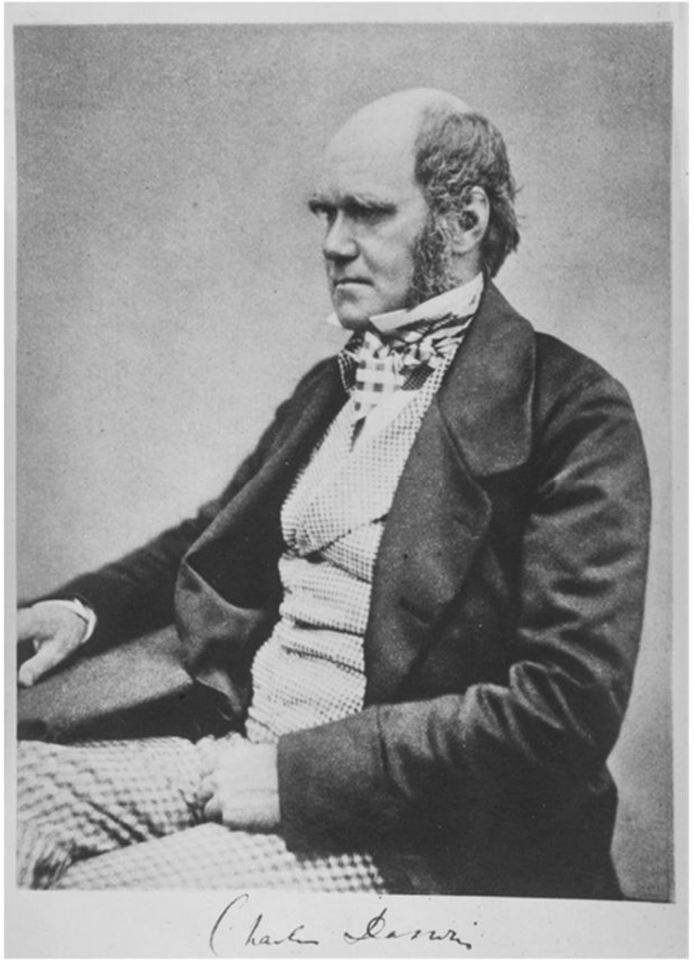
Dots and Dashes,… Zeroes and Ones

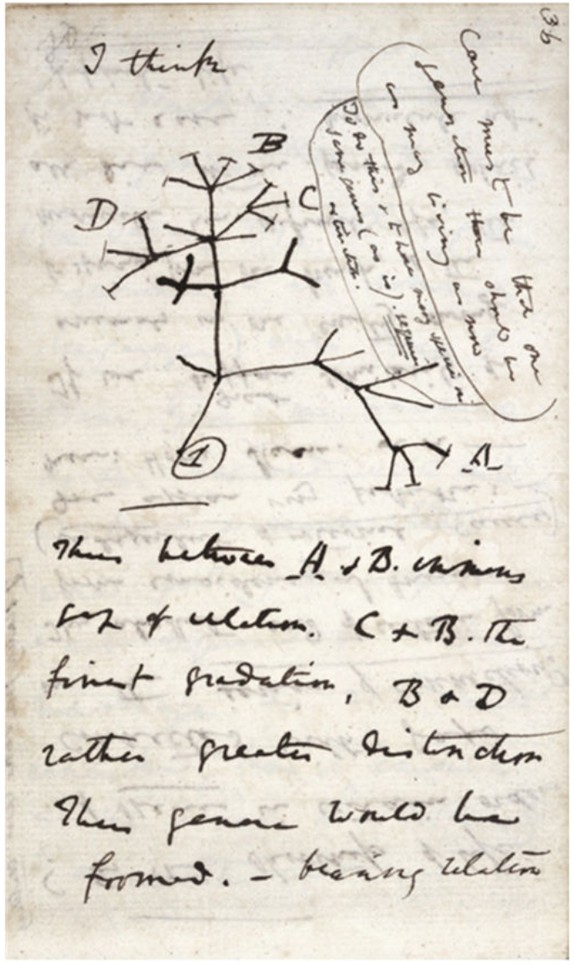
Dr. Antun Domic
EVP & GM, Design Group
Synopsys, Inc.

Max Ernst, Compendium of the History of the Universe, 1953 (Detail); Galleria Nazionale d'Arte Moderna e Contemporanea, Rome, Italy

Charles R. Darwin

"The Great Tree of Life"





Source: Charles R. Darwin, "B" Notebook, Page 36, Cambridge University Library, UK

SYNOPSYS®

Smartphones & Logic Synthesis

Have More in Common than You May Think !

Smartphones

Logic Synthesis

Claude E. Shannon
"A Symbolic Analysis of Relay and Switching Circuits"

Logic Compiler, ca. 1986
Optimal Solutions, Inc. a.k.a. Synopsys, Inc.

Power Compiler, 1997
Convergence of Logic & [Dynamic] Power Synthesis

Physical Compiler, 1999
Convergence of Synthesis & Implementation

Design Compiler, 2001-2015
The Evolution of Synthesis !

	Area	Timing	Power		Runtime
			Dynamic	Static	
2001	100	100	100	100	100
2005	78	91	82	78	20
2010	65	74	66	49	1.6(1)
2015	53	63	66	38	0.6(1)

(1) Design Compiler Multicore

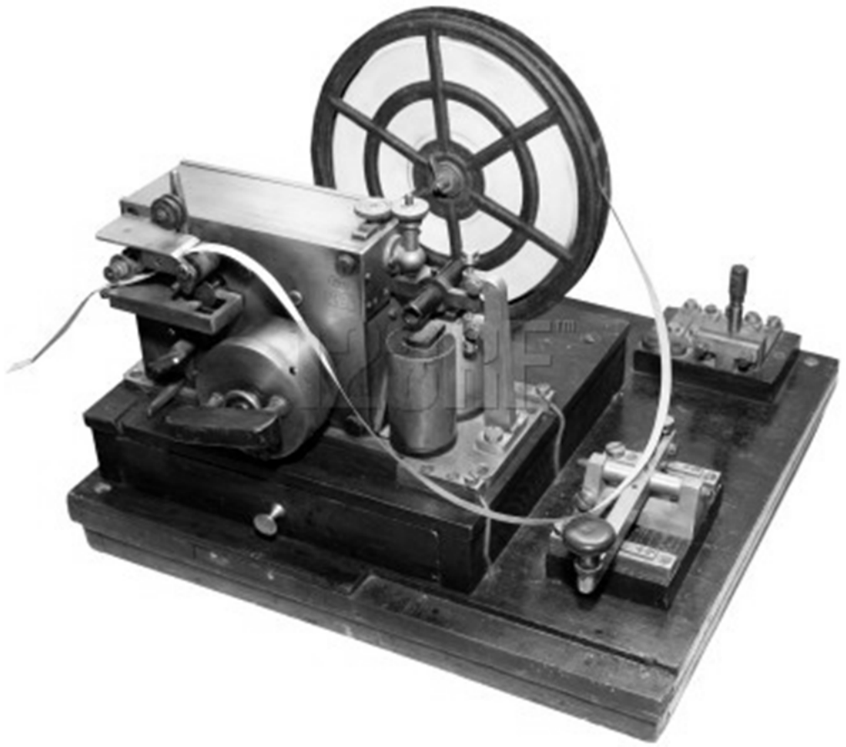
© 2015 Synopsys, Inc. 23 Source: Synopsys Research 2015 SYNOPSYS

SYNOPSYS®

Once Upon A Time…

Samuel F.B. Morse's Telegraph and George Boole's Algebra

Dots and Dashes…



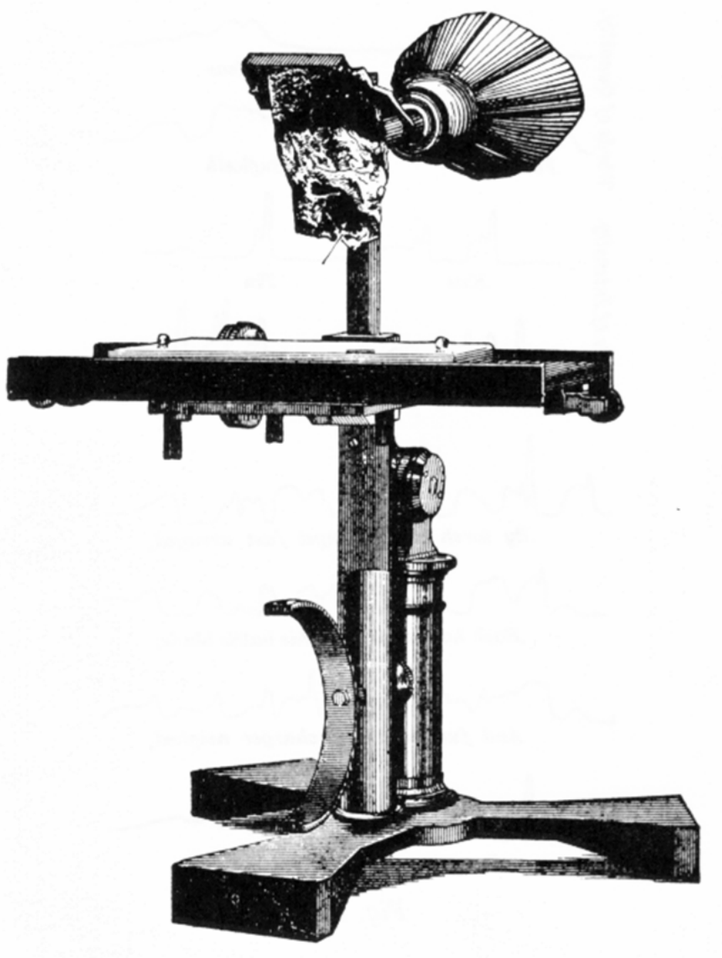
…Zeros and Ones

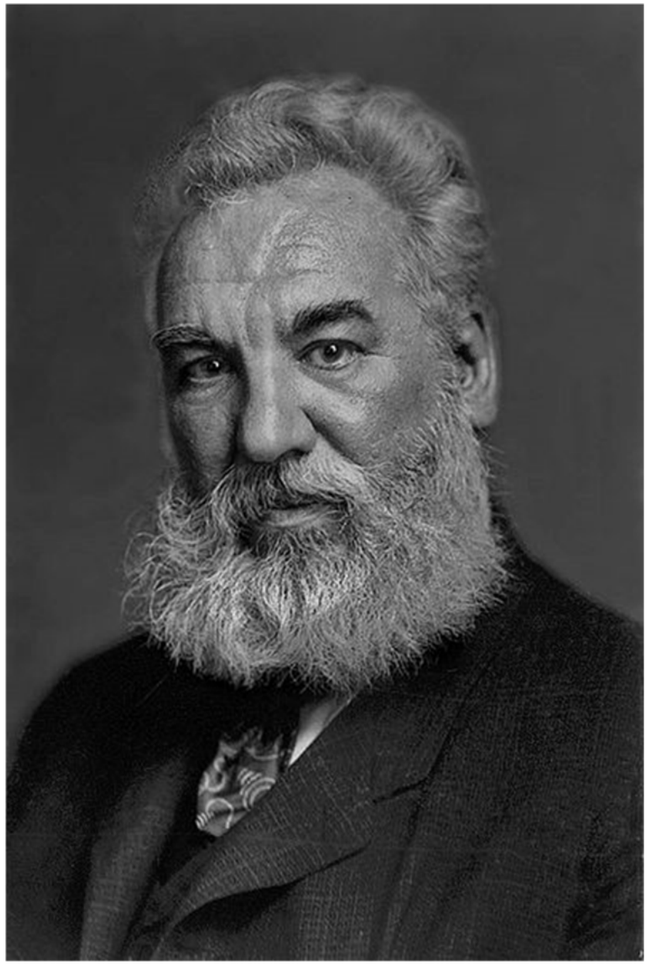
"An Investigation of the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities", 1854

$$f(A,B,C,D) =$$
$$= (\overline{A}BC\overline{D})+(A\overline{BCD})+(A\overline{BC}D)+(A\overline{B}C\overline{D})+$$
$$+(A\overline{B}CD)+(AB\overline{CD})+(AB\overline{C}D)+(ABC\overline{D})$$

Once Upon A Time…

A. Graham Bell's Telephone – the "Speaking Telegraph"





Source: A History of Engineering & Science in the Bell System, Bell Labs, 1975

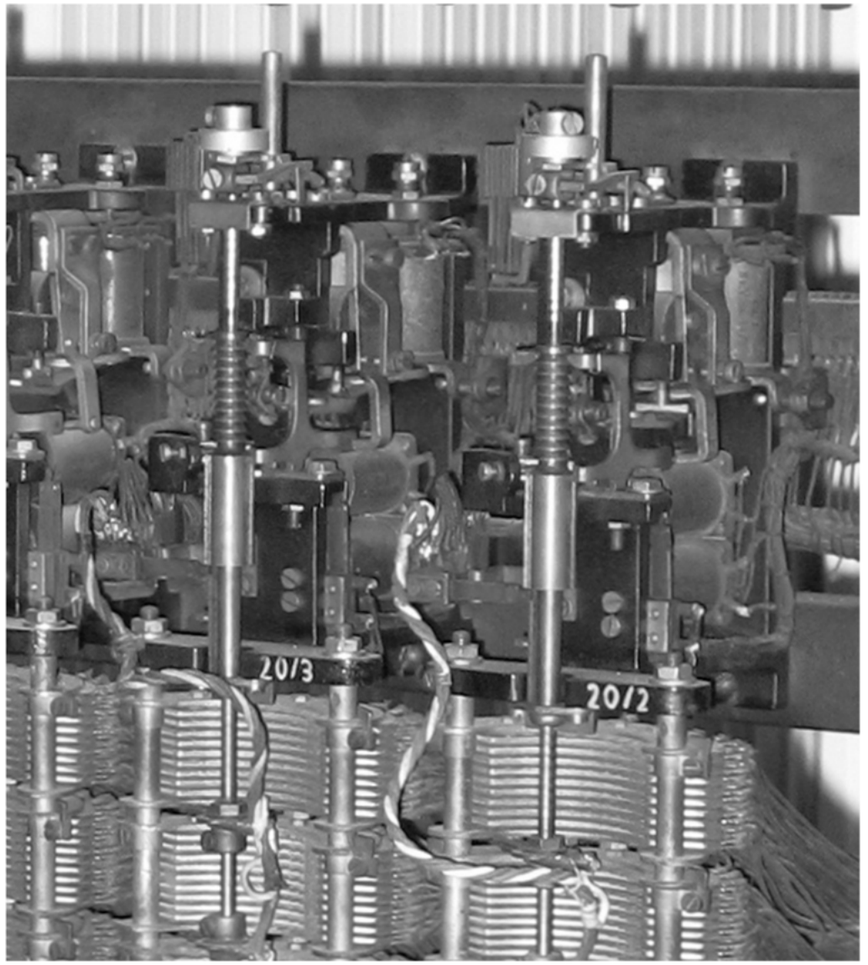
Once Upon A Time…

Bell Labs Radio-Telephone – the "Mobile Phone"

Source: Bell Labs, Radio-Telephone, 1924

Once Upon A Time…

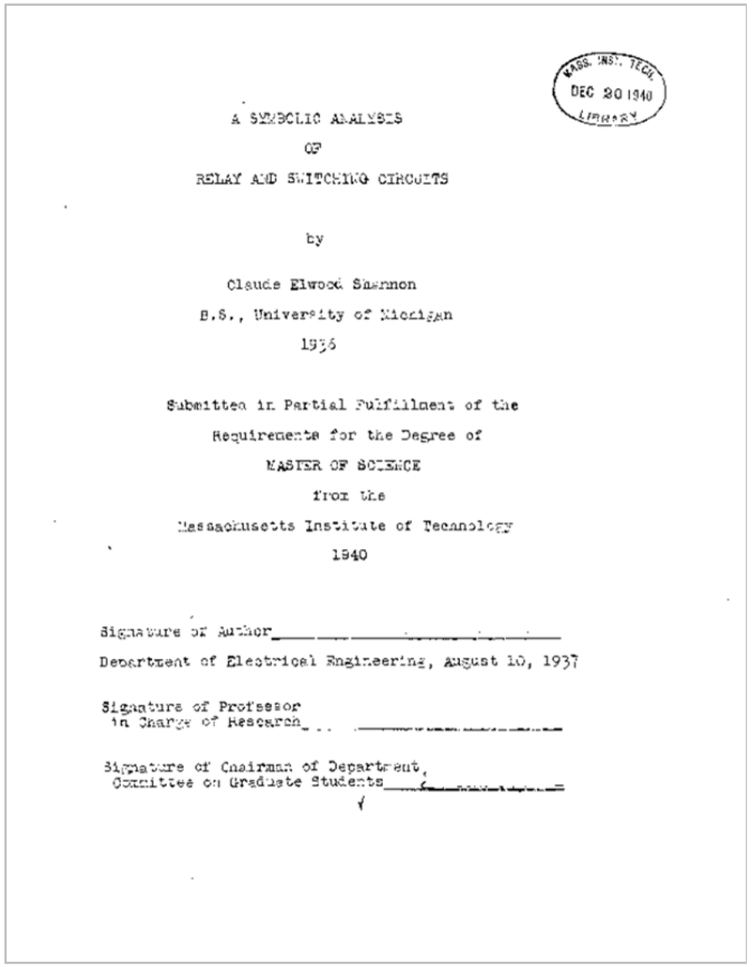
Almon B. Strowger ‘s Rotary Dial Telephone, and…
Electromechanical Telephone Exchange !



Source: Western Electric, Model 202; Strowger Switch, Fernmeldemuseum, Aachen, Germany

Claude E. Shannon

"A Symbolic Analysis of Relay and Switching Circuits"
Master's Thesis, MIT, 1937



MASS. INST. TECH. DEC 20 1940 Library

A SYMBOLIC ANALYSIS
OF
RELAY AND SWITCHING CIRCUITS

by

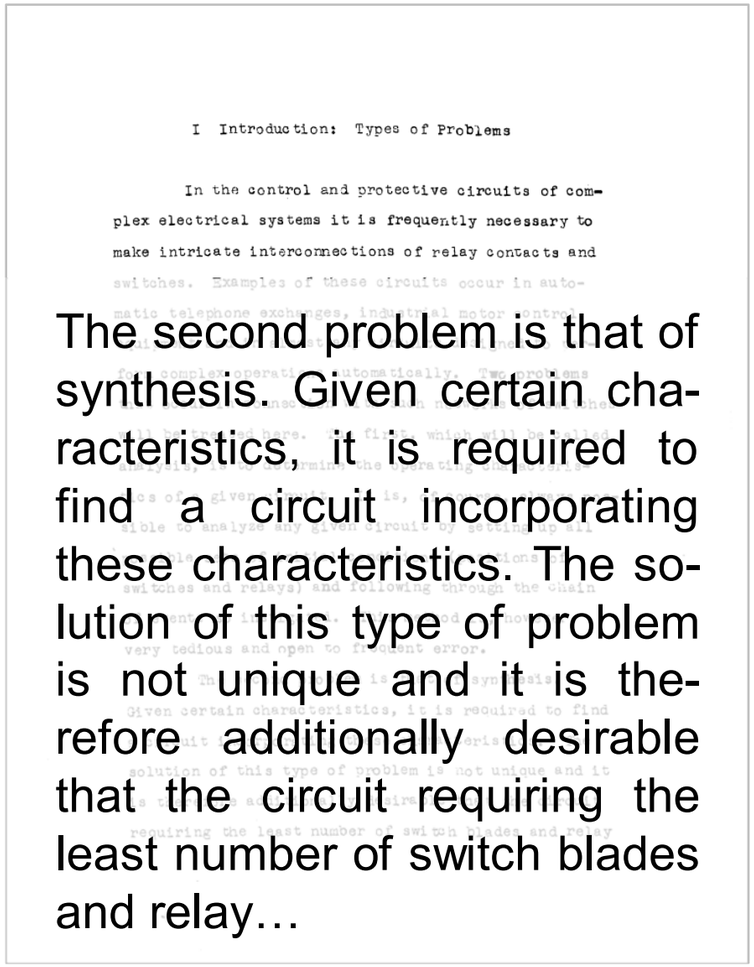
Claude Elwood Shannon
B.S., University of Michigan
1936

Submitted in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
from the
Massachusetts Institute of Technology
1940

Signature of Author_____
Department of Electrical Engineering, August 10, 1937

Signature of Professor
in Charge of Research_____

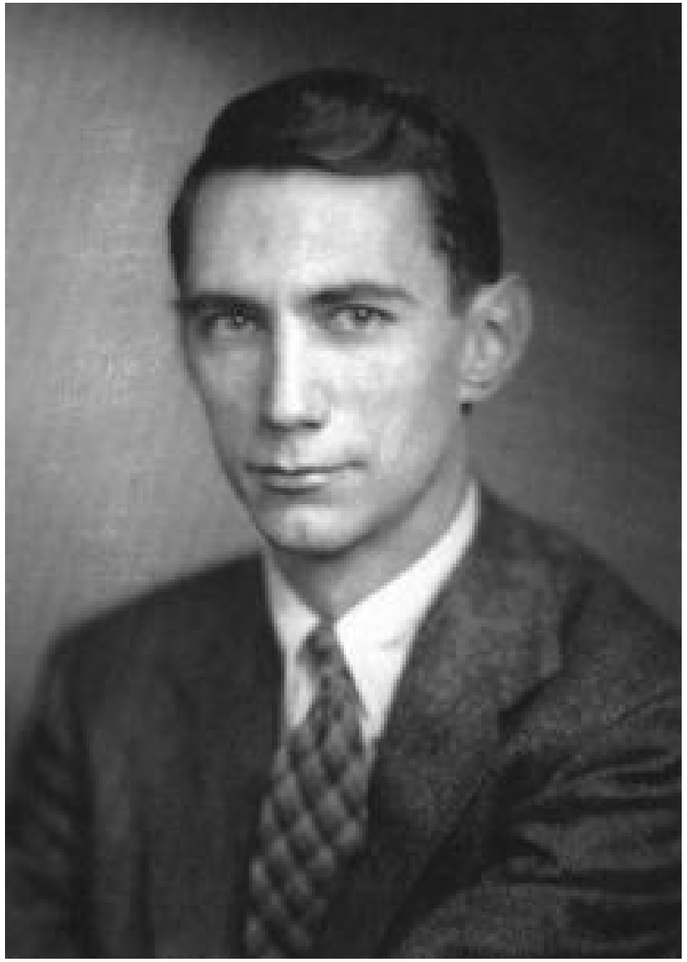
Signature of Chairman of Department,
Committee on Graduate Students_____



I Introduction: Types of Problems

In the control and protective circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. Examples of these circuits occur in automatic telephone exchanges, industrial motor control

The second problem is that of synthesis. Given certain characteristics, it is required to find a circuit incorporating these characteristics. The solution of this type of problem is not unique and it is therefore additionally desirable that the circuit requiring the least number of switch blades and relay…



