DESIGN OF MICROELECTRONIC INTEGRATED CIRCUITRY

William Bricken
September 2001

CONTENTS

There are four general phases to the production of integrated circuits (ICs):

1. Design
   converting an idea into a validated physical model

2. Fabrication
   converting a model into a physical circuit

3. Testing
   assuring that physical fabrication has not introduced error

4. Packaging
   converting a validated physical device into a salable product

The Losp/Pun suite of tools addresses only the first phase, Design. However, the generation of test vectors for Phase 3 is also accomplished by Losp/Pun design verification tools.

There are three general aspects to circuit design:

1. Conceptualization and modeling
   converting an idea into a formal model

2. Synthesis and optimization
converting a formal model into an efficient transistor network

3. Validation
assuring that design has not introduced error

The Losp/Pun suite addresses all aspects of design.

Conceptualization and Modeling

Circuitry begins as a mathematical model; the objective of *conceptualization* is to express a design idea in a language that can readily be converted into physical circuit behavior. Design ideas for ICs are naturally constrained to those tasks and processes which we know circuits can perform. *Modeling* involves identifying the best language to express the idea, and then actually converting the idea into a formal description which is sufficiently specific to allow the construction of physical realizations that perform the behavior intended by the design.

Unlike entertainment, marketing, and other creative industries, conceptualization in the microelectronics industry is drastically limited by the available formal tools and languages. Currently, these tools have evolved organically into complex assemblages which lack a coherent formal structure. As a consequence, formal verification tools are seen to be a separate product, one which is applied in concert with more dated tools. Current tools are not well suited for design of multilevel circuits since they lack some essential capabilities, especially minimization of logic networks. Losp/Pun provides some of these missing tools, including network minimization.

Networks of transistors can do very little; they can propagate binary signals, toggle the state of binary signals, store binary values, and sequence these events over time, when time is defined by a synchronous internal clock. The formal language which describes this range of behavior is *timed binary logic*. Timed binary logic is extremely limited in its expressability, which is essentially what can be said with sequences of signals selected from two states, positive and not-positive. Binary logic is the simplest formal symbol system of any interest at all, transformations expressed in common language as {NOT, AND, OR}. Timed sequences of binary logic are simply temporal replications of single uses of this simple formal system.

The dominant languages for expressing models of ICs are hardware description languages (HDLs), formal systems which are specifically designed to capture the behavior of physical transistor networks. These languages support the parallelism native to physical devices, and the hierarchical abstraction necessary for understanding and modeling massive replication of binary

transformations. HDLs are hybrid, expressing both mathematical structure and physical behavior. The entire conceptual design process is focused on generating an HDL description of the intended behavior of the IC.

Like HDLs, Losp/Pun expresses mathematical structure; unlike HDLs, it provides automated support for the other design phases. To achieve backward compatibility, Losp/Pun inputs and outputs conventional HDL and netlist specifications, thus it can be used as a black-box attachment to conventional design processes.

Reducibility and Replicability

The two factors which contribute substantively to the power of timed binary logic are *reducibility* and *replicability*.

The ways in which formal symbol systems are constructed and manipulated permit them to be translated one to another. Almost all formal systems can express binary logic, and many can be reduced to only binary logic. Technically, most formal languages are *computable*, that domain of functionality defined by timed binary logic. This means that many different formal systems could be used to express a given behavioral design, some examples being binary equations, binary matrices, schematics, and programming languages. The program Mathematica demonstrates that almost all computations of higher mathematics can be expressed by replicated binary logic.

The magic of microelectronics is that it can replicate simple binary behavior incredibly quickly, performing logical transformations billions of times each second. As well, all components of an integrated circuit act in parallel, at the same time. Thus every transistor in a million transistor network can produce a billion potential transformations each second.

Microelectronics rests on massive replication of simple binary actions. However, our conceptualization of these processes can be expressed in complex formal languages such as programming languages, finite state machines, logic equations, and hardware description languages.

Synthesis and Optimization

The objective of *circuit synthesis* is to generate a highly detailed model, sufficient to provide exact fabrication specifications. In contrast to conceptual design, logic synthesis incorporates both mathematical and physical models. At the synthesis stage, physical characteristics are introduced into the behavioral model built by the modeling process. The functional description, expressed in timed binary logic, is extended informally by models of physical processes such as geometric layout,

placement and size of wiring and connections, signal transmission speed, power consumption, and the many constraints of physical fabrication.

