LOSP SYNTHESIS SYSTEM: TECHNICAL DESCRIPTIONS

William Bricken
August 2004

CONTENTS

LOSP TECHNICAL DESCRIPTION

The Losp Synthesis System is a suite of three hierarchical software tools. At the core, the *Boundary Logic Engine* provides Boolean minimization of logic networks. The *Design Engine* provides design-specific transformations of circuit structure. The outer layer of functionality, the *Application Interface*, provides input and output of netlist specifications, batch-mode processing, test-vector verification, statistical analysis, and interfaces to CAD and other interactive systems.

The Losp suite takes the netlist representation of a design as input and returns a functionally equivalent netlist with improved delay and area performance as output. Losp performance improvement can be applied at any step of the design tool-chain for which a netlist or other formal description is available, and is general across types of designs.

Losp consists of hundreds of boundary logic algorithms, and dozens of parameterizations. The tools include:

• netlist parsers for HDL languages such as Verilog, VHDL, and EDIF

• independent reduction engines for logic, area, and delay reduction

• TSMC logic library mapping

- course-grain parametric control that allows a designer to choose a preferred trade-off between logic area and signal propagation speed
- fine-grain parametric control that allows a designer to customize the performance of any cell group in a design
- fine-grain filter control that allows a designer to select both cells and transformations for reduction processing
- automated push-button control that reduces design iterations and provides competitive design performance with low effort.

A DESCRIPTION OF BOUNDARY LOGIC

Boundary logic is a non-Boolean form of logic that emphasizes *minimality* in representation and in transformation effort by using only one operator.

A Unary Calculus

Boundary logic is *unary*, based on one concept, rather than binary, which is based on two concepts, such as 0 and 1. The single concept is a boundary, or container, ( ). Conventional logic distinguishes between two values, True and False. In contrast, containers distinguish between two spaces -- inside and outside -- using only one value, the presence or absence of a boundary. Although thinking of logic as containment relations is new and unfamiliar, everything that can be done with conventional logic expressions can also be done using container boundaries.

Conventional Boolean logic can also be expressed using a minimal basis of one binary operator (either NOR or NAND); however, combinations of the single Boolean operator explode in size, generating clumsy descriptions and slow algorithms. Boundary logic overcomes the explosion of form that makes conventional minimal bases undesirable. The boundary concept of *spatial distinction* provides a single-operator minimal basis that is accompanied by highly efficient transformations.

New Data Structures

The data structure underneath almost all advanced logic synthesis tools is the Binary Decision Diagram (BDD and variants). Historically, the first EDA tools, based on two-level logic networks, were very inefficient for larger designs. Since the early 1990s, BDDs have gained dominance as a better and more efficient approach, since they handle multilevel networks of logic gates.

Boundary logic improves upon BDDs. Many different conventional logic expressions condense into the same boundary logic form -- a boundary form can be read as many different logical expressions. This makes the container-based data structures within Losp formally more efficient than BDD and other Boolean methods.

The Losp boundary logic representation can be directly read as a conventional circuit and directly manipulated as a Boolean algebra. Losp thus offers both new capabilities and new efficiencies in achieving those capabilities.

As a visual example, consider the conventional and boundary logic representations of a simple logic expression

`(NOT a) AND (b OR c)`	***Conventional logic***
`[a [b c]]`	***Boundary logic***

Above, a bracket is used to represent the boundary logic container. Conventional logic uses three different operators; as a consequence, there are many transformation rules that switch between equivalent forms of each logical expression. Boundary logic uses only nested containers, a more succinct form that requires less computational effort to transform.

New Computational Concepts

The minimalist calculus of boundary logic reduces the complexity of computational operations by introducing new concepts of virtual forms and transparent boundaries. *Virtual forms* are those that can be deleted at any time, although their presence can catalyze other reduction operations. *Transparent boundaries* are those that can be treated as not present with regard to specific reduction operations.

Some boundary logic transforms are insensitive to the depth of nesting of a logical expression, that is, to the layers of logic that constitute a multilevel logical network. This innovation effectively addresses an outstanding problem in multilevel logic optimization, that of the absence of efficient Boolean techniques for handling complex nested logical structures.

Losp Technical Advantages

The boundary logic computational engines in Losp provide distinct technical advantages for logic synthesis and optimization, including:

1. A simple, integrated, and comprehensive theory guides all possible network transformations. Structural patterns are constructed from a single component type (i.e. the container), transformation rules for different cell and logic types are not necessary.

2. Boundaries condense the conventionally separate ideas of logic gates and interconnecting wires, allowing fluidity in meeting logic and connectivity design goals. Boundaries between spaces *both connect and separate*. Conventionally wires connect while gates separate; however in boundary logic, a container can be used to represent both wires and logic gates, in effect overloading the single operator. The separating aspect of a container can represent gates, while the connecting aspect -- a boundary is the shared border between two spaces -- can represent wires.