Source: MIT

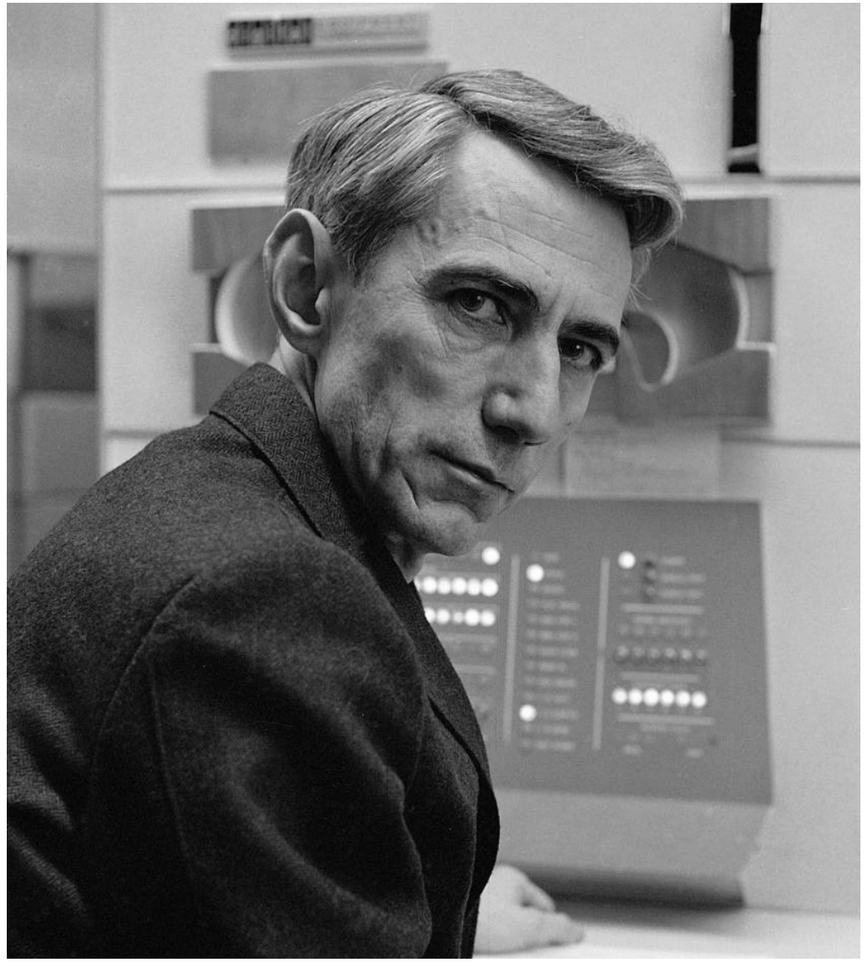
The Sampling Theorem

a.k.a. Nyquist-Shannon-Kotelnikov Theorem

The Foundation of Digital Communications

If a function x(t) contains no frequencies higher than B Hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart.

$$f\left(\frac{n}{2W}\right) = \frac{1}{2\pi}\int_{-2\pi W}^{2\pi W} F(\omega)e^{i\omega\frac{n}{2W}}\,d\omega.$$

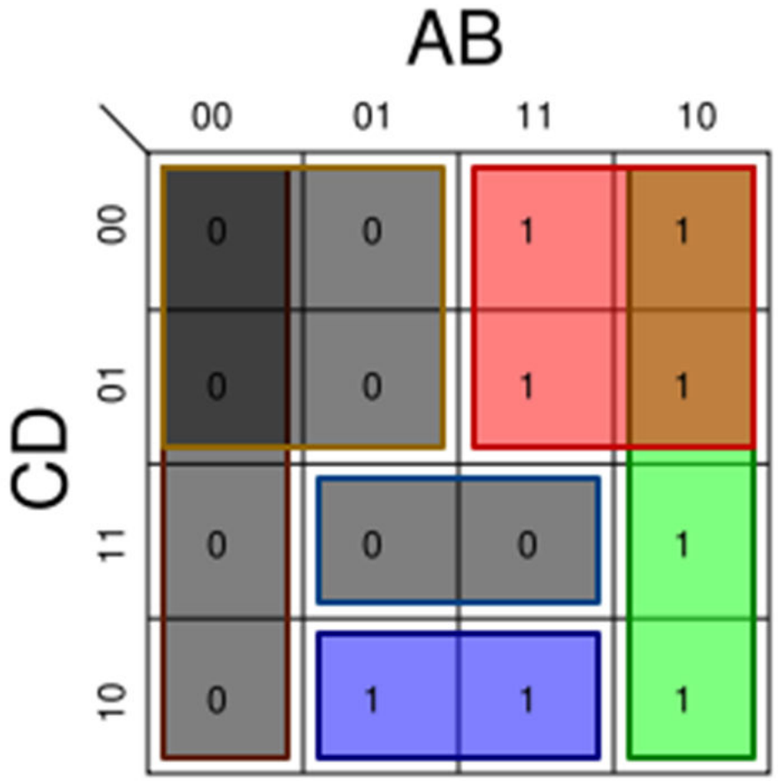


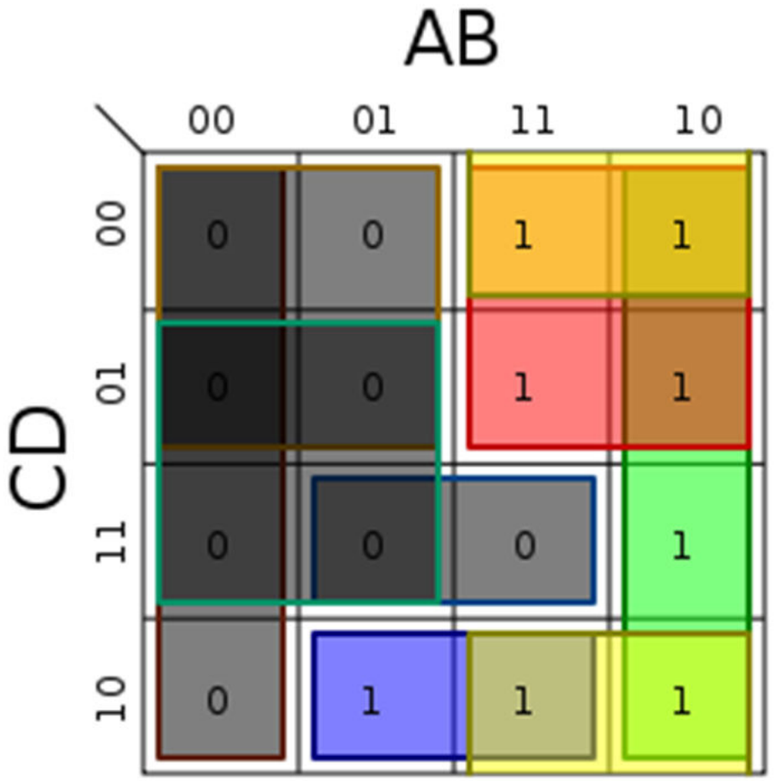
Source: LIFE, 1963

SYNOPSYS®

Maurice Karnaugh

"The Map Method for Synthesis of Combinational Logic Circuits", Bell Labs, 1953

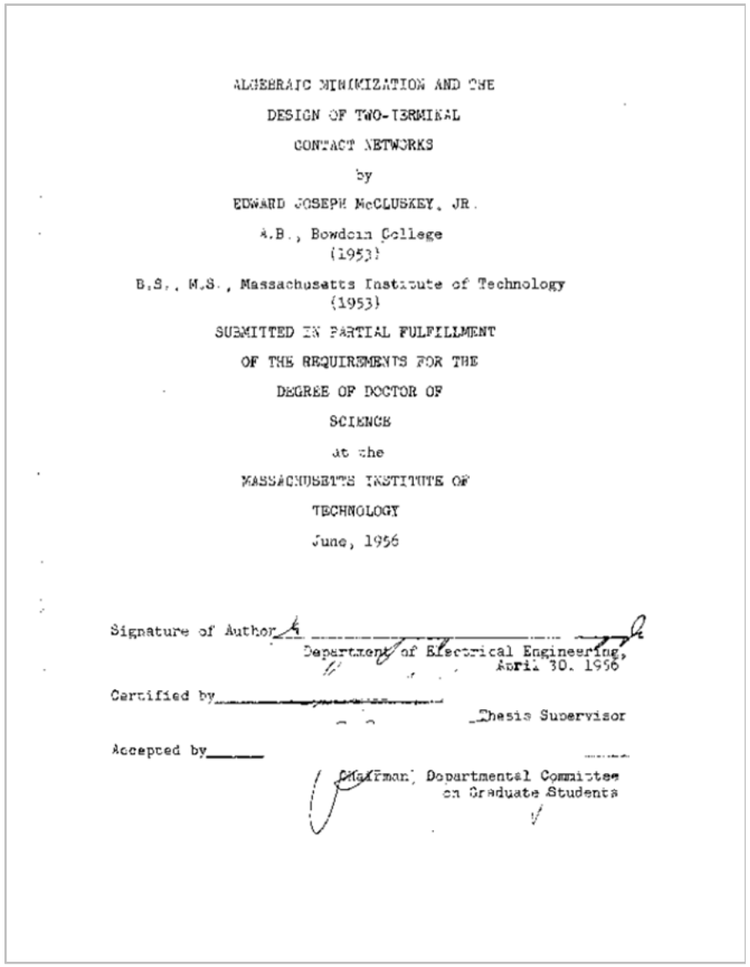




SYNOPSYS®

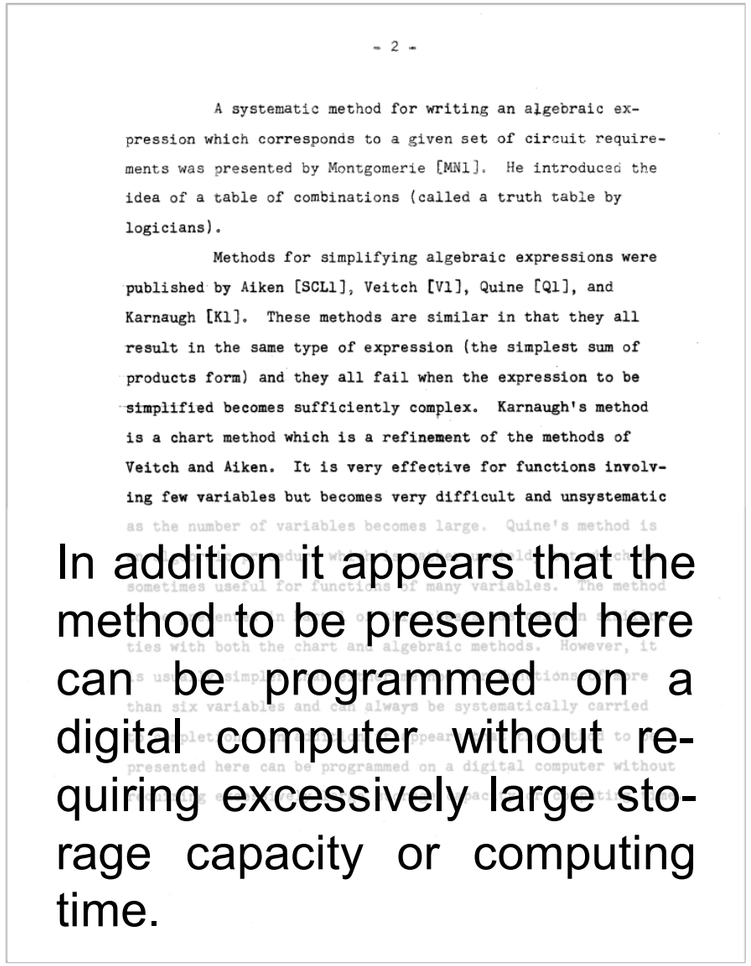
Edward J. McCluskey

"Algebraic Minimization and the Design of Two-Terminal Contact Networks"
Ph.D. Thesis, MIT, 1956



ALGEBRAIC MINIMIZATION AND THE
DESIGN OF TWO-TERMINAL
CONTACT NETWORKS
by
EDWARD JOSEPH McCLUSKEY, JR.
A.B., Bowdoin College
(1953)
B.S., M.S., Massachusetts Institute of Technology
(1953)
SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
SCIENCE
at the
MASSACHUSETTS INSTITUTE OF
TECHNOLOGY
June, 1956

Signature of Author _____
Department of Electrical Engineering,
April 30, 1956
Certified by _____
Thesis Supervisor
Accepted by _____
Chairman, Departmental Committee
on Graduate Students

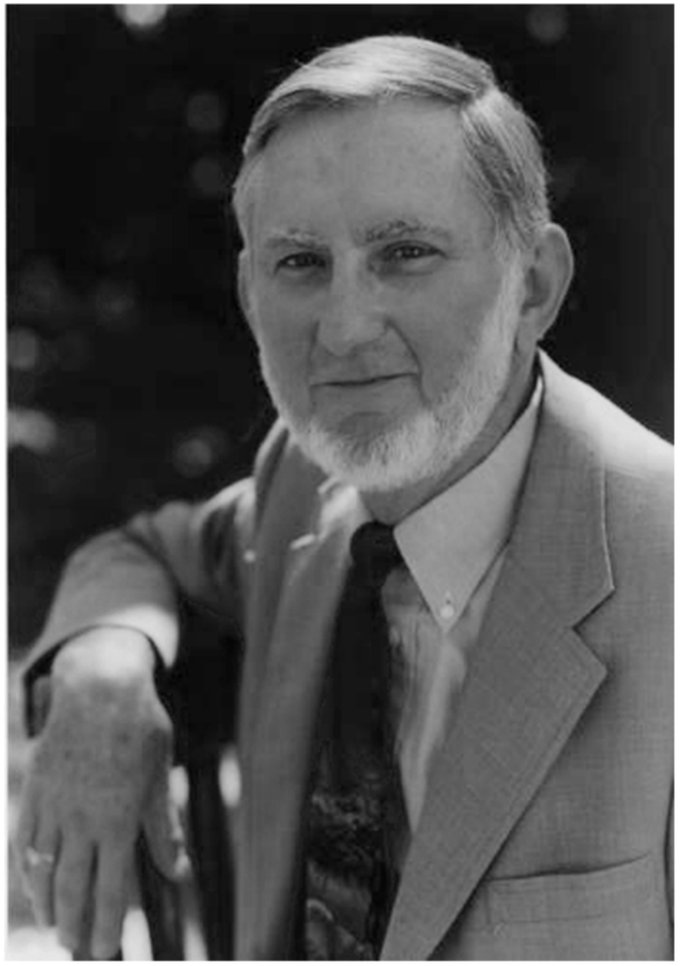


- 2 -

A systematic method for writing an algebraic expression which corresponds to a given set of circuit requirements was presented by Montgomerie [MN1]. He introduced the idea of a table of combinations (called a truth table by logicians).

Methods for simplifying algebraic expressions were published by Aiken [SCL1], Veitch [V1], Quine [Q1], and Karnaugh [K1]. These methods are similar in that they all result in the same type of expression (the simplest sum of products form) and they all fail when the expression to be simplified becomes sufficiently complex. Karnaugh's method is a chart method which is a refinement of the methods of Veitch and Aiken. It is very effective for functions involving few variables but becomes very difficult and unsystematic as the number of variables becomes large. Quine's method is

In addition it appears that the method to be presented here can be programmed on a digital computer without requiring excessively large storage capacity or computing time.

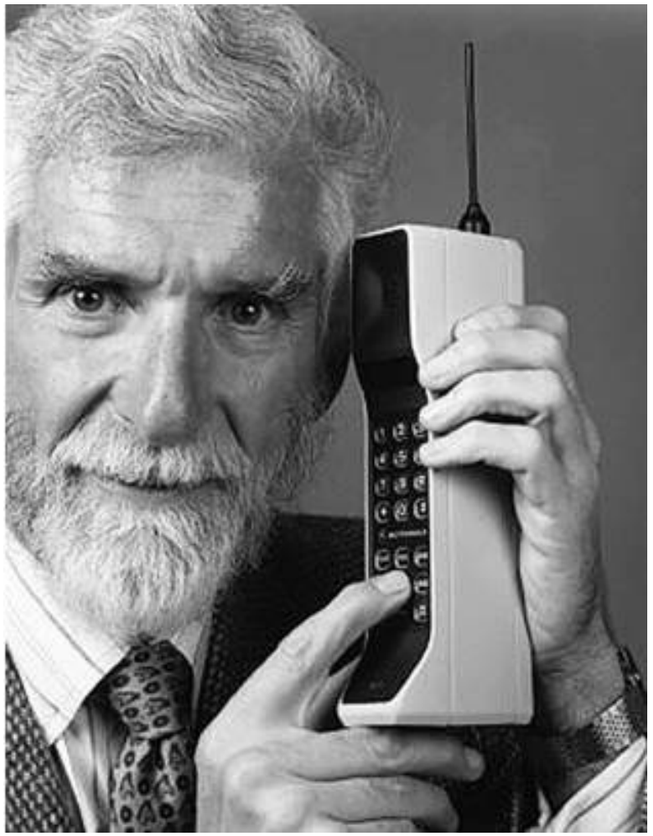


Source: MIT

…Multi-Level Minimization,…

IBM, UCB, and University of Colorado at Boulder

Motorola DynaTAC



***John Darringer, William H. Joyner, and Louise H. Trevilyan**, e.g. "Logic Synthesis Through Local Transformations", IBM, 1981*

***Robert K. Brayton, Gary D. Hachtel, A. Richard Newton, and Alberto L. Sangiovanni-Vincentelli**, e.g. "Logic Minimization Algorithms for VLSI Synthesis", UCB, 1984*

Source: Motorola DynaTAC

…And, Eventually : Logic Synthesis !

General Electric, and AT&T

Nokia 100

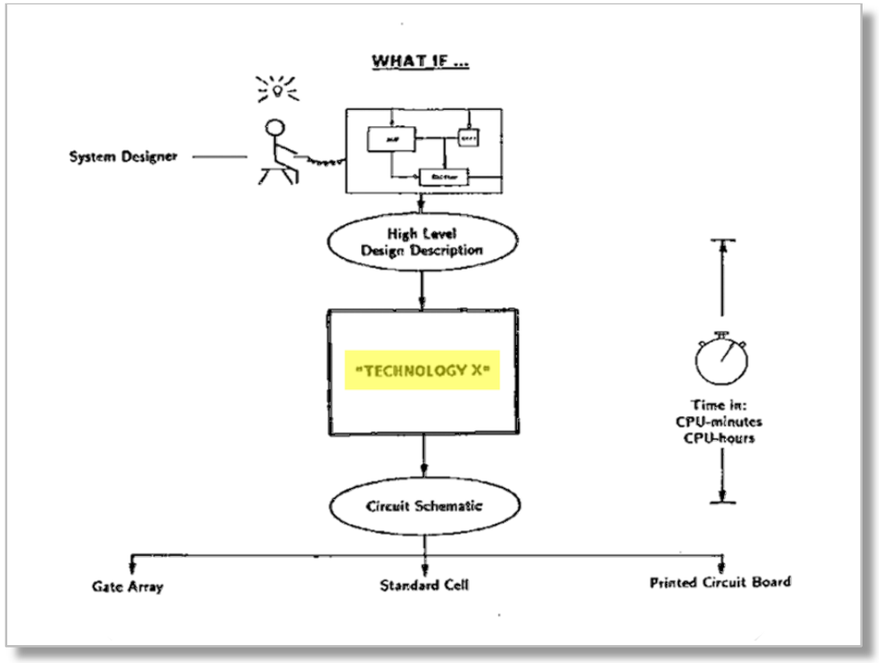
***David Gregory, Karen Bartlett, Aart J. de Geus, Gary D. Hachtel**, e.g. "SOCRATES: a System for Automatically Synthesizing and Optimizing Combinational Logic", GE Calma, 1986*

***Kurt Keutzer**, e.g. "DAGON: Technology Binding and Local Optimization by DAG Matching", AT&T, 1988*

Source: Nokia 100

Logic Compiler, ca. 1986

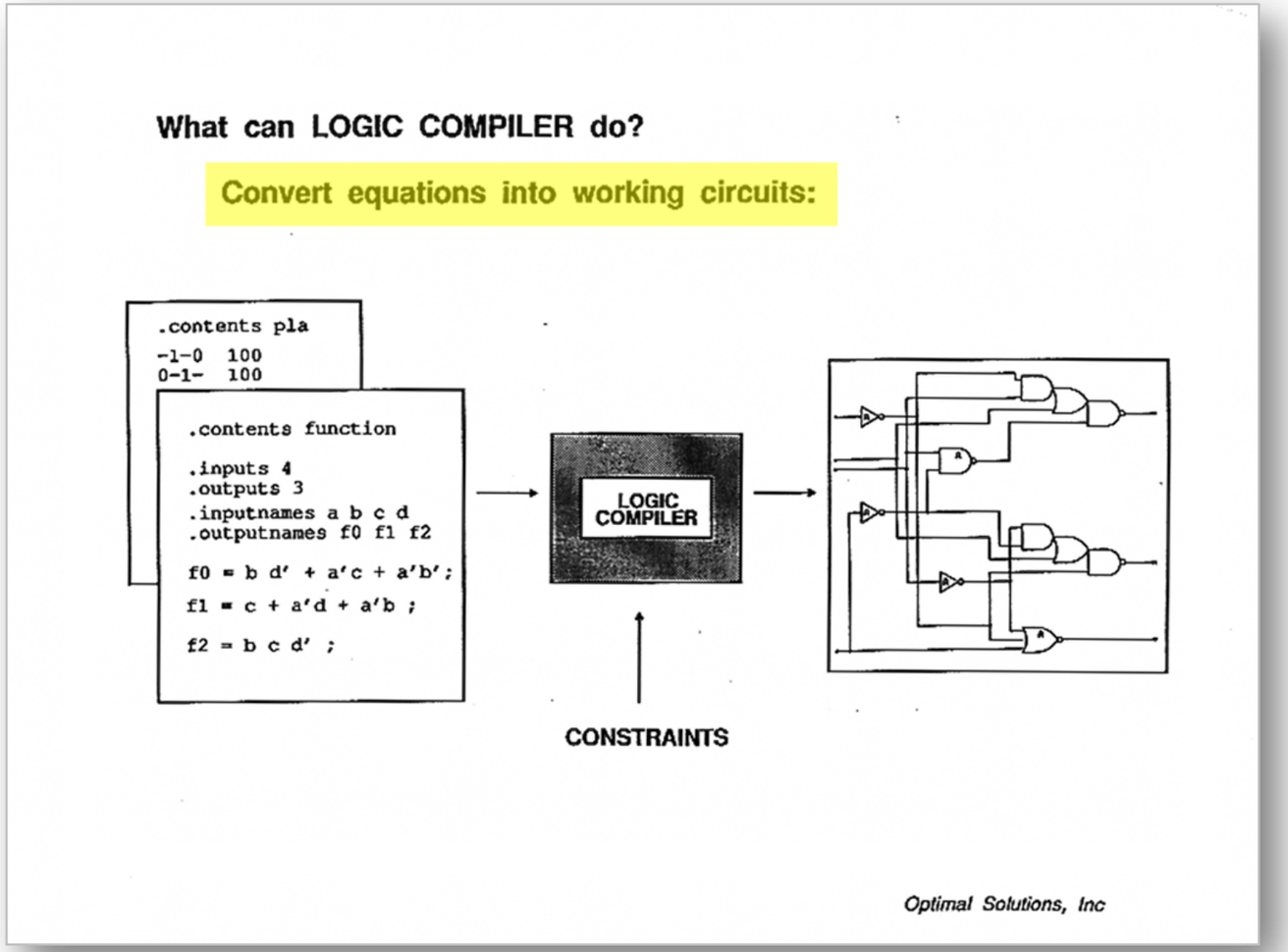
The Beginning of a New Era ! Just the Beginning…



SYNOPSYS®

Logic Compiler, ca. 1986

Optimal Solutions, Inc. a.k.a. Synopsys, Inc.



SYNOPSYS®

“MIS: A Multiple-Level Logic Optimization System”, 1987

Robert K. Brayton, Richard L. Rudell, Alberto L. Sangiovanni Vincentelli, Albert R. Wang

1062 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, VOL. CAD-6, NO. 6, NOVEMBER 1987

MIS: A Multiple-Level Logic Optimization System

ROBERT K. BRAYTON, FELLOW, IEEE, RICHARD RUDELL,
ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, IEEE, AND ALBERT R. WANG

Abstract—MIS is both an interactive and a batch-oriented multilevel logic synthesis and minimization system. MIS starts from the combinational logic extracted, typically, from a high-level description of a macrocell. It produces a multilevel set of optimized logic equations preserving the input-output behavior. The system includes both fast and slower (but more optimal) versions of algorithms for minimizing the area, and global timing optimization algorithms to meet system-level timing constraints. This paper provides an overview of the system and a description of the algorithms used. Included are some examples illustrating an input language used for specifying logic and don't-cares. Parts on an industrial chip have been re-synthesized using MIS with favorable results as compared to equivalent manual designs.

Keywords—Multiple-level logic, logic minimization, kernels, extraction, resubstitution, global phase assignment, factorization, decomposition, simplification, don't cares.

I. INTRODUCTION

OVER THE past few years, placement and routing techniques have been developed which perform reasonably well for most block-oriented design styles. However, the synthesis of the circuit itself—deciding how to partition the logic, in what form to implement specific pieces of the logic, and what layout style to use for implementation—is still largely a manual process. Often the control logic portion of the chip is the most time consuming to design, is generally on the critical path for timing, and is implemented in an inefficient way. In addition, control and dataflow logic are generally separated unnaturally, leading to inefficiencies in layout and timing. Automated synthesis of the logic, optimized for speed and area, provides one of the next major challenges for CAD.