IC synthesis, once a design idea is expressed in a model, consists of identifying among many trade-offs which are most desired, and then modifying the formal model accordingly. Optimization of an IC is not a mathematical optimization in which a best solution is found. Rather it is a constant compromise between many competing design goals. All optimization is focussed on models of physical performance, such as size, speed, power requirements, and manufacturability of a circuit. These qualities do not have simple relationships.

The synthesis phase must provide step-by-step verification, assuring absolutely that no design changes or elaborations deviate from the behavioral specification. Synthesis must also verify models which deviate significantly from the formality which makes them tractable. To cope with this, designers usually divide synthesis into two supposedly orthogonal components, a technology independent phase, *logic synthesis*, and a technology dependent phase, *technology mapping*. Unfortunately this division is artificial and, for large designs, dysfunctional.

Criteria of Merit

Generally, optimization increases the merit of a design, the competing criteria of merit being:

Performance
: the speed and efficiency of computation

Area
: the number and layout of transistors and wiring

Testability
: the ease of verifying absence of manufacturing defects

Power
: the physical effort it takes to run the IC

Competitive marketplaces make performance important; cost of manufacturing makes area important; the huge cost of performance failure makes testability important; and the prominence of portable computers makes power usage important. These criteria can be summarized as: cost-of-fabrication is of critical importance to the profitability of an IC.

From a market perspective, the criterion of merit is *revenue life*, the length of time during which an IC product generates commercial revenues. In

microelectronics, revenue life has been rapidly shortening, with improved versions of products fabricated on monthly rather than yearly cycles.

Revenue life dictates that time-to-market dominates all other considerations, such as cost overrun, performance quality, and logic optimization. Time-to-market, in turn, creates these criteria of merit:

Rapid design
: converting conceptualization to a physical model quickly

Error-free design
: reducing errors and thus design cycles before fabrication

Verifiable design
: assuring that rapid design does not introduce error

Design for manufacturability
: designing with simplicity for high fabrication yield rates

Competition is sufficiently rigorous in the microelectronics industry that even with the above time-to-market constraints, the more technical criteria of merit listed above must also be met.

Due to complexity and rapidity, IC design requires sophisticated software tools. Losp/Pun is a software tool suite which has been specifically designed to meet time-to-market needs, while at the same time out-performing existing tools on technical criteria such as optimization of performance, area, and testability.

Verification

Prior to the costly fabrication step, IC models, both formal and physical, must be verified as meeting the design objectives. Each model, whether conceptual or logical or physical, is verified for every possible input configuration. *Verification* is intended to identify any design errors introduced during the design phase. The cost of correcting design errors prior to fabrication is relatively low, since models can be simulated in software and in reconfigurable hardware.

Verification can be achieved by two different methods: simulation and formal proof.

Simulation uses concrete test vectors to examine the behavior of a model exhaustively. Simulation applies different sets of input patterns while watching for the appropriate accompanying output patterns. The number of input patterns may be huge, for large circuits simulation may take days. As designs become more complex, simulation becomes more ineffective.

Very large designs require formal methods. *Formal verification* achieves the same result as simulation by using a model which provides transformations on unbound variables, on inputs which may have either binary value. Formal verification requires consistent formal models throughout the design phase.

Losp/Pun integrates formal verification into every transformation step, assuring continuously correct modeling.

Complexity

Today, the primary method for dealing with complexity is IP cores, fairly large pre-designed and verified functional components which can be assembled as build-blocks. Losp/Pun provides lower-level tools for the construction and contextual modification of functional cores.

Another prominent technique for management of complexity is *design decomposition*, breaking a design down into manageable components. Since conceptual design is usually expressed as component functionalities, such as data and control, a natural high-level decomposition is usually available. However, once the design is refined down to the structural level, the conceptualization gives way to detailed formal and physical constraints. Losp/Pun excels at identification of low and medium level structural components. Losp/Pun is not a top-down conceptual design tool, instead it takes a finished rough design prior to timing and optimization, and returns a design ready for physical modeling. Like other IC synthesis tools, Losp/Pun converts high-level descriptive languages into logic networks, automatically improving the design along the way. Unlike conventional tools, Losp/Pun provides

1) efficient path-oriented network minimization,
2) powerful abstraction capabilities,
3) integrated formal verification, and
4) rapid generation of candidate circuits.

An *exponential relationship* is one in which one quantity increases hugely while another makes only a small change. Exponential relations are *computationally intractable*, they are practical only for small collections. Input patterns have an exponential relationship to the number of input variables and registers in an IC. Adding a few variables can increase the number of possible input patterns enormously. Almost all synthesis, optimization, and verification algorithms are exponential, they are simply infeasible for large circuits.