Research done over the past 30 years has led to efficient methods for implementing combinational logic in optimal two-level form using programmable logic arrays (PLA's). However, many logic blocks are inappropriate for this kind of implementation. For example, there exist functions whose minimum two-level representation has $2^n - 1$ product terms, where n is the number of primary inputs. In addition, even if a two-level representation contains a reasonable number of terms, there are many cases in which a multilevel representation can be implemented in less area and generally as a much faster circuit. Two-level logic representations can be viewed as special cases of more general multilevel representations. Hence, a general framework for logic design should offer multilevel synthesis tools which can select between two-level or multilevel implementations depending on the area and/or speed that can be obtained. However, to be able to explore the design tradeoffs, such a system should also offer a variety of both electrical design styles (e.g., domino logic, static CMOS) and layout design styles (e.g., Weinberger arrays, gate matrix, standard cells, and gate arrays).

Optimal multilevel logic synthesis is a known difficult problem which also has been studied since the 1950's. However, much work still remains to be done in order to achieve the same level of advancement as for two-level logic synthesis. In recent years, an increasing level of research has been apparent in multilevel logic synthesis. One of the first of the modern developments is the Logic Synthesis System (LSS) [9], [10] at IBM, which has as target technology a variety of gate arrays and standard cells. The Yorktown Silicon Compiler [6], which automatically synthesizes and lays out CMOS dynamic logic, is based completely on multilevel logic and has domino CMOS logic as its primary target technology. The SOCRATES system [11] is a multilevel logic synthesis system which uses gate arrays and standard cells, and is one of the earliest to emphasize optimized timing performance. The recently developed MIS system [7], which is the subject of this paper, is targeted at both area and timing optimization and uses static CMOS complex gates or macrocells, but, similar to the logic synthesis in the Yorktown Silicon Compiler, its algorithms can easily support a variety of target technologies.

A widely accepted optimization criterion for multilevel logic synthesis 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. Considerations such as testability should also be included; however, in most current systems, testability is only considered indirectly as a side effect of a less redundant implementation.

For multilevel design, two basic methodologies have evolved: 1) ''global'' optimization, where the logic functions are ''factored'' into an optimal multilevel form with little consideration of the form of the original description (e.g., the Yorktown Silicon Compiler, part of Angel [17], SOCRATES, and FDS [13]); 2) ''peephole'' optimiza-

Manuscript received January 26, 1987; revised July 9, 1987. This work was supported in part by the SRC under Contract 82-11-008, the Defence Advanced Research Projects Agency under Contract N00039-86-R-0365, the California State Micro Program, the National Science Foundation under Subcontract ECS-8430435, and the University of Colorado.

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

IEEE Log Number 8716754.

Source: 1st A. Richard Newton Technical Impact Award in Electronic Design Automation, DAC 2009

SYNOPSYS®

…Binary Decision Diagrams (BDD)…

Randal E. Bryant ⇨ Olivier Coudert, and Jean-Christophe Madre

Logic Synthesis (and Formal Verification)

IEEE TRANSACTIONS ON COMPUTERS, VOL. C-35, NO. 8, AUGUST 1986 677

Graph-Based Algorithms for Boolean Function Manipulation

RANDAL E. BRYANT, [illegible]

Abstract—In this paper we present a new data structure for representing Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acyclic graphs in a manner similar to the representation introduced by Lee [1] and Akers [2], but with further restrictions on the ordering of decision variables in the graph. Although a function requires, in the worst case, a graph of size exponential in the number of arguments, many of the functions encountered in typical applications have a more reasonable representation. Our algorithms have time complexity proportional to the sizes of the graphs being operated on, and hence are quite efficient as long as the graphs do not grow too large. We present experimental results from applying these algorithms to problems in logic design verification that demonstrate the practicality of our approach.

Index Terms—Boolean functions, binary decision diagrams, logic design verification, symbolic manipulation.

I. Introduction

BOOLEAN Algebra forms a cornerstone of computer science and digital system design. Many problems in digital logic design and testing, artificial intelligence, and combinatorics can be expressed as a sequence of operations on Boolean functions. Such applications would benefit from efficient algorithms for representing and manipulating Boolean functions symbolically. Unfortunately, many of the tasks one would like to perform with Boolean functions, such as testing whether there exists any assignment of input variables such that a given Boolean expression evaluates to 1 (satisfiability), or two Boolean expressions denote the same function (equivalence) require solutions to NP-complete or co NP-complete problems [3]. Consequently, all known approaches to performing these operations require, in the worst case, an amount of computer time that grows exponentially with the size of the problem. This makes it difficult to compare the relative efficiencies of different approaches to representing and manipulating Boolean functions. In the worst case, all known approaches perform as poorly as the naive approach of representing functions by their truth tables and defining all of the desired operations in terms of their effect on truth table entries. In practice, by utilizing more clever representations and manipulation algorithms, we can often avoid these exponential computations.

A variety of methods have been developed for representing and manipulating Boolean functions. Those based on classical representations such as truth tables, Karnaugh maps, or canonical sum-of-products form [4] are quite impractical—*every* function of n arguments has a representation of size 2^n or more. More practical approaches utilize representations that, at least for many functions, are not of exponential size. Example representations include as a reduced sum of products [4] (or equivalently as sets of prime cubes [5]) and factored into onate functions [6]. These representations suffer from several drawbacks. First, certain common functions still require representations of exponential size. For example, the even and odd parity functions serve as worst case examples in all of these representations. Second, while a certain function may have a reasonable representation, performing a simple operation such as complementation could yield a function with an exponential representation. Finally, none of three representations are *canonical forms*, i.e., a given function may have many different representations. Consequently, testing for equivalence or satisfiability can be quite difficult.

Due to these characteristics, most paths in the sequence of operations on Boolean functions exhibit erratic behavior. They proceed at a reasonable pace, but then suddenly "blow up," either running out of storage or failing to complete an operation in a reasonable amount of time.

In this paper we present a new class of algorithms for manipulating Boolean functions represented as directed acyclic graphs. Our representation resembles the binary decision diagram notation introduced by Lee [1] and further popularized by Akers [2]. However, we place further restrictions on the ordering of decision variables in the vertices. These restrictions enable the development of algorithms for manipulating the representations in a more efficient manner.

Our representation has several advantages over previous approaches to Boolean function manipulation. First, most commonly encountered functions have a reasonable representation. For example, all symmetric functions (including even and odd parity) are represented by graphs where the number of vertices grows at most as the square of the number of arguments. Second, the performance of a program based on our algorithms when processing a sequence of operations degrades slowly, if at all. That is, the time complexity of any single operation is bounded by the product of the graph sizes for the functions being operated on. For example, complementing a function requires time proportional to the size of the function graph, while combining two functions with a binary operation (of which intersection, subtraction, and testing for

Manuscript received November 28, 1984; revised June 11, 1985. This work was supported in part by the Defense Advanced Research Projects Agency under Order 3597 and 1967.

The author is with the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 8609099.

0018-9340/86/0800-0677$01.00 © 1986 IEEE

Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions

O. Coudert and J. C. Madre
Bull Corporate Research Center
Rue Jean Jaurès
78340 Les Clayes-sous-bois, FRANCE

Abstract

Recently introduced *implicit* set manipulation techniques have made it possible to formally verify finite state machines with state graphs too large to be built. This paper shows that these techniques can also be used with success to compute and manipulate implicitly extremely large sets of prime and of essential prime implicants of incompletely specified Boolean functions. These sets are denoted by *meta-products* that are represented with binary decision diagrams. This paper describes two procedures. One is based on the standard BDD operators, and the other, more efficient, takes advantage of the structural properties of BDDs and of meta-products to handle a larger class of functions than the former one.

Introduction

We have recently introduced a technique [4] for verifying finite state machines that can deal with machines with state graphs too large to be built. The key concepts that make this verification possible are to denote subsets of $\{0,1\}^n$ with their characteristic functions, and represent these Boolean functions with binary decision diagrams (BDDs) that are a very compact graph representation of such functions [3]. Since there is no relation between the size of a set and the size of the BDD that denotes it, the computation cost of this technique is completely independent from the number of states of the machines [4].

In this paper we show that these concepts can be used with success to implement *implicit* computation procedures of the sets of prime and of essential prime implicants of incompletely specified Boolean functions. These procedures have costs that are independent of the sizes of these sets, and thus they overcome the limitations of methods based on explicit prime manipulations [2, 8, 11, 12].

This paper is divided in 6 parts. Section 2 presents the elementary concepts that will be used to solve the problem treated here. Section 3 introduces the canonical *meta-product* representation of sets of products built out of finite sets of Boolean variables. Section 4 gives the expressions that define the sets of prime and of essential prime implicants of incompletely specified Boolean functions, and shows that these equations can be evaluated with the standard BDD operators. Section 5 describes how this standard procedure can be transformed into a more efficient incremental procedure. Section 6 gives experimental results obtained with these procedures and discusses them.

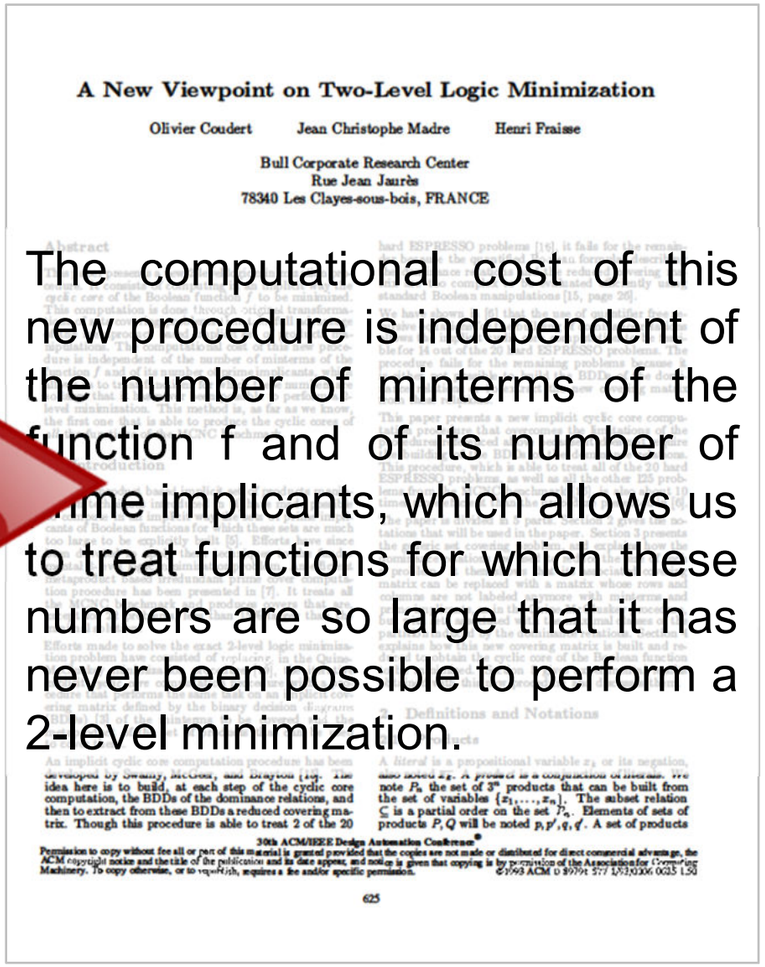
2 Definitions

A Boolean function f from $\{0,1\}^n$ into [illegible] unique subset S_f of $\{0,1\}^n$, made of [illegible] of $\{0,1\}^n$ such that $f(x) = 1$. Conversely, [illegible] of $\{0,1\}^n$ can be represented by a unique Boolean function X_S from $\{0,1\}^n$ into $\{0,1\}$, called its *characteristic function*, such that $X_S(x) = 1$ if and only if $x \in S$. Characteristic functions are a very interesting representation of Boolean sets because Boolean operators correspond with set operators [4].

We note P_n the set of products that can be built out of the set of variables $\{x_1, \ldots, x_n\}$. A *product* is a formula of the form $(l_1 \ldots l_k)$, $k \leq n$, whose literals l_j are built out of distinct variables. The product p contains the product p' ($p \geq p'$) if and only if $S_p \supseteq S_{p'}$. The relation "$\geq$" is a partial order on P_n.

An *incompletely specified* Boolean function f_c, also called a function with a *care set*, is defined by a couple (C, f), where f is a function from $\{0,1\}^n$ into $\{0,1\}$, and the care set C is a subset of $\{0,1\}^n$. This function is defined by $f_c(x) = f(x)$ if $x \in C$, and $f_c(x) = *$ if $x \notin C$, where the symbol "*" can be either 0 or 1 [2].

The product p of P_n is a *cube* of the function $f_c = (C, f)$ if and only if the set $S_p \cap C$ is not empty, and for any element of this set, the function f evaluates to 1. The cube p of f_c is a *prime implicant* or *prime* of f_c if and only if p is a maximal element of the set of cubes of f_c with respect to the partial order "$\geq$". Finally the



A New Viewpoint on Two-Level Logic Minimization

Olivier Coudert Jean Christophe Madre Henri Fraisse

Bull Corporate Research Center
Rue Jean Jaurès
78340 Les Clayes-sous-bois, FRANCE

The computational cost of this new procedure is independent of the number of minterms of the function f and of its number of prime implicants, which allows us to treat functions for which these numbers are so large that it has never been possible to perform a 2-level minimization.

[illegible] idea here is to build, at each step of the cyclic core computation, the BDDs of the dominance relations, and then to extract from these BDDs a reduced covering matrix. Though this procedure is able to treat 2 of the 20

[illegible] note P_n the set of 3^n products that can be built from the set of variables $\{x_1, \ldots, x_n\}$. The subset relation $\subseteq$ is a partial order on the set P_n. Elements of sets of products P, Q will be noted p, p', q, q'. A set of products

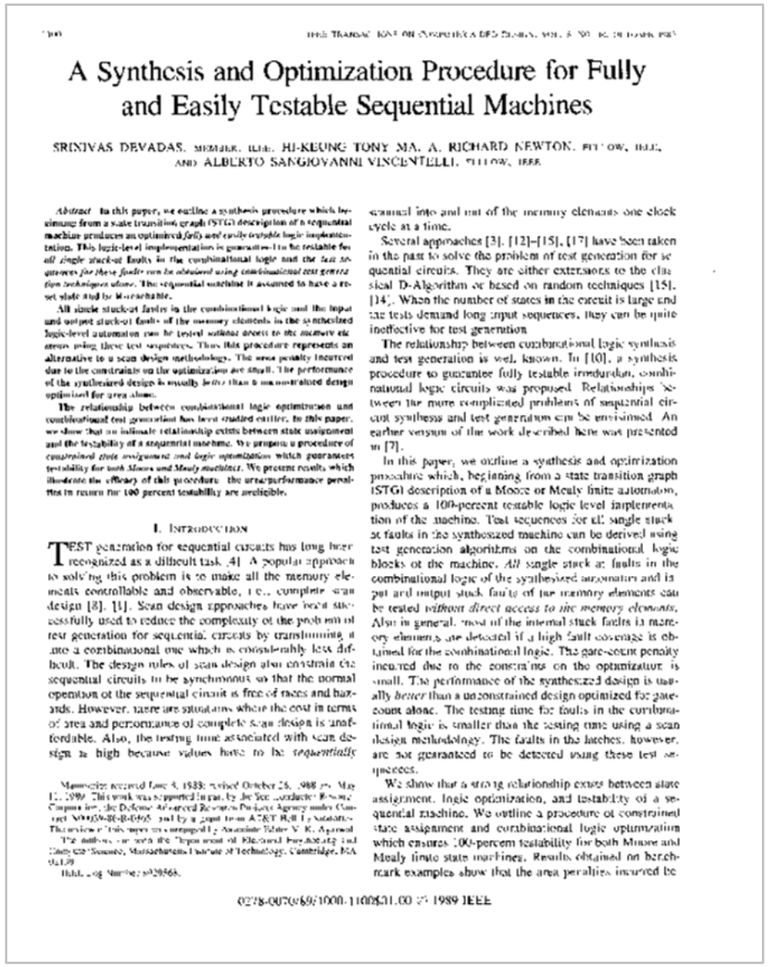
30th ACM/IEEE Design Automation Conference®

SYNOPSYS®

Jackson Pollock, Convergence, 1952, Albright-Knox Art Gallery, Buffalo, NY

Convergence Of Logic And Test Synthesis

Srinivas Devadas, Hi-Keung Tony Ma,
A. Richard Newton, Alberto L. Sangiovanni-Vincentelli



A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines

SRINIVAS DEVADAS, MEMBER, IEEE, HI-KEUNG TONY MA, A. RICHARD NEWTON, FELLOW, IEEE, AND ALBERTO SANGIOVANNI VINCENTELLI, FELLOW, IEEE

Abstract—In this paper, we outline a synthesis procedure which beginning from a state transition graph (STG) description of a sequential machine produces an optimized, *fully and easily testable* logic implementation. This logic-level implementation is guaranteed to be testable for *all single stuck-at* faults in the combinational logic and the *test sequences for these faults can be obtained using combinational test generation techniques alone*. The sequential machine is assumed to have a reset state and be initializable.

All single stuck-at faults in the combinational logic and the input and output stuck-at faults of the memory elements in the synthesized logic-level automaton can be tested without access to the memory elements using these test sequences. Thus this procedure represents an alternative to a scan design methodology. The area penalty incurred due to the constraints on the optimization are small. The performance of the synthesized design is usually *better* than a unconstrained design optimized for area alone.

The relationship between combinational logic optimization and combinational test generation has been studied earlier. In this paper, we show that an intimate relationship exists between state assignment and the testability of a sequential machine. We propose a procedure of *constrained state assignment and logic optimization* which guarantees testability for both *Moore and Mealy machines*. We present results, which illustrate the efficacy of this procedure: the area/performance penalties in return for 100 percent testability are available.

I. INTRODUCTION

TEST generation for sequential circuits has long been recognized as a difficult task. [4] A popular approach to solving this problem is to make all the memory elements controllable and observable, i.e., complete scan design [8], [1]. Scan design approaches have been successfully used to reduce the complexity of the problem of test generation for sequential circuits by transforming it into a combinational one which is considerably less difficult. The design rules of scan design also constrain the sequential circuits to be synchronous so that the normal operation of the sequential circuit is free of races and hazards. However, there are situations where the cost in terms of area and performance of complete scan design is unaffordable. Also, the testing time associated with scan design is high because values have to be *sequentially* scanned into and out of the memory elements one clock cycle at a time.

Several approaches [3], [12]–[15], [17] have been taken in the past to solve the problem of test generation for sequential circuits. They are either extensions to the classical D-Algorithm or based on random techniques [15], [14]. When the number of states in the circuit is large and the tests demand long input sequences, they can be quite ineffective for test generation.

The relationship between combinational logic synthesis and test generation is well known. In [10], a synthesis procedure to guarantee fully testable irredundant combinational logic circuits was proposed. Relationships between the more complicated problems of sequential circuit synthesis and test generation can be envisioned. An earlier version of the work described here was presented in [7].

In this paper, we outline a synthesis and optimization procedure which, beginning from a state transition graph (STG) description of a Moore or Mealy finite automaton, produces a 100-percent testable logic level implementation of the machine. Test sequences for all single stuck-at faults in the synthesized machine can be derived using test generation algorithms on the combinational logic blocks of the machine. *All* single stuck-at faults in the combinational logic of the synthesized automaton and input and output stuck faults of the memory elements can be tested *without direct access to the memory elements*. Also in general, most of the internal stuck faults in memory elements are detected if a high fault coverage is obtained for the combinational logic. The gate-count penalty incurred due to the constraints on the optimization is small. The performance of the synthesized design is usually *better* than a unconstrained design optimized for gate-count alone. The testing time for faults in the combinational logic is smaller than the testing time using a scan design methodology. The faults in the latches, however, are not guaranteed to be detected using these test sequences.

We show that a strong relationship exists between state assignment, logic optimization, and testability of a sequential machine. We outline a procedure of constrained state assignment and combinational logic optimization which ensures 100-percent testability for both Moore and Mealy finite state machines. Results obtained on benchmark examples show that the area penalties incurred be

Manuscript received June 4, 1988; revised October 26, 1988 and May 12, 1989. This work was supported in part by the Semiconductor Research Corporation, the Defense Advanced Research Projects Agency under Contract N00039-87-C-0182, and by a grant from AT&T Bell Laboratories. This paper was recommended by Associate Editor V. K. Agarwal.

The authors are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number: 8928568.

0278-0070/89/1000-1100$01.00 © 1989 IEEE

FOR IMMEDIATE RELEASE

MOMENTUM BUILDS FOR SYNOPSYS TEST SYNTHESIS

— Test Compiler orders surpass $2 million since announcing the product —

"Our goal was to solve the problem of developing highly testable designs with minimum penalties in chip speed and size," **said Aart de Geus, co-founder and senior vice-president of marketing at Synopsys.**

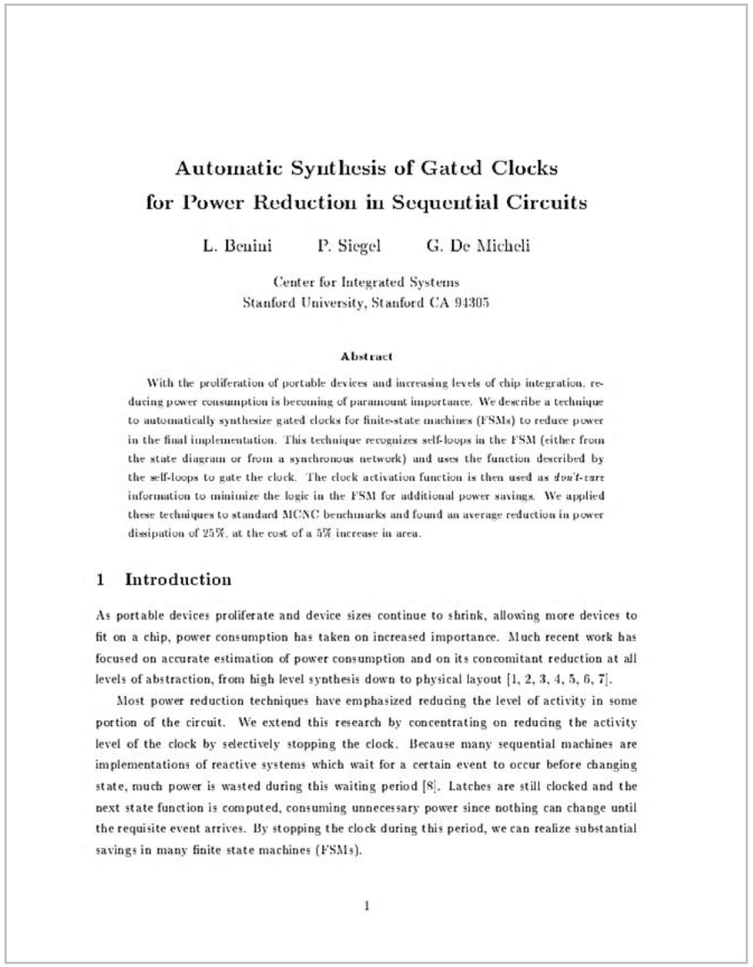
design for speed and size with the test structures in place. As a result, designers are able to hand over a highly testable design to the manufacturing test engineer and help reduce the risk of undetected manufacturing defects.

Source: Nokia 1011

SYNOPSYS®

Convergence Of Logic And Power Synthesis

L. Benini, P. Siegel, G. De Micheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits", Stanford University, 1994



Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits

L. Benini P. Siegel G. De Micheli

Center for Integrated Systems
Stanford University, Stanford CA 94305

Abstract

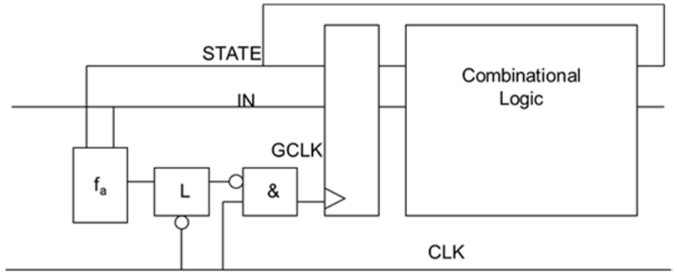
With the proliferation of portable devices and increasing levels of chip integration, reducing power consumption is becoming of paramount importance. We describe a technique to automatically synthesize gated clocks for finite-state machines (FSMs) to reduce power in the final implementation. This technique recognizes self-loops in the FSM (either from the state diagram or from a synchronous network) and uses the function described by the self-loops to gate the clock. The clock activation function is then used as *don't-care* information to minimize the logic in the FSM for additional power savings. We applied these techniques to standard MCNC benchmarks and found an average reduction in power dissipation of 25%, at the cost of a 5% increase in area.

1 Introduction

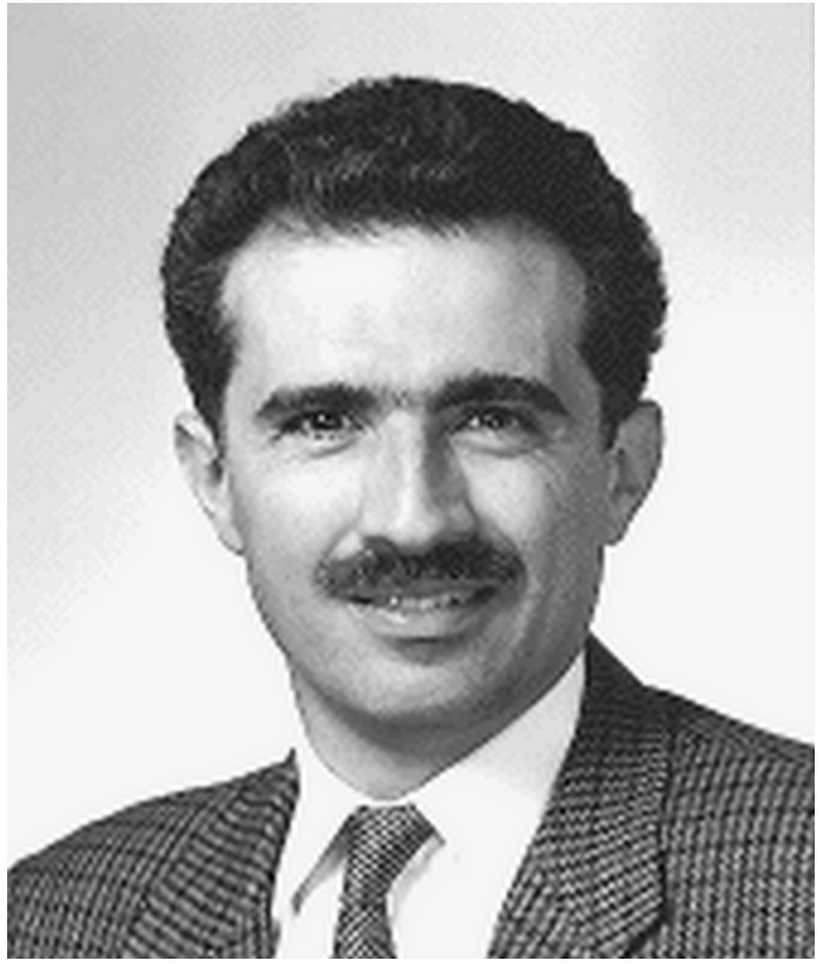
As portable devices proliferate and device sizes continue to shrink, allowing more devices to fit on a chip, power consumption has taken on increased importance. Much recent work has focused on accurate estimation of power consumption and on its concomitant reduction at all levels of abstraction, from high level synthesis down to physical layout [1, 2, 3, 4, 5, 6, 7].

Most power reduction techniques have emphasized reducing the level of activity in some portion of the circuit. We extend this research by concentrating on reducing the activity level of the clock by selectively stopping the clock. Because many sequential machines are implementations of reactive systems which wait for a certain event to occur before changing state, much power is wasted during this waiting period [8]. Latches are still clocked and the next state function is computed, consuming unnecessary power since nothing can change until the requisite event arrives. By stopping the clock during this period, we can realize substantial savings in many finite state machines (FSMs).

1



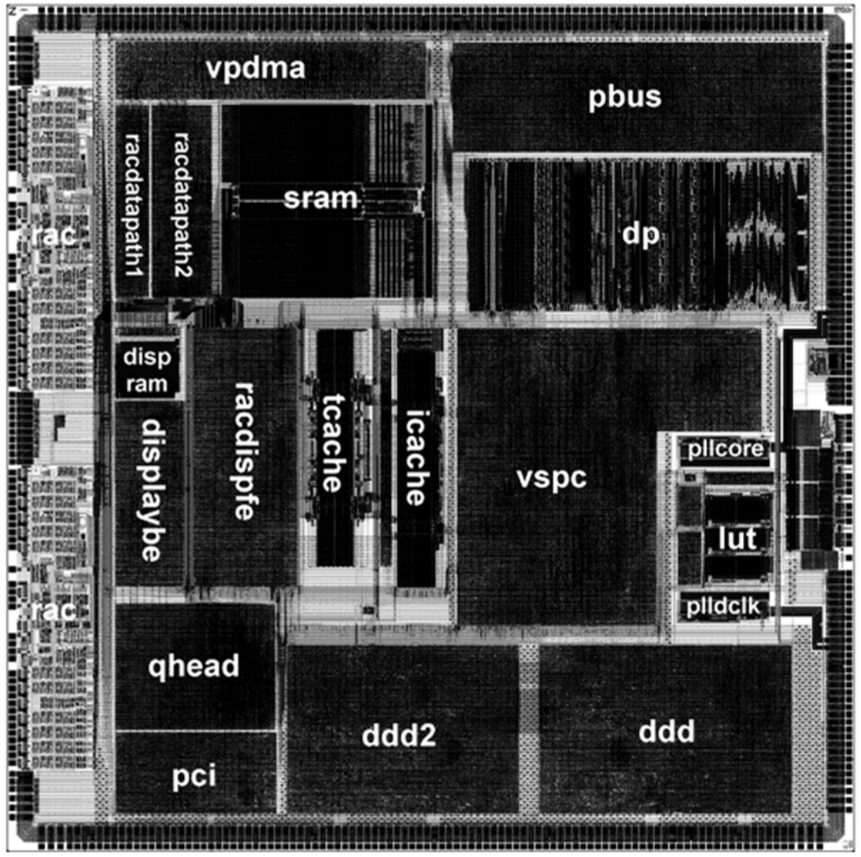
Circuit	Original Size	Original Power	Gated Size	Gated Power	Power Reduction (%)
ex	128	51μW	118	27μW	48%
bbsse	966	212μW	1002	190μW	11%
bbara	348	328μW	390	127μW	61%
bbtas	178	62μW	188	50μW	19%
sse	912	175μW	946	150μW	15%
s386	856	179μW	882	140μW	21%
cse	1320	125μW	1406	70μW	44%
dk14	758	212μW	852	211μW	1%
s27	146	60μW	178	54μW	10%
mc	182	73μW	225	61μW	17%
sand1	2220	265μW	2108	180μW	32%



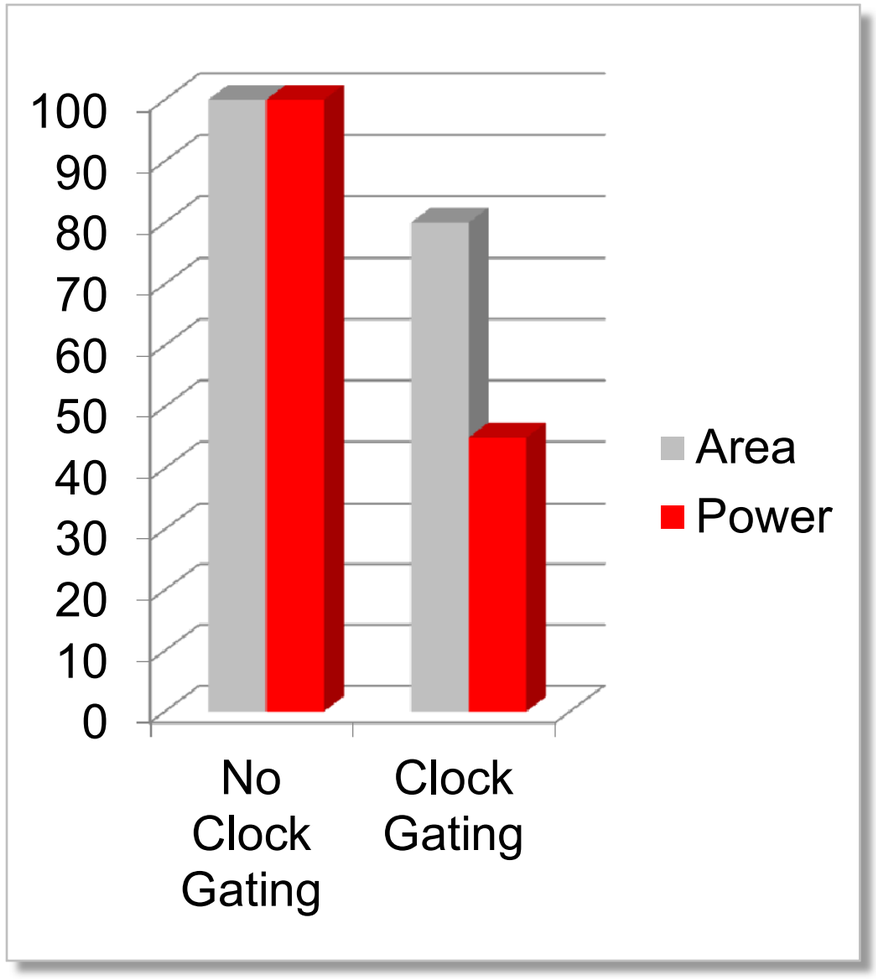
SYNOPSYS®

Convergence Of Logic And Power Synthesis

Power Compiler, 1997



Wireless Application, 250nm



Source: R. Zafalon, STMicroelectronics, SNUG1999

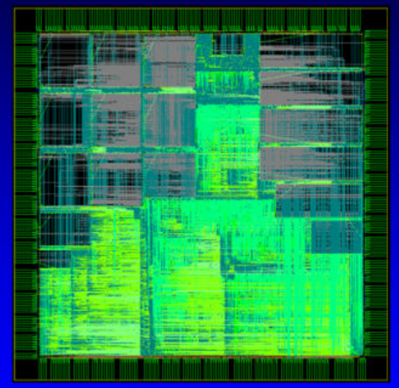
Convergence of Synthesis And Implementation

Physical Compiler, 1999

Physical Compiler

Evaluation Vehicle #1

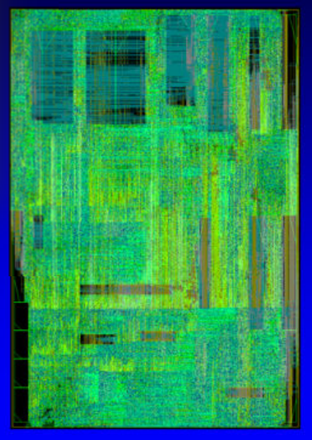
- **Evaluation Objective**
  - Stress robustness, capacity and limits of the tool
- **Design Characteristics**
  - 250 Nanometers
  - Transmission application
  - 220K placeable instances, ~800K gates, 18 RAMs
  - 70% utilization, **irregular**, non-rectilinear floor-plan
  - 70+ Clocks, fastest @ 65 MHz
  - **Traditional flow converged after 4 weeks**
- **Evaluation Results**
  - Compiled in 27 hours
  - Met timing in one pass
  - Design routable, no congestion



gates, 22 RAMs

- 35 combinatorial levels between 2 sequential stages on the critical path
- 2 Clocks, fastest @ 80 MHz
- **Traditional flow converged after 2 weeks**

- **Evaluation Results**
  - Compiled in 10 hours
  - 300ps worst negative slack post-routing after one pass

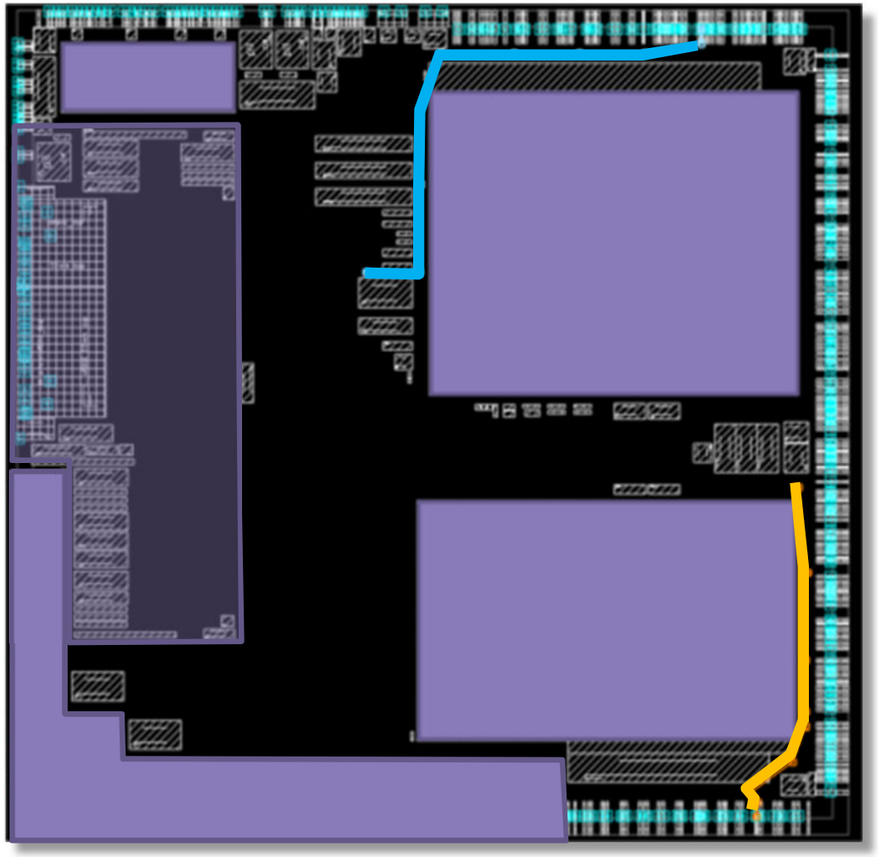


Source: Motorola Timeport; M. Casale-Rossi, STMicroelectronics, 1999

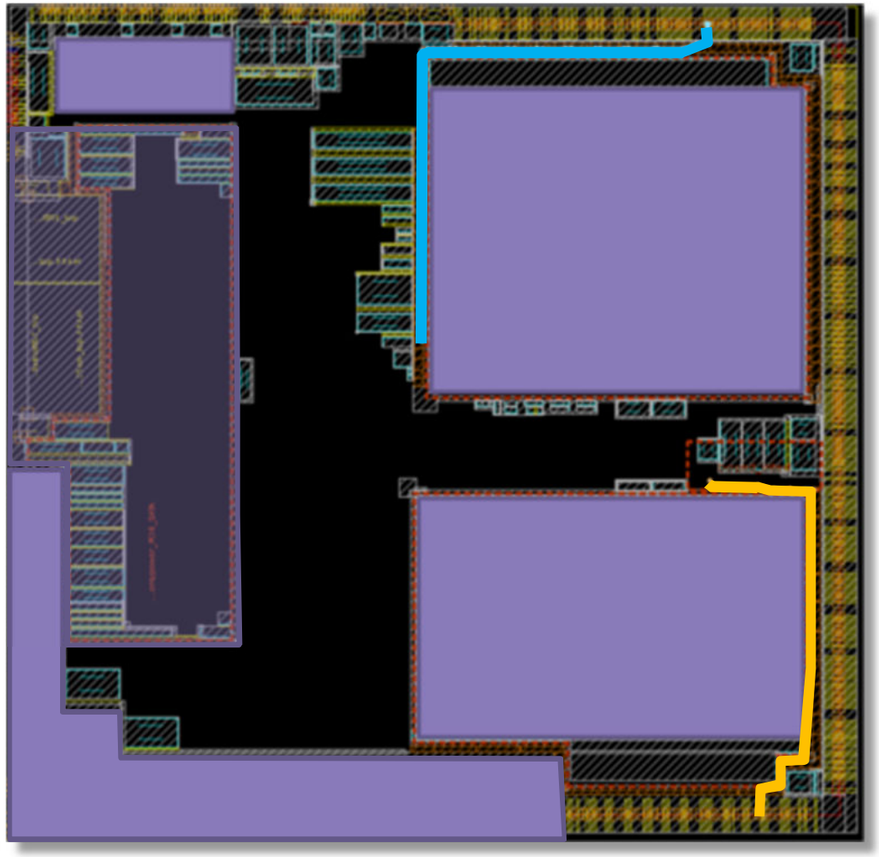
Convergence Continues, 2005

"Some" Placement in Synthesis: Same Critical Paths, Correlation ±5%

Design Compiler



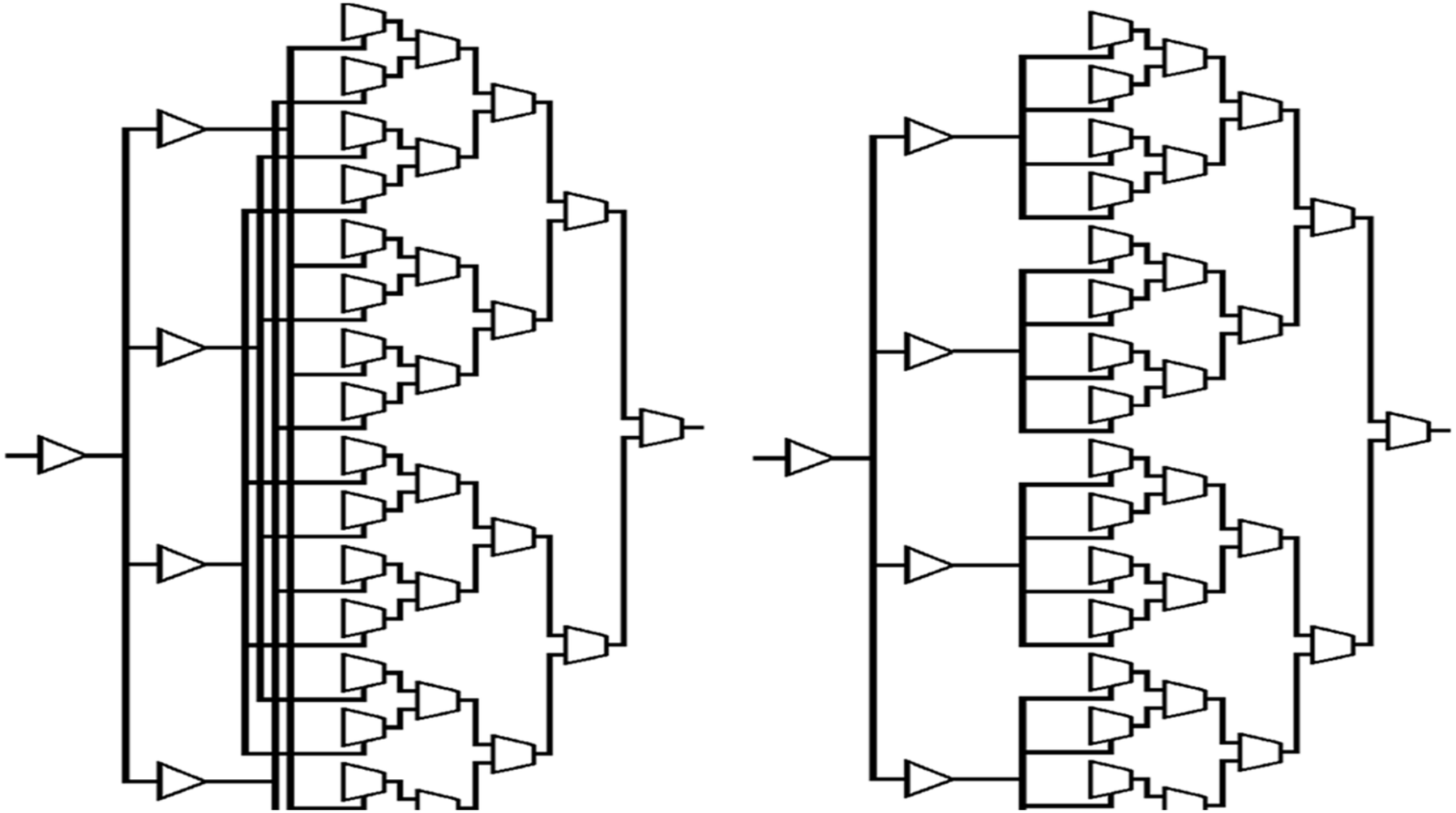
IC Compiler



Source: IDM 2011

Convergence Continues, 2007

“Some” Global Routing In Synthesis: Improving Planarity in a Graph

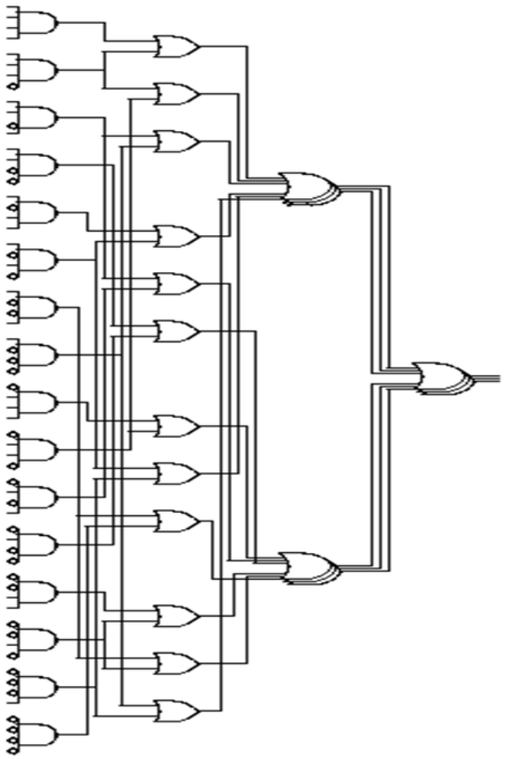


Convergence Continues, 2014

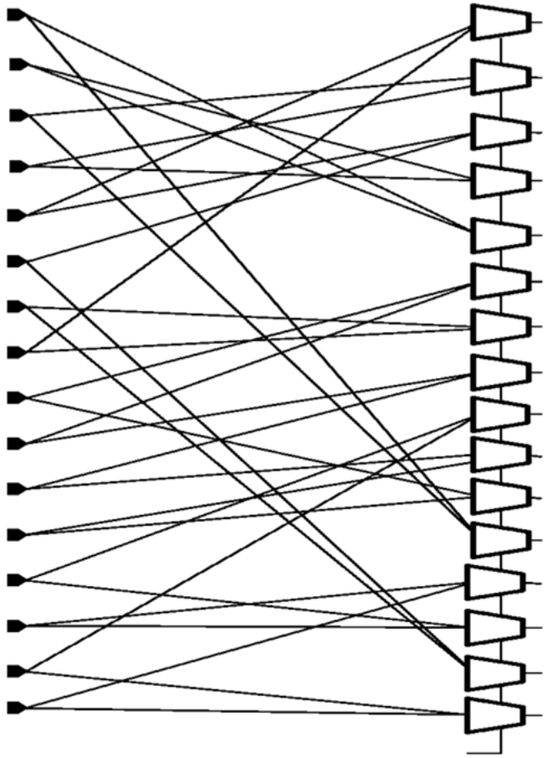
Congestion Analysis Pre-Synthesis Technology