Gigahertz processing rates make it infeasible to examine all possible states in sequential circuits, just as exponential algorithms make it infeasible to examine large combinational circuits. The history of synthesis,

optimization, and verification tools is one of heuristics and patches, approximations and fixes, addressing a mathematical puzzle which is impossible to solve efficiently. Thus, abstraction and hierarchical decomposition have become necessary tools to deal with overwhelming complexity.

The two other dominant methods for coping with circuit complexity are committing hundreds of person days to a design and accepting poor synthesis results.

Varieties of Abstraction

A model is an abstraction, it expresses with symbols what happens in physical reality. Models rest on selected blindness, intentionally ignoring some constraints in order to express others in a form simple enough for our understanding. Models used in the synthesis process fall into three groups:

Architecture Models
: the set of computational operations

Logic Models
: the formal description of signal behavior

Geometry Models
: the circuit constrained by a physical layout

Each group also embodies three perspectives on its model:

Behavioral Perspective
: the functionality expressed formally

Structural Perspective
: the interconnection of components

Physical Perspective
: the transistor network in silicon

We thus have many phases of abstraction during design, each with its own languages and each with its own set of required skills. Such is complexity. Examples of these languages and tools are presented in the following chart:

	architecture	*logic*	*geometry*
behavioral	software	state machines	Losp/Pun
structural	block diagrams	schematics	logical effort
physical	placement	routing	fabrication

Table I. Types and Examples of Abstract Models

Behavioral Synthesis

Behavioral/architectural synthesis is the high-level design technique which defines the set of computational objects and their interrelationships. This includes identifying the necessary resources used by necessary objects, scheduling and timing of resource use, and binding resources to processes. Connectivity is expressed as *data paths*; computation, as *control units*. The details of each is specified by the structural perspective.

Computer programming languages can provide high-level abstraction. Program data structures model the intended architecture. Program execution models the intended behavior. Programmed models can be designed at any level of complexity or of abstraction. The data structures and processes which simulate hardware behavior use the instruction set of the general purpose processor. This technique allows design of large scale abstractions and wide bus-widths while avoiding the complexities of bit-level processes, gigahertz clock cycles, binary representations, physical behavior, and transistor networks. Since programs are processed by a serial vonNeumann processor, they do not simulate hardware behavior in any way.

Behavioral/logic synthesis is the specification of intention using formal languages: state machines, logic networks, Boolean equations, truth tables, and HDLs. Design optimization occurs at this level. The formal systems are high-level descriptions which map to low-level binary logic. They literally define the conceptual vocabulary of the designer.

For example, a finite state machine (FSM) is a collection of states and of specified transition conditions which permit change from state to state. A standard example is the logic in a vending machine which tracks how much you have put in, how much you owe, and in what ways you can bring the two into balance. Expressed as a graph, this FSM would have states/nodes corresponding to all the combinations of acceptable coinage and changes/links corresponding to what happens when another coin is deposited.

This high-level description translates very awkwardly into large networks of logic gates and registers, in other words, into highly inefficient circuitry. Optimization is mandatory since high-level conceptual descriptions do not efficiently translate into low-level transistor networks. Optimization is the process of modifying a formal description of the structure of a specification within the rulers of that formal language which maintain functional invariance

Behavioral/geometry synthesis to date is undefined by EDA tools. Losp/Pun is such a behavioral geometry, it defines logical behavior in terms of geometric connectivity.

The boundary mathematics implemented within Losp translates a conventional behavioral specification into a configuration of nested enclosures, or

boundaries. Pun translates this into a graph. Losp/Pun then transforms the geometric connectivity of the graph in ways which achieve the same results as the conventional tools of behavioral/logic synthesis. Given a limited selection of criteria of merit for a design, Pun uses Losp to generate a candidate circuit structure which closely aligns with the desired criteria. Formal verification is maintained throughout, as is a close mapping to a realizable circuit (this is in contrast to BDD technology). Thus, Losp/Pun uses geometric logic to achieve circuit optimization.

Structural Synthesis

Structural/architectural synthesis defines the functional interconnections between structural components, how the behavioral model is assembled as a model of physical processes. Flowcharts and block diagrams are typical tools of this type.