Detects RTL Structures Causing Congestion down in the Implementation Flow

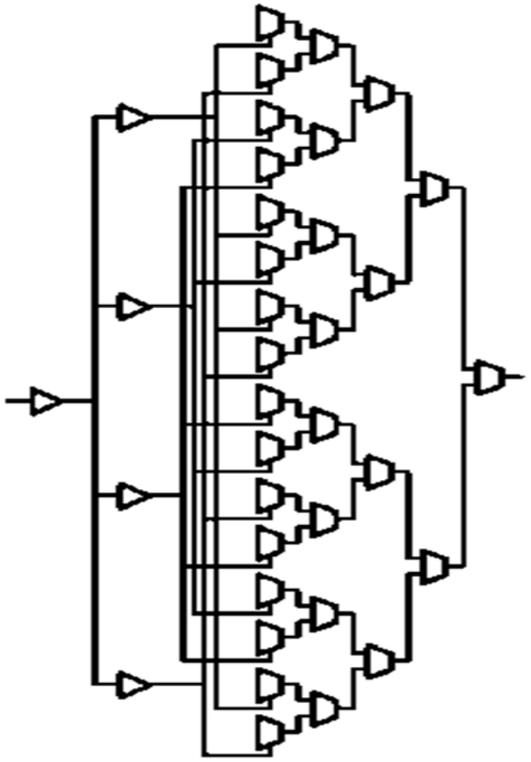
Large ROMs



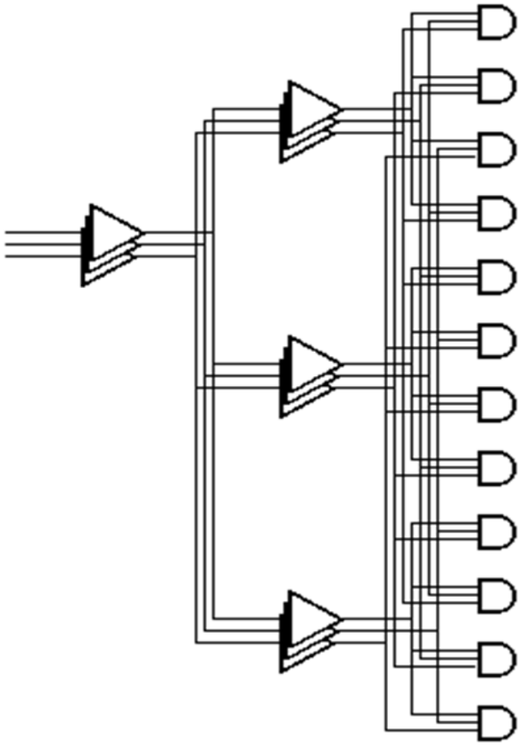
Large Data Switches



Large MUXs



Large Selectors



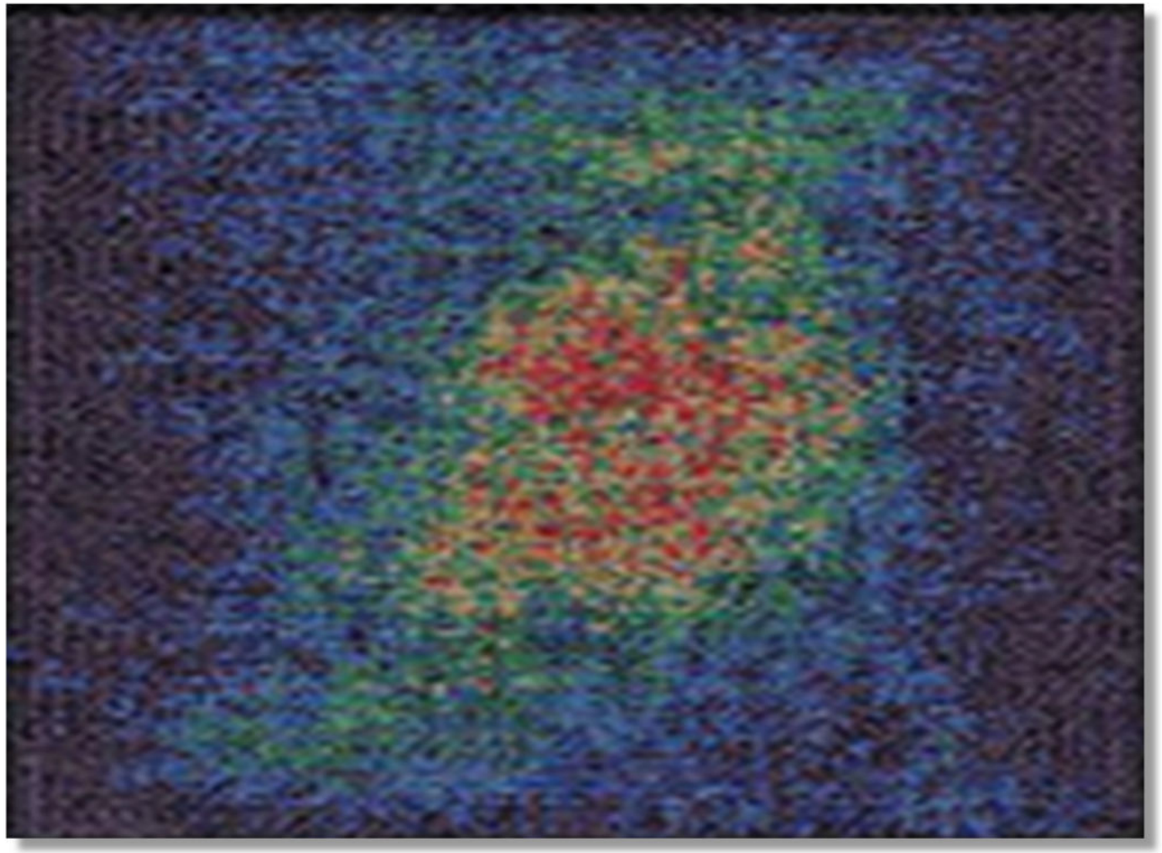
SYNOPSYS®

Congestion Analysis Pre-Synthesis Technology

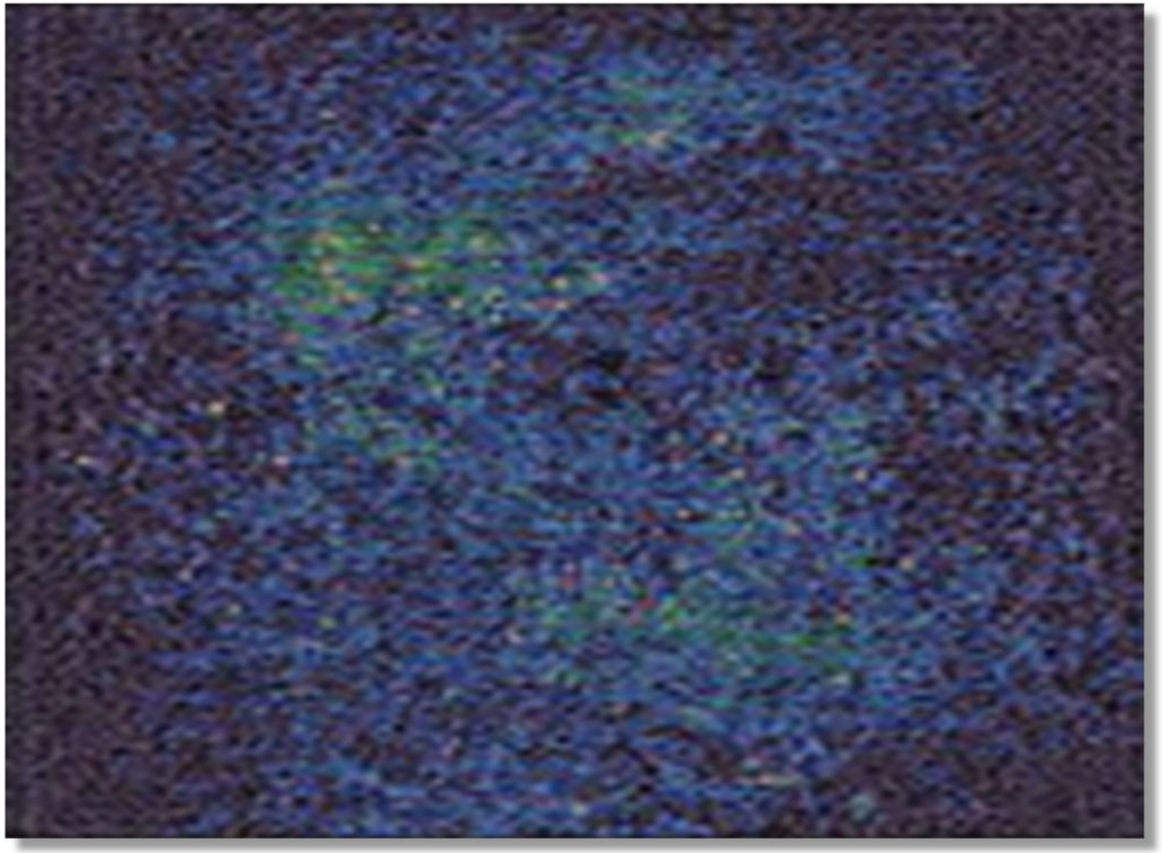
E.g. Large MUX Structures

Same-Inputs Sharing & Connectivity-Based Decomposition

Before



After



Source: Synopsys Research, 2014

Design Compiler, 2001-2015

The Evolution of Synthesis !

	Area	Timing	Power		Runtime
			Dynamic	Static	
2001	100	100	100	100	100
2005	78	91	82	78	20
2010	65	74	66	49	1.6(1)
2015	52	63	66	38	0.6(1)

(1) Design Compiler Multicore

Source: Synopsys Research 2015

Where Do We Stand ?

Well, Telephones Have Made a Great Deal of Progress !

SYNOPSYS®

Smartphones

A Revolutionary… Evolution : Convergence !

50 Things We Won't Do/Use Anymore (Mostly Because of/Thanks to Smartphones)

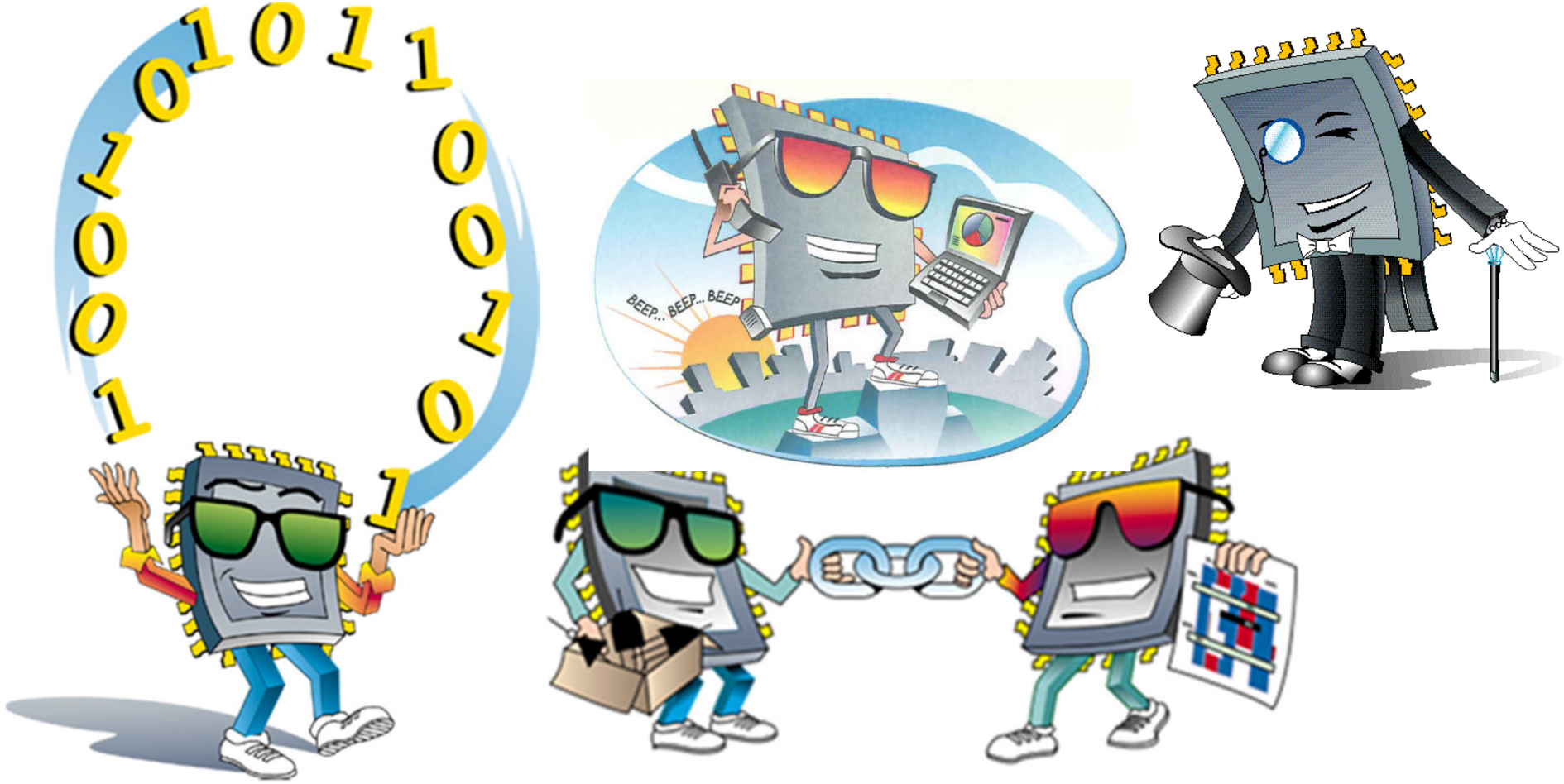
Yesterday

Today

SYNOPSYS®

Where Do We Stand ?

Even Synthesis Went Far Beyond Logic, Isn't It ?

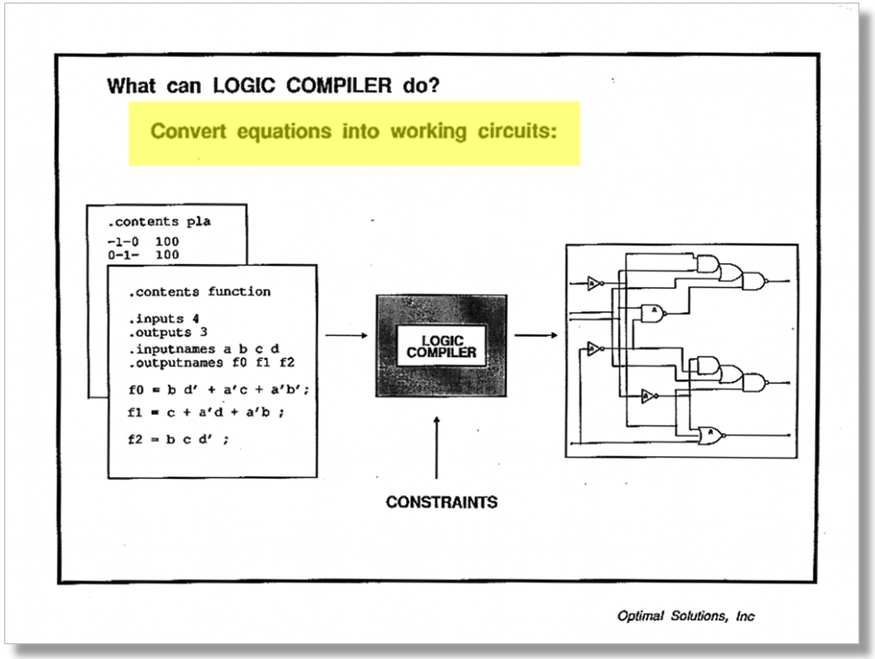


SYNOPSYS®

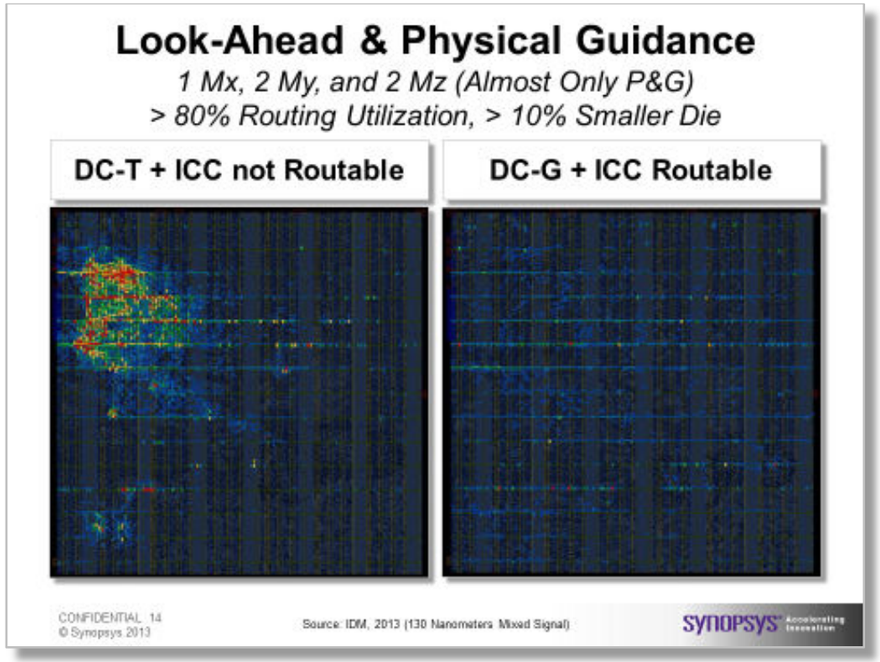
Logic Synthesis

A Revolutionary… Evolution Too : Convergence !

Logic Compiler



Design Compiler

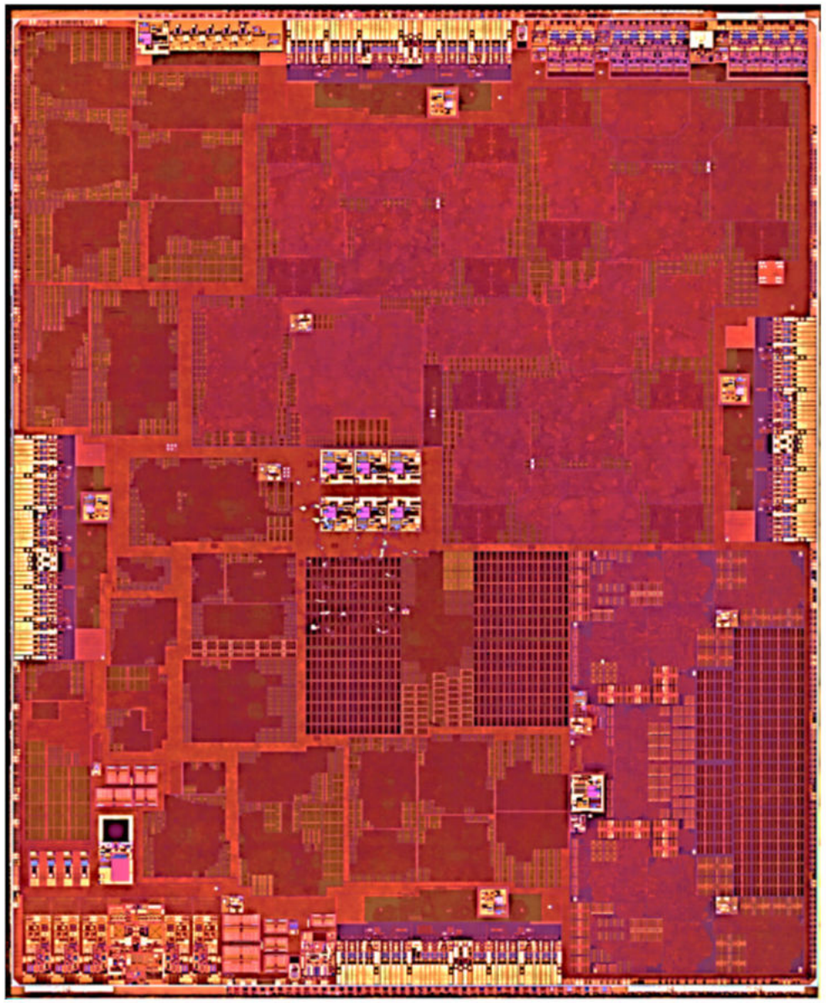


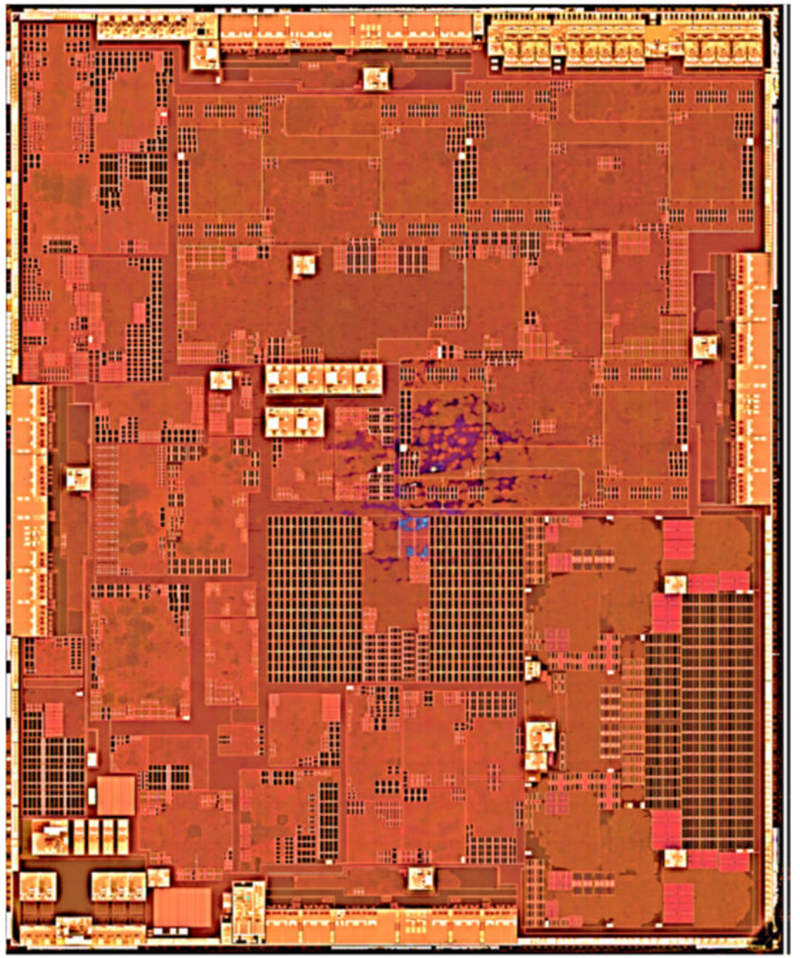
From Equations to Gates, to… "Placed" and Routable Gates

SYNOPSYS®

From Telephone To Logic Synthesis…

…And from Logic Synthesis to… Smartphones



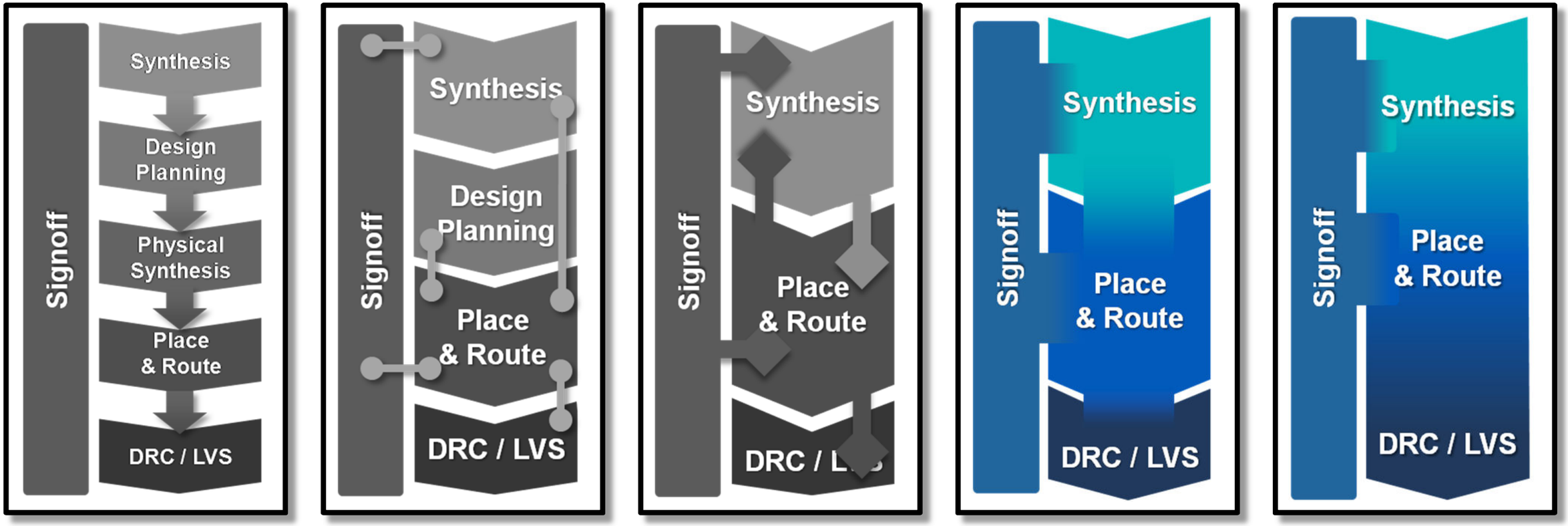


Apple APL1022 TSMC 16 Nanometers (left) and APL0898 Samsung 14 Nanometers (right); Courtesy of ChipWorks, Inc., OCT/2015

SYNOPSYS®

The Evolution Of Logic Synthesis

Convergence !



2003
90 Nanometers
“Interoperability”

2005
65 Nanometers
“Correlation”

2007
45/40 Nanometers
“Look Ahead”

2009
32/28 Nanometers
“In-Design”

2011
22/20 Nanometers
“Convergence”

SYNOPSYS®

The Evolution Of Logic Synthesis

Convergence… But Not Done Yet : Still Differences !

$$G_{ij} = \bigcup_{k=0}^{p_j - 1} (r^{\iota})_j^k c_j^k = 0$$

$$(c \# p)_j = \begin{cases} c_j \cap \bar{p}_j & \text{if } i = j \\ c_j & \text{if } i \neq j \end{cases}$$

$$f(x,y) = \frac{k}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(x',y') \frac{\vec{r} - \vec{r}'}{\left|\vec{r} - \vec{r}'\right|^2} dx'dy'$$

DRC / LVS

SYNOPSYS®

Looking Into The Next Decade

There Is a Great Deal Of New Technology Ahead !

Giacomo Balla, Future, 1918; Private Collection

Looking Into The Next Decade

1T Transistors per Die by the End of This Decade, "1" nm by the End of the Next

Infinitely Large, Infinitely Small, Infinitely Many

	2013	2015	2017	2019	2021	2023	2025	202x
"Technology Node" (nm)	**"14"**	**"10"**	**"7"**	**"5"**	**"3.5"**	**"2.5"**	**"1.8"**	**"1.3"**
DRAM ½ Pitch (nm)	28	24	20	17	14	12	10	7.7
MPU/ASIC ½ Pitch (nm)	40	32	25	20	16	13	10	7
FLASH ½ Pitch (nm)	18	15	13	11	9	8	8	8
MPU Printed Gate Length (nm)	28	22	18	14	11	9	7	5
MPU Physical Gate Length (nm)	20	17	14	12	10	8	7	5
Theoretical Integration Capacity (BT)	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**

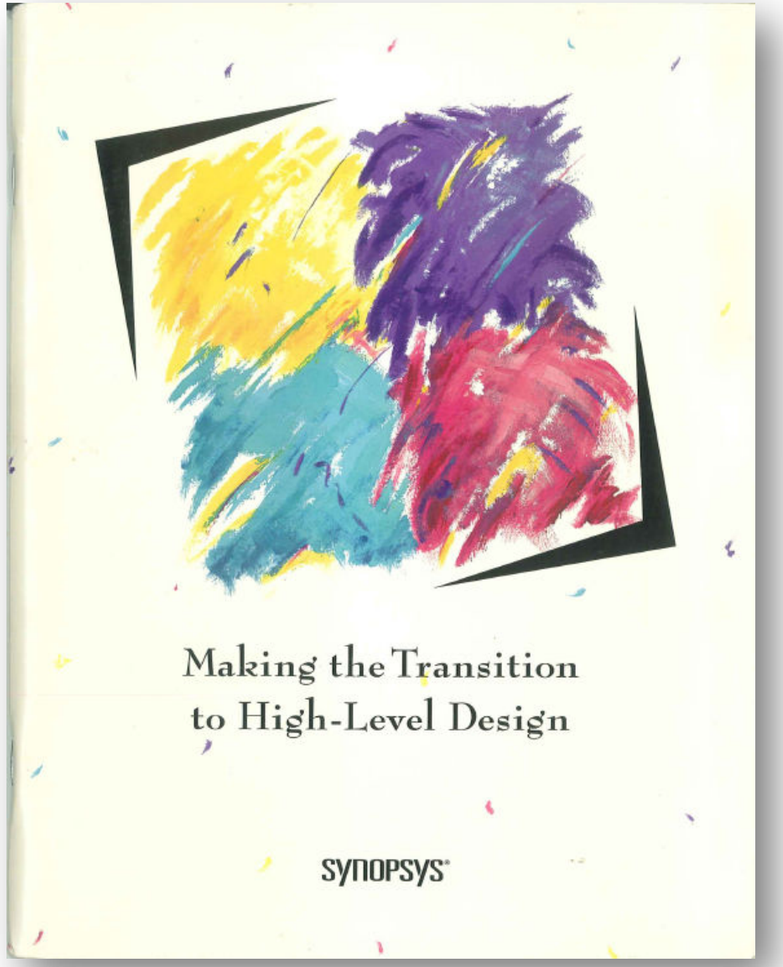
(Assuming 450mm Wafers in Production in 2018, and EUV in Production after 10nm)

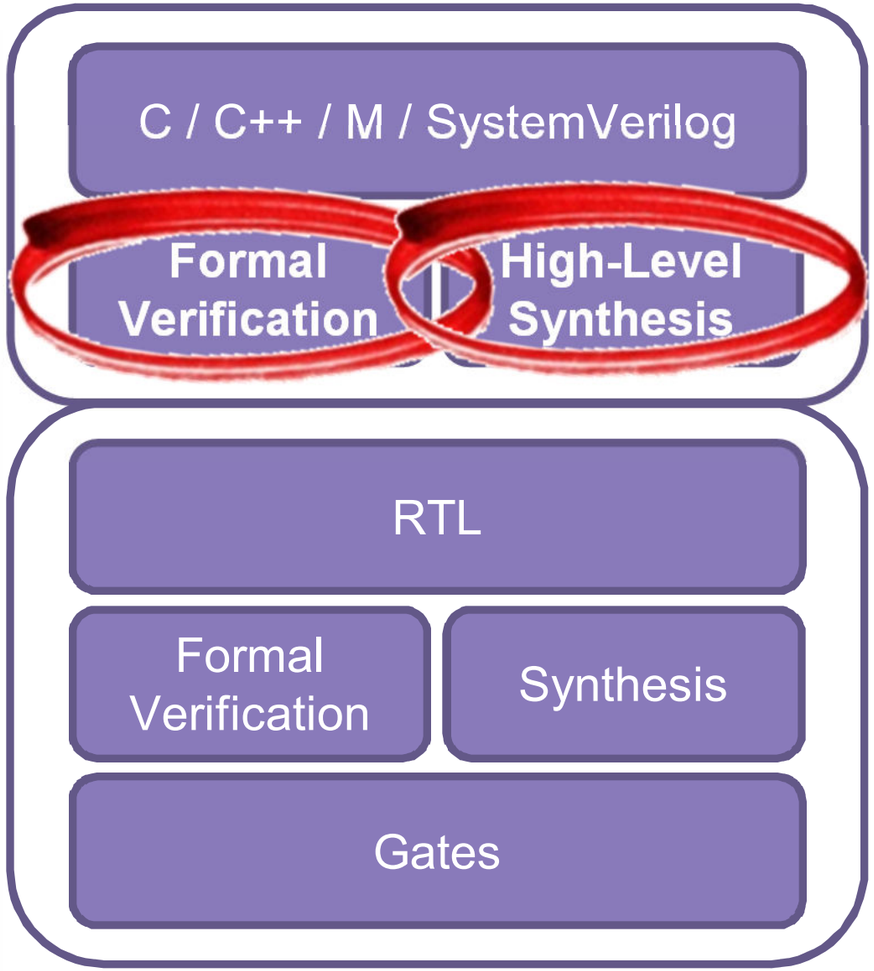
	2013	2015	2017	2019	2021	2023	2025	202x
Size of Internet (IP Addresses)		**25B**		**~ 50B**			**100B**	

Source: ITRS, 2014; G.-Q. Zhang, et al. "Evolution of the Internet and its Cores." New Journal of Physics, 2008

Looking Into The Next Decade

Making the Transition to High-Level Design… Again !

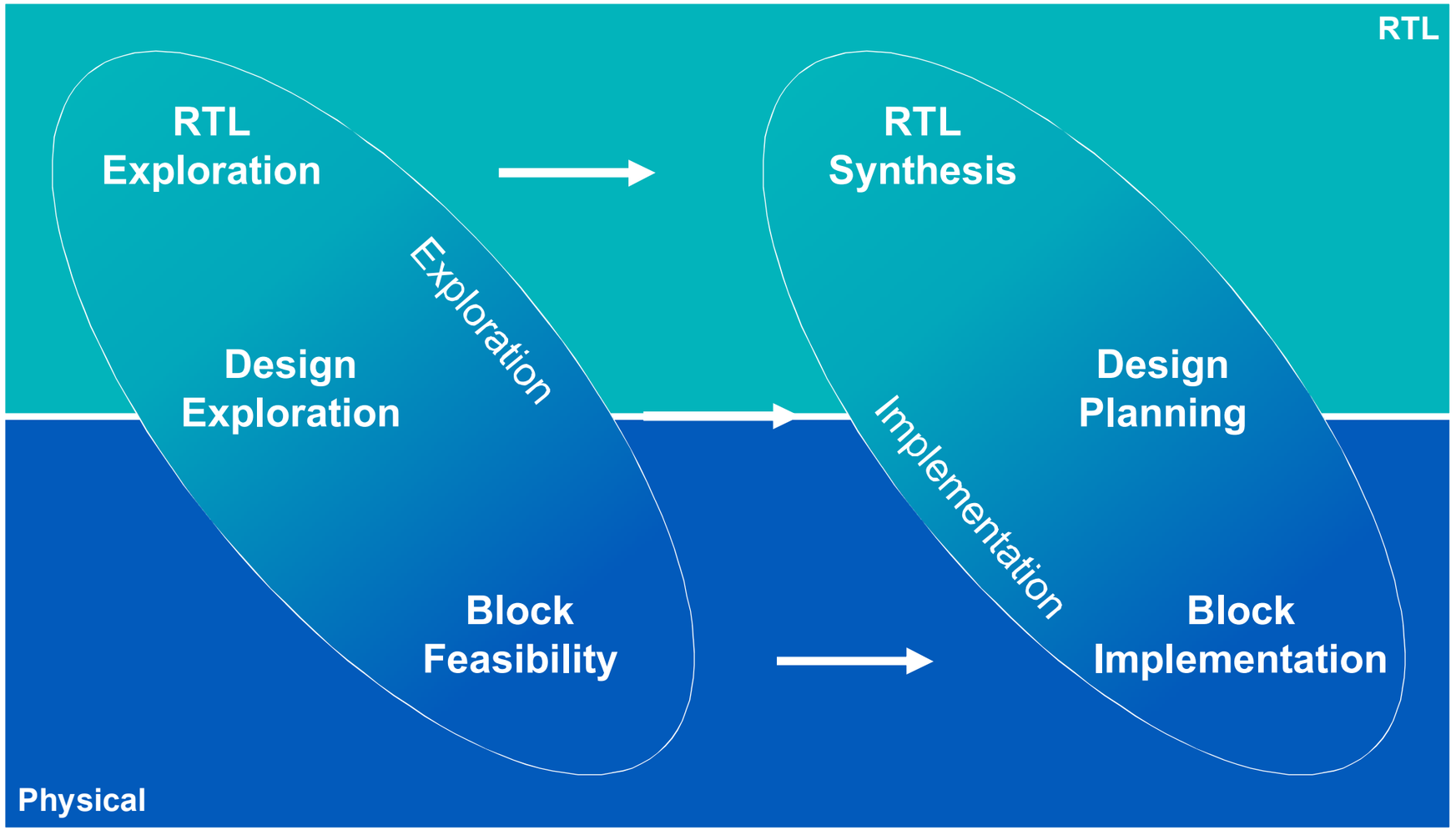




SYNOPSYS®

Looking Into The Next Decade

Evolving Towards Two, Closely Connected "Sub-Systems"
Explore & Analyze, then Implement



SYNOPSYS®

Looking Into The Next Decade

Evolving Towards Two, Closely Connected "Sub-Systems"
Explore & Analyze, then Implement

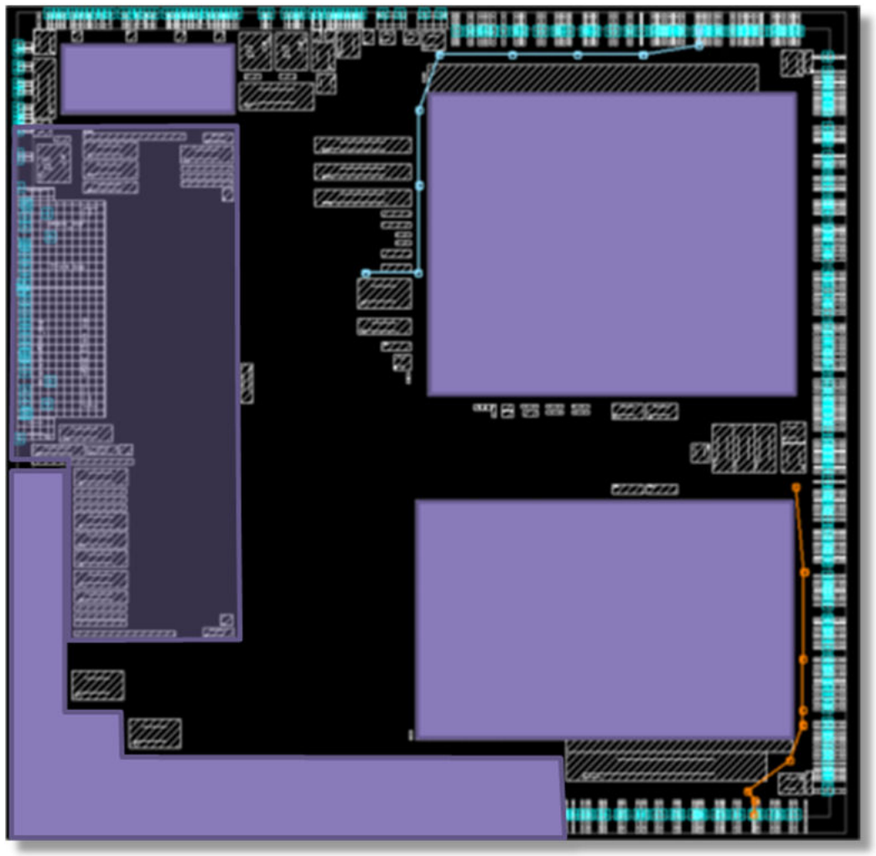
- Until now :
  - Mostly "preserve" the gates
  - "Change" placement, and/or routing to close timing, power, …
    - Very, very time consuming, and
    - Leads to an infinite number of iterations
- In the future :
  - Once the objective is "within reach" …
  - "Hold" placement and routing
  - Systematically "change" the gates
    - Same footprint, different timing, power, temperature inversion point, etc.
    - The richness of the library is fundamental

Looking Into The Next Decade

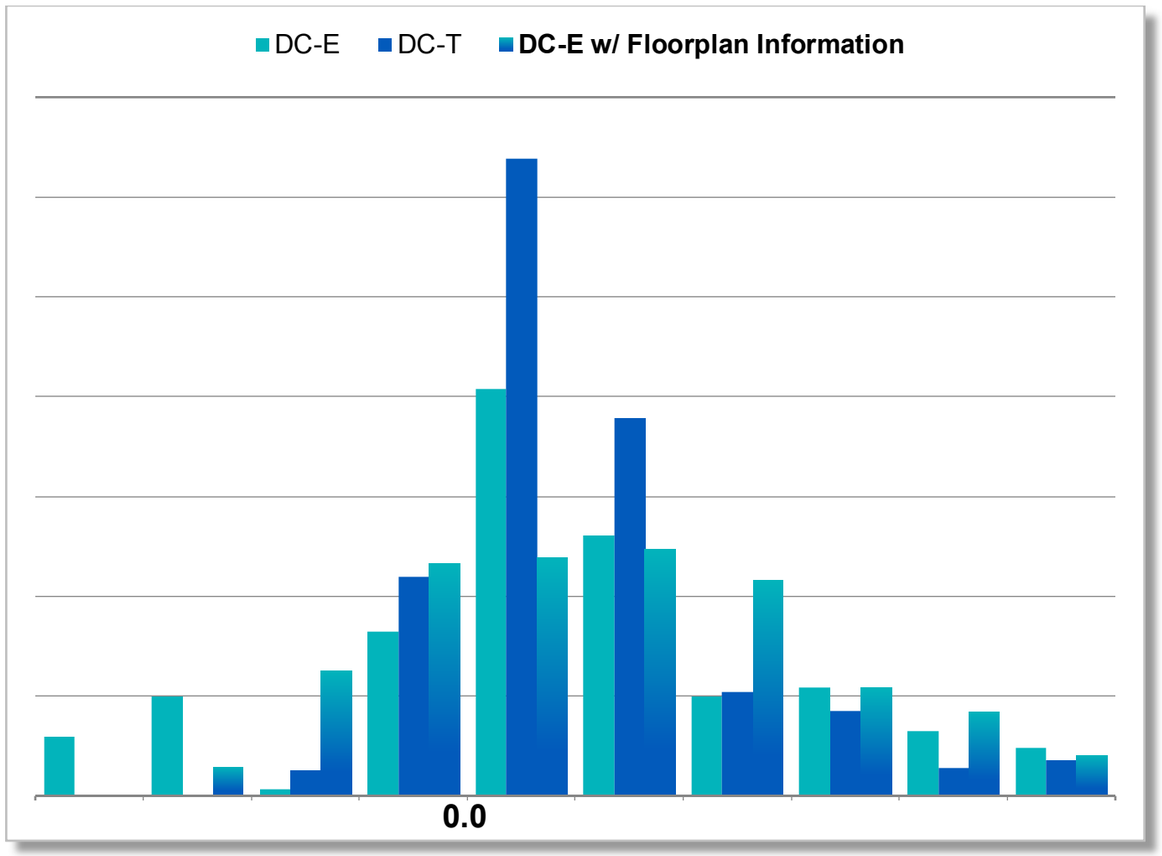
RTL Exploration Is ~ 6X Faster Than Full Synthesis

Slack Distribution Comparison, Correlation ±8%

Floorplan Information



Slack Histogram

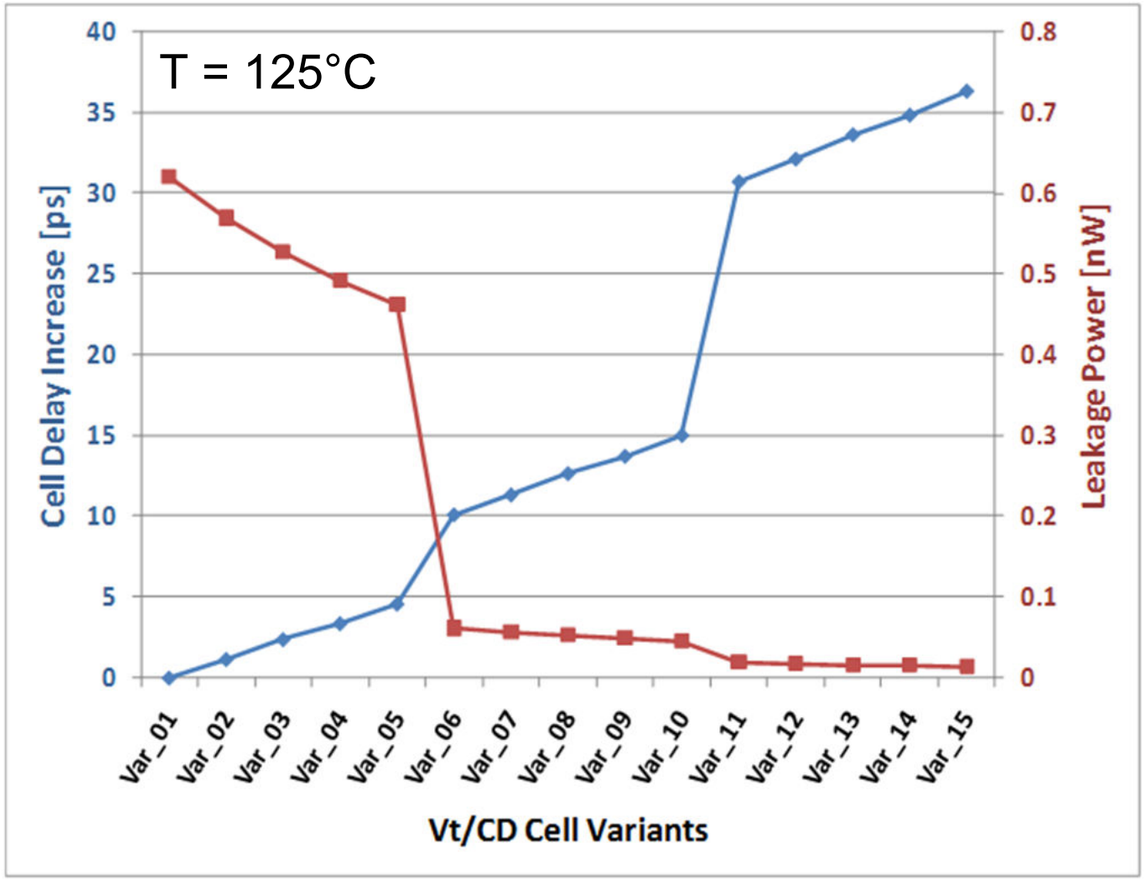


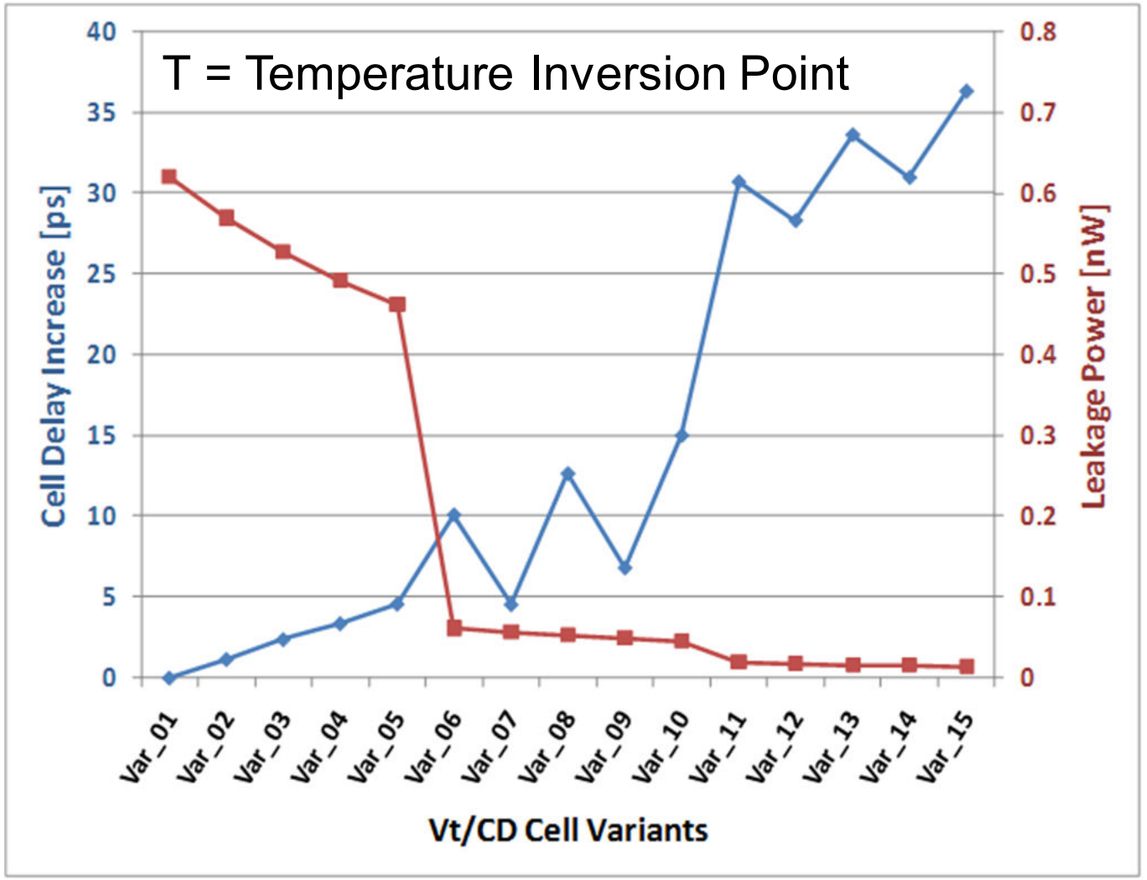
SYNOPSYS®

Looking Into The Next Decade

Today Libraries Contain Thousand of Elements

Many Variants with the Same Footprint but Different "Performance"