A flowchart specifies, often in detail, the mathematical transformations performed on abstract data structures. Flow charts identify the logical flow of control, as well as abstract processes such as numerical addition. Tightly constrained programming languages can be used to express flowchart dynamics. Transforming flowcharts into logic networks is fairly direct; it is the quality of the flowchart architecture which defines the quality of the accompanying circuitry.

Structural/logic synthesis identifies the logic network, the gates and wires which constitute the IC fabrication model. HDLs and more visual schematic diagrams depict logic processing networks. This aspect is the one most probably visualized when we think of a designer toiling over sets of arcane symbols.

A schematic is a diagram which shows logic gates, and the wires which connect them. Schematics are blueprints, but not of the fabrication process. They are a low-level picture of the mathematical functionality, accented with most of the logical accoutrements required for physical functioning (input ports, sources and grounds, clocks, busses, heat sinks, and the like). All structural/logical tools are hybrid, a somewhat challenging mix of logic, wiring, and physical necessity. This is perhaps the most awkward of the synthesis techniques, somewhat due to the difficulty of the modeling task, somewhat due to weakness inflicted by hardware evolution.

Structural/geometry synthesis includes converting the abstract logic network into a network of physical devices, specifically a transistor network. Here the non-logical restrictions placed on a logic network by a physical topology are added. Primary examples of non-logical geometry are the FPGA architecture, which provides a uniform and configurable substrate, and ASIC sea of gates architecture, which also provides a uniform substrate, but one that is physically wired during fabrication.

Logical effort is a technique which identifies efficient relationships between logic and transistor networks. Given a desired functionality expressed by a logic network, logical effort separates the constraints placed on a transistor layout by the logic, and those attributable to the physical aspects of transistors such as power and delay. It then computes a close-to-optimal transistor geometry which achieves the logical intent.

Physical Synthesis

Physical synthesis, in each of the three modeling groups, involves converting all other abstract and formal models into a model which can be physically fabricated. Converting a model to a physical realization very often undermines the properties of the abstract model. Thus, it is now very common to reduce a model to physicality using relatively expensive reconfigurable hardware devices, prior to custom fabrication.

Physical/architectural synthesis is the alignment of a formal structural description with the physical form of hardware. One dominant aspect is that of *placement*, determining where in the physical layout each abstract component should be placed. Naturally, physical placement is highly determined by the physical architecture which supports the functional circuit behavior.

As a simple example, it is usually preferable to place components which share local i/o signals in close physical proximity. Another example is mapping logical structure onto physical architectures which are not based on logic gates. FPGAs provide look-up tables (LUTs) which compute any function of a given number of variables. A four-input LUT is insensitive to the collection of logic transforms, placement instead focuses on groups of four signals transformed by a set of gates.

Physical/logical synthesis determines how signals are physically connected. The paramount aspect is *routing*, determining which wires in a physical substrate are used to connect which logical processes. Placement and routing are generally considered to be tightly coupled; where are component is placed and which wires it uses are determined concurrently. During fabrication, necessary routing resources directly influence the number of fabrication layers, since wires which cross in two dimensions must be separated in the third dimension.

In FPGA hardware architectures, the computational substrate provides a generic wiring resource. In today's large designs, wiring is a scarce commodity. Components are placed so as to use wiring judiciously. Sometimes lack of placement locations forces long wires between functionally adjacent components. When wires differ in length, the time that it takes for a signal to traverse each wire differs. This interjects significant difficulties in the coordination of timing across the circuit. Thus routing is both

critically determined by the available architectures and critical to efficient functioning.

Physical/geometric synthesis defines the configuration of fabrication tools, such as the construction of masks and layers, which define the physical template used to manufacture a chip.

Abstraction and Physicality

Traditionally, behavioral and structural design have been separated from physical design, under the assumption that the physical design process mimics the behavioral intention. This assumption has been supported in two directions: physical components are built to replicate simple structural descriptions, and structural models are designed to align with simple physical behaviors.

Today, complexity has driven abstraction and physicality together, IC conceptualization must include aspects of physical fabrication from the start. A difficulty is that including physical synthesis in the languages and tools of conceptual design undermines the intended use of the tools, that of capturing design ideas in understandable formal models. The physical behavior of circuits, in return, is poorly expressed by symbolic models. Optimizations gained by manipulating formal models, behavioral and structural, are often lost when these models are converted to a physical basis. Thus, the problems addressed by EDA software have migrated, requiring new approaches to different optimization criteria and constraints.