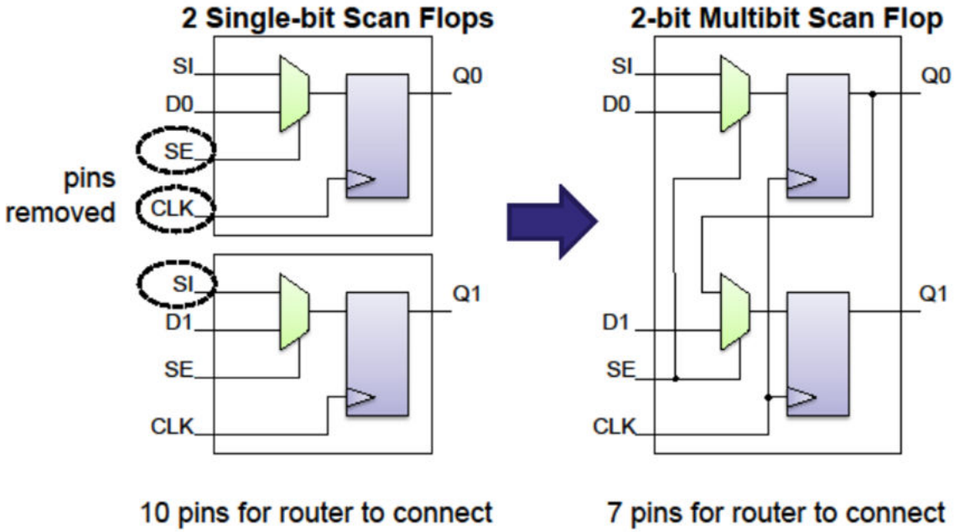
SYNOPSYS®

Looking Into The Next Decade

Extending the Use of Multi-Bit Structures Will Save Area and Power, and Will Alleviate Congestion, Simplifying Routing

Multi-bit Flip-Flop (MBFF)

Fewer pins for the router to connect

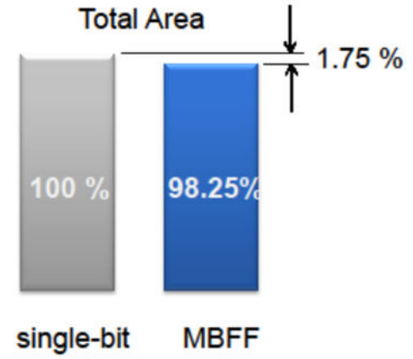


SNUG 2014 6

Non-MBFF vs. MBFF - Area Results

1.75% Area Savings

Savings	DC-G	ICC (post-CTS)
Register Area	- 7.3%	- 7.10%
Overall Std-cell Area	- 2.6%	- 1.75%



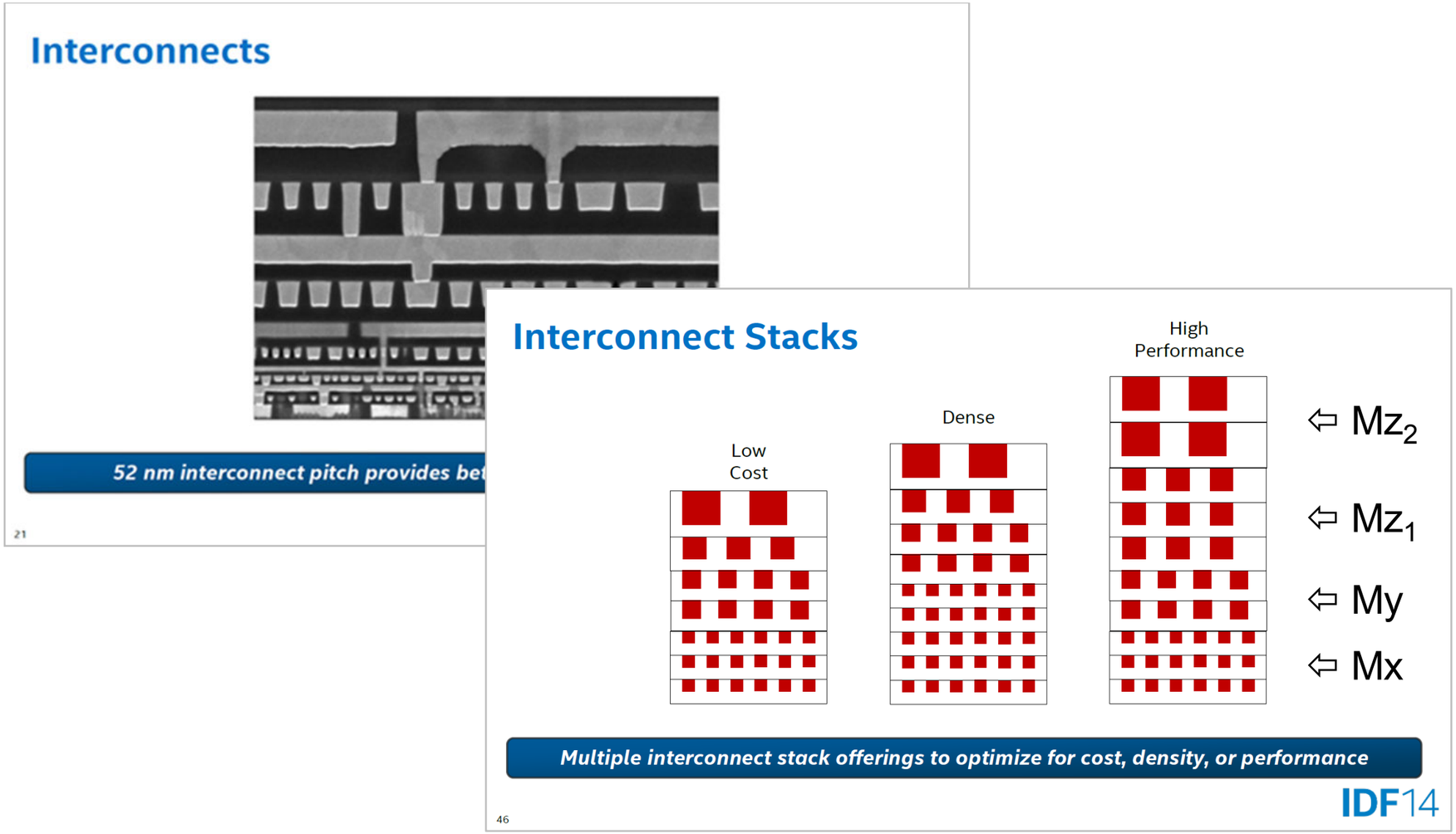
SNUG 2014 24

Source: K. Umino, Hyon Han, Samsung, SNUG Austin 2014

SYNOPSYS®

Convergence Of Synthesis And Implementation

RC Variation Among the Many Metal Layers Makes Estimates Extremely Difficult
Forcing to Bring Global Routing, and Probably Detailed, into the Synthesis Picture



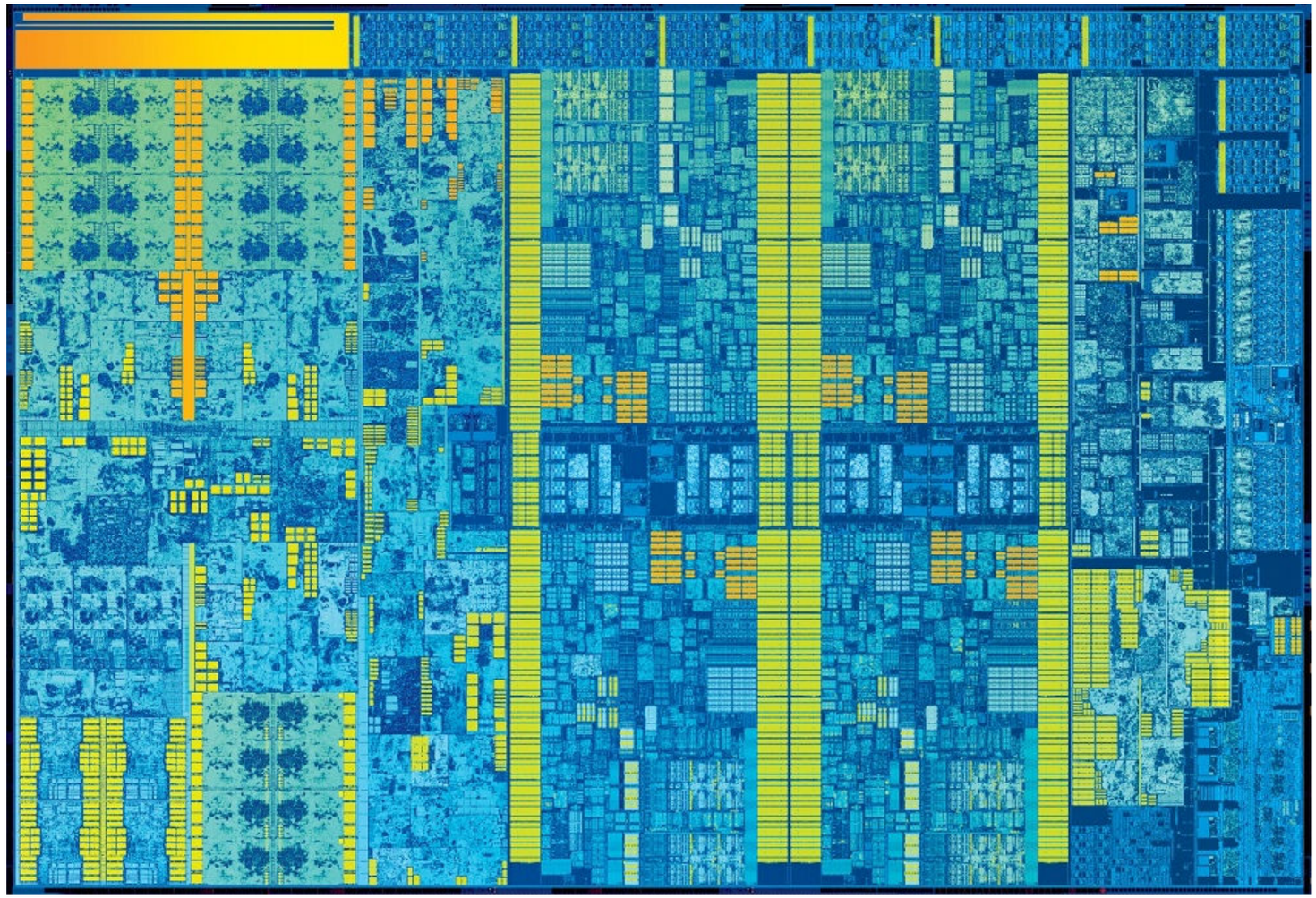
Source: M. Bohr, Intel, IDF 2014; ITRS Edition 2013

SYNOPSYS®

Looking Into The Next Decade

Today MPU Have 4-8 Cores, 16 Cores Are Just Around the Corner

Logic Synthesis Algorithms Must Be Suitable for Fine Parallelization

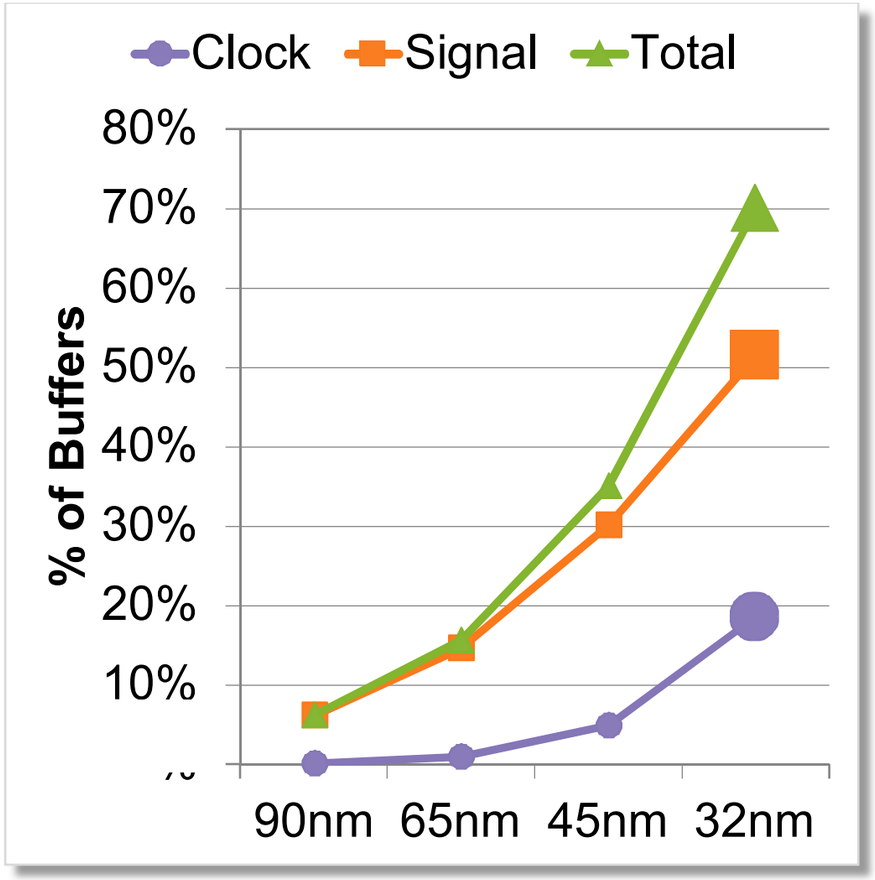


Intel 6th Gen Core, Skylake, 14 Nanometers; Source: Intel, 2015

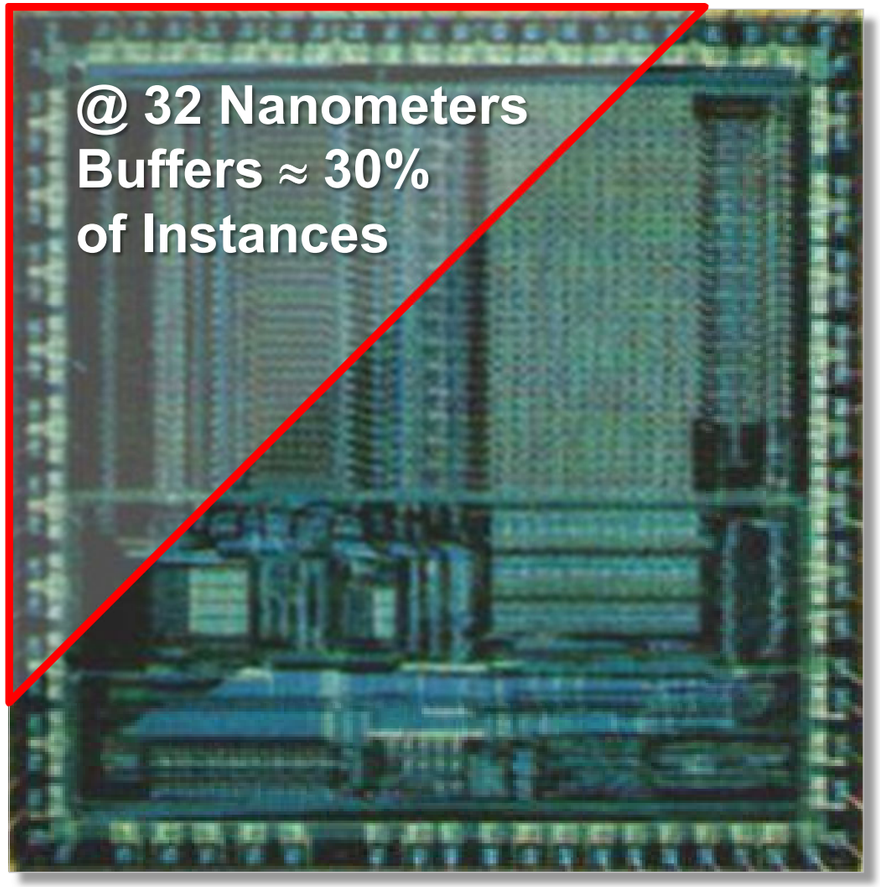
Looking Into The Next Decade

Buffers Insertion Optimization

A 2003 Estimate…



A 2010 Data Point…

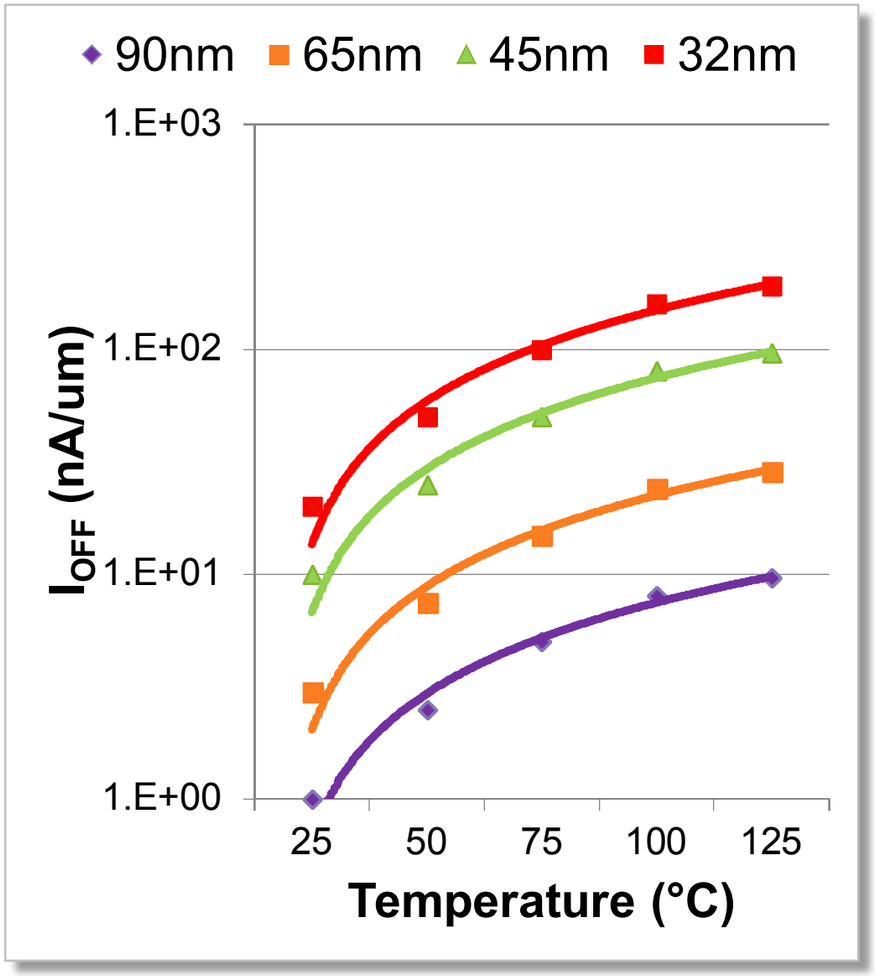


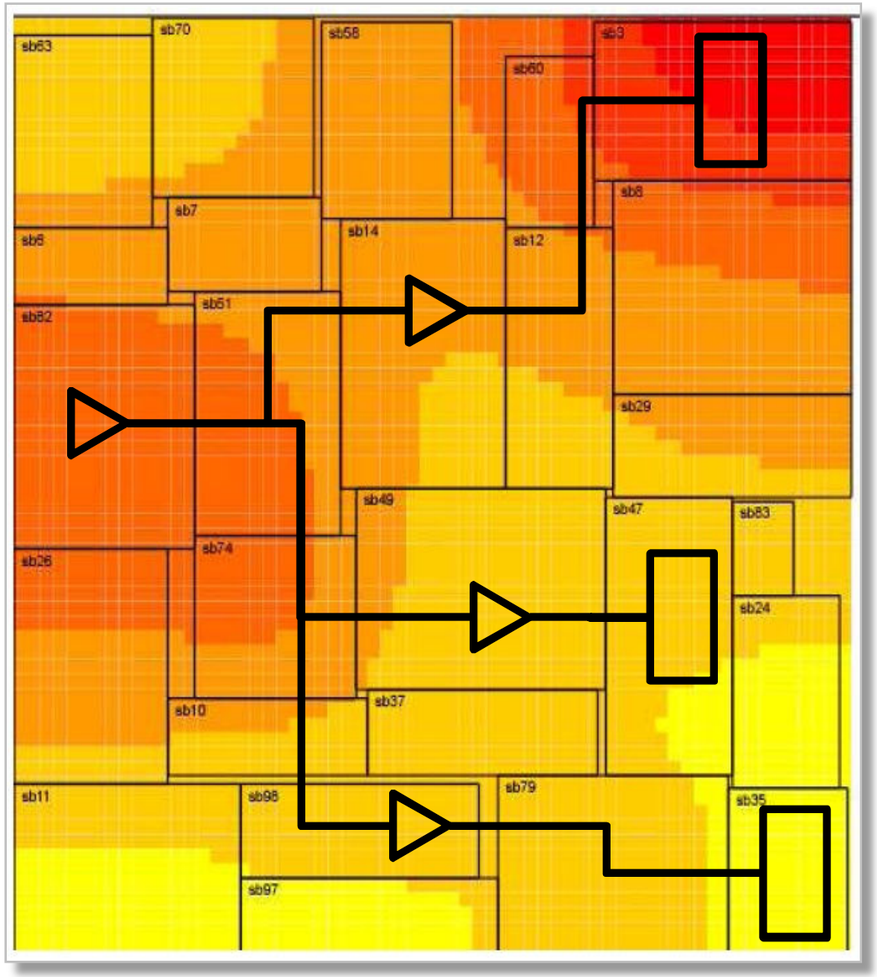
Source: P. Saxena, Intel, IDPS 2003; Synopsys Research 2010

SYNOPSYS®

Looking Into The Next Decade

Temperature-Aware Synthesis



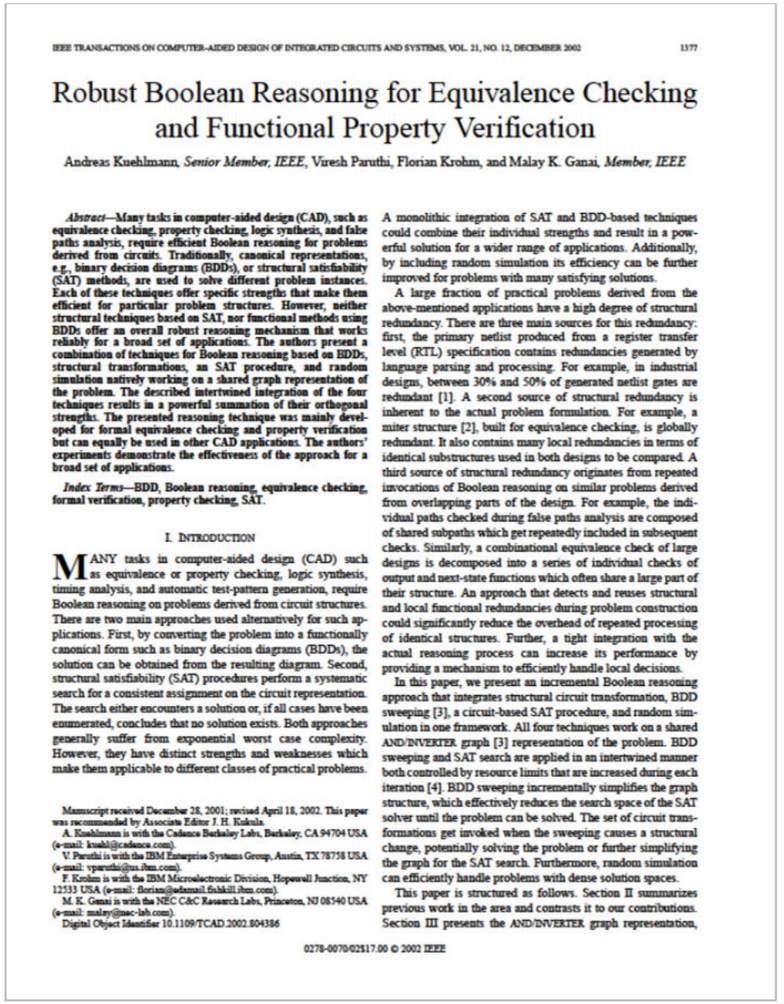


Source: P. Gelsinger, Intel, DAC 2004

SYNOPSYS®

Looking Into The Next Decade

Andreas Kuehlmann, Robert K. Brayton, Alan Mishchenko,
Luca Amarù, Pierre-Emmanuel Gaillardon, Giovanni De Micheli



IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 21, NO. 12, DECEMBER 2002 1377

Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification

Andreas Kuehlmann, *Senior Member, IEEE*, Viresh Paruthi, Florian Krohm, and Malay K. Ganai, *Member, IEEE*

Abstract—Many tasks in computer-aided design (CAD), such as equivalence checking, property checking, logic synthesis, and false paths analysis, require efficient Boolean reasoning for problems derived from circuits. Traditionally, canonical representations, e.g., binary decision diagrams (BDDs), or structural satisfiability (SAT) methods, are used to solve different problem instances. Each of these techniques offer specific strengths that make them efficient for particular problem structures. However, neither structural techniques based on SAT, nor functional methods using BDDs offer an overall robust reasoning mechanism that works reliably for a broad set of applications. The authors present a combination of techniques for Boolean reasoning based on BDDs, structural transformations, an SAT procedure, and random simulation natively working on a shared graph representation of the problem. The described intertwined integration of the four techniques results in a powerful summation of their orthogonal strengths. The presented reasoning technique was mainly developed for formal equivalence checking and property verification but can equally be used in other CAD applications. The authors' experiments demonstrate the effectiveness of the approach for a broad set of applications.

Index Terms—BDD, Boolean reasoning, equivalence checking, formal verification, property checking, SAT.

I. Introduction

MANY tasks in computer-aided design (CAD) such as equivalence or property checking, logic synthesis, timing analysis, and automatic test-pattern generation, require Boolean reasoning on problems derived from circuit structures. There are two main approaches used alternatively for such applications. First, by converting the problem into a functionally canonical form such as binary decision diagrams (BDDs), the solution can be obtained from the resulting diagram. Second, structural satisfiability (SAT) procedures perform a systematic search for a consistent assignment on the circuit representation. The search either encounters a solution or, if all cases have been enumerated, concludes that no solution exists. Both approaches generally suffer from exponential worst case complexity. However, they have distinct strengths and weaknesses which make them applicable to different classes of practical problems.

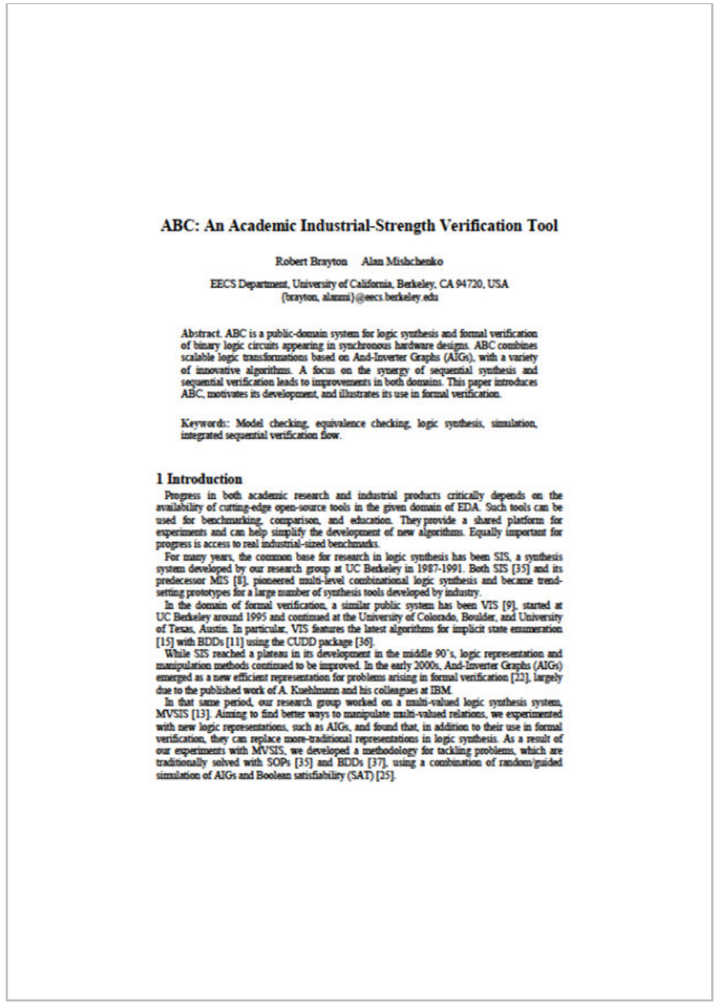
A monolithic integration of SAT and BDD-based techniques could combine their individual strengths and result in a powerful solution for a wider range of applications. Additionally, by including random simulation its efficiency can be further improved for problems with many satisfying solutions.

A large fraction of practical problems derived from the above-mentioned applications have a high degree of structural redundancy. There are three main sources for this redundancy: first, the primary netlist produced from a register transfer level (RTL) specification contains redundancies generated by language parsing and processing. For example, in industrial designs, between 30% and 50% of generated netlist gates are redundant [1]. A second source of structural redundancy is inherent to the actual problem formulation. For example, a miter structure [2], built for equivalence checking, is globally redundant. It also contains many local redundancies in terms of identical substructures used in both designs to be compared. A third source of structural redundancy originates from repeated invocations of Boolean reasoning on similar problems derived from overlapping parts of the design. For example, the individual paths checked during false paths analysis are composed of shared subpaths which get repeatedly included in subsequent checks. Similarly, a combinational equivalence check of large designs is decomposed into a series of individual checks of output and next-state functions which often share a large part of their structure. An approach that detects and reuses structural and local functional redundancies during problem construction could significantly reduce the overhead of repeated processing of identical structures. Further, a tight integration with the actual reasoning process can increase its performance by providing a mechanism to efficiently handle local decisions.

In this paper, we present an incremental Boolean reasoning approach that integrates structural circuit transformation, BDD sweeping [3], a circuit-based SAT procedure, and random simulation in one framework. All four techniques work on a shared AND/INVERTER graph [3] representation of the problem. BDD sweeping and SAT search are applied in an intertwined manner both controlled by resource limits that are increased during each iteration [4]. BDD sweeping incrementally simplifies the graph structure, which effectively reduces the search space of the SAT solver until the problem can be solved. The set of circuit transformations get invoked when the sweeping causes a structural change, potentially solving the problem or further simplifying the graph for the SAT search. Furthermore, random simulation can efficiently handle problems with dense solution spaces.

This paper is structured as follows. Section II summarizes previous work in the area and contrasts it to our contributions. Section III presents the AND/INVERTER graph representation,

Manuscript received December 28, 2001; revised April 18, 2002. This paper was recommended by Associate Editor J. H. Kukula.
A. Kuehlmann is with the Cadence Berkeley Labs, Berkeley, CA 94704 USA (e-mail: kuehl@cadence.com).
V. Paruthi is with the IBM Enterprise Systems Group, Austin, TX 78758 USA (e-mail: vparuthi@us.ibm.com).
F. Krohm is with the IBM Microelectronic Division, Hopewell Junction, NY 12533 USA (e-mail: florian@edxmail.fishkill.ibm.com).
M. K. Ganai is with the NEC C&C Research Labs, Princeton, NJ 08540 USA (e-mail: malay@nec-lab.com).
Digital Object Identifier 10.1109/TCAD.2002.804386



ABC: An Academic Industrial-Strength Verification Tool

Robert Brayton Alan Mishchenko

EECS Department, University of California, Berkeley, CA 94720, USA
{brayton, alanmi}@eecs.berkeley.edu

Abstract. ABC is a public-domain system for logic synthesis and formal verification of binary logic circuits appearing in synchronous hardware designs. ABC combines scalable logic transformations based on And-Inverter Graphs (AIGs), with a variety of innovative algorithms. A focus on the synergy of sequential synthesis and sequential verification leads to improvements in both domains. This paper introduces ABC, motivates its development, and illustrates its use in formal verification.

Keywords: Model checking, equivalence checking, logic synthesis, simulation, integrated sequential verification flow.

1 Introduction

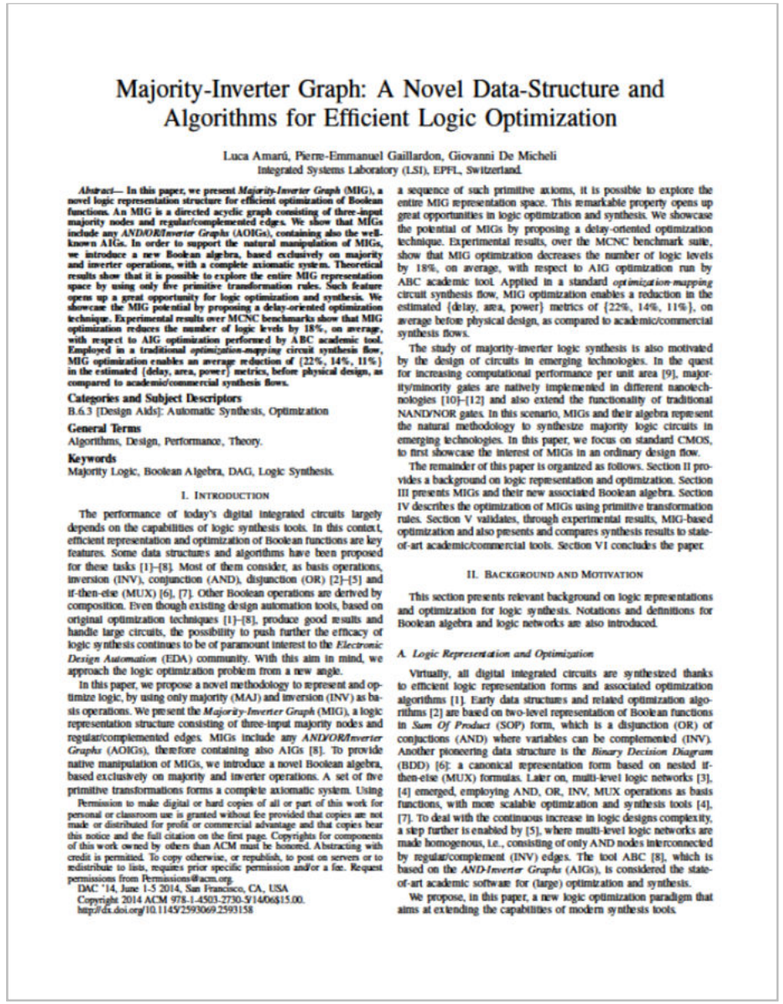
Progress in both academic research and industrial products critically depends on the availability of cutting-edge open-source tools in the given domain of EDA. Such tools can be used for benchmarking, comparison, and education. They provide a shared platform for experiments and can help simplify the development of new algorithms. Equally important for progress is access to real industrial-sized benchmarks.

For many years, the common base for research in logic synthesis has been SIS, a synthesis system developed by our research group at UC Berkeley in 1987-1991. Both SIS [35] and its predecessor MIS [8], pioneered multi-level combinational logic synthesis and became trend-setting prototypes for a large number of synthesis tools developed by industry.

In the domain of formal verification, a similar public system has been VIS [9], started at UC Berkeley around 1995 and continued at the University of Colorado, Boulder, and University of Texas, Austin. In particular, VIS features the latest algorithms for implicit state enumeration [15] with BDDs [11] using the CUDD package [36].

While SIS reached a plateau in its development in the middle 90's, logic representation and manipulation methods continued to be improved. In the early 2000s, And-Inverter Graphs (AIGs) emerged as a new efficient representation for problems arising in formal verification [22], largely due to the published work of A. Kuehlmann and his colleagues at IBM.

In that same period, our research group worked on a multi-valued logic synthesis system, MVSIS [13]. Aiming to find better ways to manipulate multi-valued relations, we experimented with new logic representations, such as AIGs, and found that, in addition to their use in formal verification, they can replace more-traditional representations in logic synthesis. As a result of our experiments with MVSIS, we developed a methodology for tackling problems, which are traditionally solved with SOPs [35] and BDDs [37], using a combination of random/guided simulation of AIGs and Boolean satisfiability (SAT) [25].



Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization

Luca Amarú, Pierre-Emmanuel Gaillardon, Giovanni De Micheli
Integrated Systems Laboratory (LSI), EPFL, Switzerland.

Abstract— In this paper, we present *Majority-Inverter Graph* (MIG), a novel logic representation structure for efficient optimization of Boolean functions. An MIG is a directed acyclic graph consisting of three-input majority nodes and regular/complemented edges. We show that MIGs include any *AND/OR/Inverter Graphs* (AOIGs), containing also the well-known AIGs. In order to support the natural manipulation of MIGs, we introduce a new Boolean algebra, based exclusively on majority and inverter operations, with a complete axiomatic system. Theoretical results show that it is possible to explore the entire MIG representation space by using only five primitive transformation rules. Such feature opens up a great opportunity for logic optimization and synthesis. We showcase the MIG potential by proposing a delay-oriented optimization technique. Experimental results over MCNC benchmarks show that MIG optimization reduces the number of logic levels by 18%, on average, with respect to AIG optimization performed by ABC academic tool. Employed in a traditional *optimization-mapping* circuit synthesis flow, MIG optimization enables an average reduction of {22%, 14%, 11%} in the estimated {delay, area, power} metrics, before physical design, as compared to academic/commercial synthesis flows.

Categories and Subject Descriptors
B.6.3 [Design Aids]: Automatic Synthesis, Optimization

General Terms
Algorithms, Design, Performance, Theory.

Keywords
Majority Logic, Boolean Algebra, DAG, Logic Synthesis.

I. Introduction

The performance of today's digital integrated circuits largely depends on the capabilities of logic synthesis tools. In this context, efficient representation and optimization of Boolean functions are key features. Some data structures and algorithms have been proposed for these tasks [1]–[8]. Most of them consider, as basis operations, inversion (INV), conjunction (AND), disjunction (OR) [2]–[5] and if-then-else (MUX) [6], [7]. Other Boolean operations are derived by composition. Even though existing design automation tools, based on original optimization techniques [1]–[8], produce good results and handle large circuits, the possibility to push further the efficacy of logic synthesis continues to be of paramount interest to the *Electronic Design Automation* (EDA) community. With this aim in mind, we approach the logic optimization problem from a new angle.

In this paper, we propose a novel methodology to represent and optimize logic, by using only majority (MAJ) and inversion (INV) as basis operations. We present the *Majority-Inverter Graph* (MIG), a logic representation structure consisting of three-input majority nodes and regular/complemented edges. MIGs include any *AND/OR/Inverter Graphs* (AOIGs), therefore containing also AIGs [8]. To provide native manipulation of MIGs, we introduce a novel Boolean algebra, based exclusively on majority and inverter operations. A set of five primitive transformations forms a complete axiomatic system. Using

DAC '14, June 1-5 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593158

a sequence of such primitive axioms, it is possible to explore the entire MIG representation space. This remarkable property opens up great opportunities in logic optimization and synthesis. We showcase the potential of MIGs by proposing a delay-oriented optimization technique. Experimental results, over the MCNC benchmark suite, show that MIG optimization decreases the number of logic levels by 18%, on average, with respect to AIG optimization run by ABC academic tool. Applied in a standard *optimization-mapping* circuit synthesis flow, MIG optimization enables a reduction in the estimated {delay, area, power} metrics of {22%, 14%, 11%}, on average before physical design, as compared to academic/commercial synthesis flows.

The study of majority-inverter logic synthesis is also motivated by the design of circuits in emerging technologies. In the quest for increasing computational performance per unit area [9], majority/minority gates are natively implemented in different nanotechnologies [10]–[12] and also extend the functionality of traditional NAND/NOR gates. In this scenario, MIGs and their algebra represent the natural methodology to synthesize majority logic circuits in emerging technologies. In this paper, we focus on standard CMOS, to first showcase the interest of MIGs in an ordinary design flow.

The remainder of this paper is organized as follows. Section II provides a background on logic representation and optimization. Section III presents MIGs and their new associated Boolean algebra. Section IV describes the optimization of MIGs using primitive transformation rules. Section V validates, through experimental results, MIG-based optimization and also presents and compares synthesis results to state-of-art academic/commercial tools. Section VI concludes the paper.

II. Background and Motivation

This section presents relevant background on logic representations and optimization for logic synthesis. Notations and definitions for Boolean algebra and logic networks are also introduced.

A. Logic Representation and Optimization

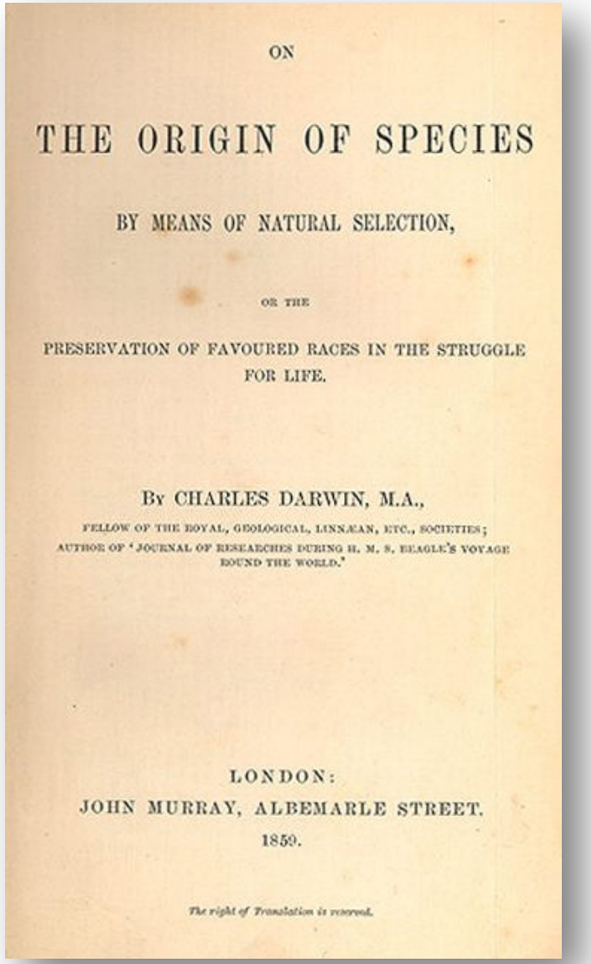
Virtually, all digital integrated circuits are synthesized thanks to efficient logic representation forms and associated optimization algorithms [1]. Early data structures and related optimization algorithms [2] are based on two-level representation of Boolean functions in *Sum Of Product* (SOP) form, which is a disjunction (OR) of conjuctions (AND) where variables can be complemented (INV). Another pioneering data structure is the *Binary Decision Diagram* (BDD) [6]: a canonical representation form based on nested if-then-else (MUX) formulas. Later on, multi-level logic networks [3], [4] emerged, employing AND, OR, INV, MUX operations as basis functions, with more scalable optimization and synthesis tools [4], [7]. To deal with the continuous increase in logic designs complexity, a step further is enabled by [5], where multi-level logic networks are made homogenous, i.e., consisting of only AND nodes interconnected by regular/complement (INV) edges. The tool ABC [8], which is based on the *AND-Inverter Graphs* (AIGs), is considered the state-of-art academic software for (large) optimization and synthesis.

We propose, in this paper, a new logic optimization paradigm that aims at extending the capabilities of modern synthesis tools.

SYNOPSYS®

Back To Charles R. Darwin

Things Don't Happen Overnight !



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

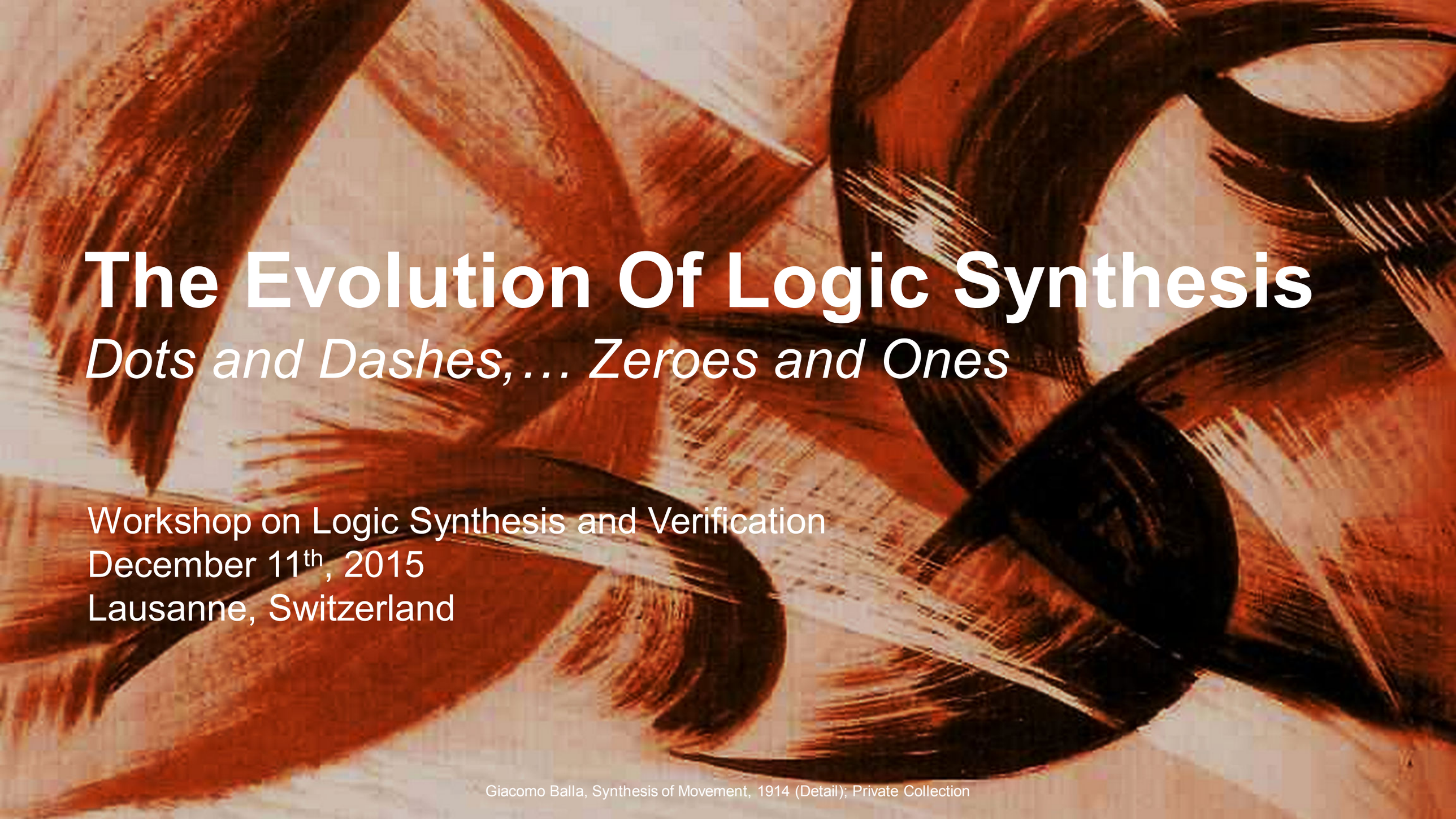
By CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

The right of Translation is reserved.

"It may be said that natural selection is daily and hourly scrutinizing …. every variation, even the slightest; rejecting that which is bad, preserving and adding up all that is good … silently and insensibly working […] at the improvement of each […]. We see nothing of these slow changes in progress, until the hand of time has marked the long lapses of ages, and then […] we only see that the forms of life are now different from what they formerly were."



The Evolution Of Logic Synthesis

Dots and Dashes,… Zeroes and Ones

Workshop on Logic Synthesis and Verification
December 11th, 2015
Lausanne, Switzerland

Giacomo Balla, Synthesis of Movement, 1914 (Detail); Private Collection