Design requires efficient and effective automated optimization; although designers are excellent at high-level description, the sheer volume of lower-level details and the substantial difficulty of using existing optimization techniques makes this phase well-suited for advanced automation. High-level descriptions connect to human understanding; low-level descriptions achieve technical efficiency. Losp/Pun is a low-level tool.

Losp/Pun accepts behavioral and logical descriptions of IC conceptualizations. Its input is a design expressed formally, but prior to introduction of any physical or structural constraints. It then provides, in interaction with a designer, the possible circuit structures which maintain validity. Whenever a design must be fine-tuned, it provides formal local transformations to achieve design goals. Losp/Pun can move rapidly across many different structural varieties while holding functionality invariant. Further, the functional networks generated by Losp/Pun provide strong guidance to physical placement and routing. Losp/Pun integrates all phases of design, from conceptual modeling through verification.

Testing after Fabrication

Fabrication of circuitry at very small scales is prone to physical defects. Dust, substrate irregularities, and random microscopic variations within the fabrication process can each undermine the integrity of the sub-micron wires and transistors. Every manufactured circuit must be exhaustively tested to assure absence of fabrication errors. The *yield* of a fabrication process is the percent of error-free chips produced. This may vary from 10 to 90 percent, depending on the complexity and size of the circuit, the fabrication technology, and the details of the fabrication process.

In order to physically test a manufactured circuit, every transistor in the circuit must be exercised, its outputs compared to what is expected. Fabrication defects have dominant characteristics; since they are essentially random, they are often isolated in single gates. The simplest form of testing is to test each gate individually. A more complex testing regime examines gate performance in pairs and in groups, in effect identifying defects which occur only during interactions between sets of gates. Exhaustive interactional testing is intractable, it is too complex to achieve. However, known defect types for particular fabrication processes can be targeted for testing regimes.

Aside for a physical mechanism which permits submission of input test vectors, the test vectors themselves must be determined and constructed. These can usually be constructed using logic synthesis tools.

Optimization of circuit structure is mandatory for testability. In particular, reconvergent paths, those which split at some internal output and then rejoin as inputs to the same gate, are untestable. Optimization can remove or reduce reconvergent paths.

Losp/Pun optimization constructs testable circuits. It also constructs test vectors to exhaustively cover the possible states, and errors, for each gate.

Diagrams

Design flow is essentially sequential, although latter steps must often be considered in earlier design choices. Each design step is verified for correctness. Should a design error occur, earlier stages are revisited iteratively.

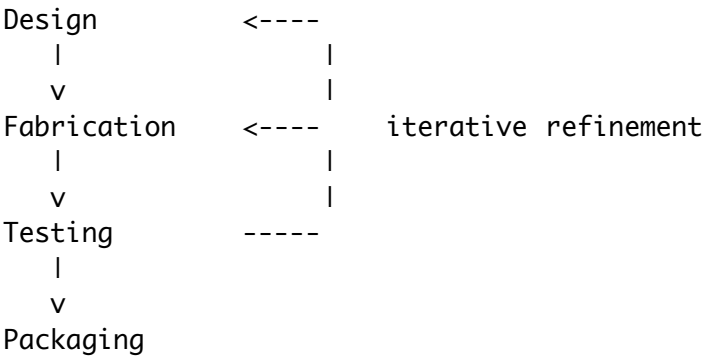

Figure I. IC Production Phases

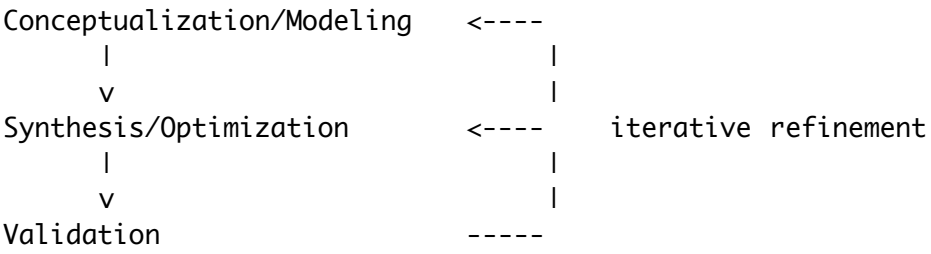

Figure II. IC Design Phases

References

G. DeMicheli (1994) *Synthesis and Optimization of Digital Circuits*, McGraw-Hill.

G.D. Hachtel and F. Somenzi (1996) *Logic Synthesis and Verification Algorithms*, Kluwer.

C. Heater (2001) Personal communication.

R.H. Katz (1994) *Contemporary Logic Design*, Cummings.