3. The duality relations in logic (True/False, And/Or, ForAll/Exists) are absorbed into the single boundary concept. Transformations such as DeMorgan's Laws, which switch between logical And and Or, are not required in Losp.

4. Reduction can occur *at a distance* within a circuit, it is not necessary to traverse a signal path through logic gates in order to reduce a path. This permits efficient multilevel logic reduction.

5. Reduction is via *deletion* rather than rearrangement, making all algorithms more efficient and requiring less space to implement.

The bottom line is that the Losp core engines consist of a collection of unique and powerful CAD/EDA algorithms that out-perform other existing techniques.

LOGIC SYNTHESIS

Logic synthesis is a technical, highly mathematical field that includes the following areas:

- equivalence checking, tautology proving
- network decomposition and transformation
- redundancy removal and optimization to specified criteria
- functional verification and model checking
- Boolean factoring and constraint reasoning
- test vector and BDD generation
- technology mapping
- management of complexity.

Losp incorporates innovative algorithms for each of these areas.

The goal of logic synthesis is to construct an *abstract model* of design functionality with desired physical performance properties. Abstract models are used throughout the circuit design process. Most of these models are based on graphs, and include netlists, Boolean networks, state diagrams, transition tables, and dataflow and sequencing graphs.

As designs shrink in size, the interdependence between specification, network structure, and physical layout becomes pronounced, so that it is no longer productive to treat logic synthesis as an independent step that addresses abstract models only. Logic synthesis must now incorporate the behavioral, structural, and physical forms of a design.

Behavioral synthesis is the conversion of a design specification into an abstract logic network. *Structural synthesis* focuses on the minimization of the flow of data and control through the network. *Physical synthesis* can involve the selection of transistor technologies and the physical manufacturing processes.

The Losp synthesis system is designed to address the entire design methodology. The current prototype implementation is focused on structural synthesis, and does not yet include behavioral synthesis. Losp includes some technology mapping capabilities, these need to be expanded to cover the diversity of hardware architectures.

Structural Optimization

A given functionality can be expressed using many different but equivalent circuit structures. *Structural optimization* is finding the structure for a given functionality that best suits the architecture and resources of the execution hardware. Circuits may be optimized with respect to several technical factors including:

- die area and number of transistors
- wiring and interconnect
- timing and propagation delays
- power consumption
- noise and interference

Optimization involves search through a huge design space with dozens of parameters to consider. Due the complexity of the logic synthesis task, often particular parameters are bounded or held constant. Thus, typical trade-offs include minimization of area for a given delay, and minimization of delay for a given area. As well, a design is constrained by available resources (such as ALU functionality, memory size, and available connectivity and interface wiring), by i/o format and timing, by performance requirements, and most importantly, by manufacturing costs.

Losp provides automated design control of several common parameters, including:

- functional behavior (usually held invariant)
- silicon area required by logic gates
- number and interconnectivity of wires (fanin and fanout)
- signal transversal time (critical path)
- pipelining (logic chunking)
- technology mapping (choosing library components)

A unique capability in Losp is *iterative Boolean minimization*, being able to algorithmically construct and modify design components to meet specified criteria. Minimization is a challenging problem, essentially asking: given functional equivalence, which structural reconfigurations of a circuit are preferable? Losp converts a logic network to a minimal standardized form that emphasizes deep nesting of gates, rather than the shallow nesting of conventional standardization. Losp then makes many small, iterative, formal constructive modifications to the minimal version to achieve design specifications.

Losp quickly generates designs that are close to ideal, but not necessarily the exact best (this is known as a *satisficing solution*). The constraint-based algorithms "tighten" around sets of good solutions, providing the engineer with automated access to sets of desirable design trade-offs. Importantly, all Losp transformations guarantee validity of the functional design.

VERIFICATION

"Logic verification belongs to the set of most difficult problems in the field of computer-aided circuit design."
[Kunz, *Reasoning in Boolean Networks*, p.164]

Verification refers to a body of techniques used to assure that changes to the structure of a design do not change the intended functionality of the design. Verification tools fall into two broad categories: approximate and formal. Approximate verification includes test vectors and simulation. Formal verification includes deductive reasoning and model checking.

Approximate Verification

Over the last 30 years, the EDA community has used exhaustive testing to verify the correctness of designs. In the *test vector method*, every possible input vector is run through the physical circuit and checked against expected output. Generation of test vectors and the construction of a testbench are both difficult and time-consuming. The number of possible test vectors is often huge, making exhaustive testing impractical. Instead, a probabilistic approach is used to verify, say, 95% of the circuit behavior. Thus, test vectors are a means of approximate verification.

Another approximate verification technique is simulation. In the *simulation method,* the abstract model of the design generated by logic synthesis is run using test vectors prior to the design being committed to silicon.

Approximate methods require a golden version of the design that is known to be correct. The behavior of a new version is checked against that of the golden version. Of course, the golden version itself must also be verified, but this cannot be achieved with approximate techniques.

Approximate methods use input values in an experimental fashion. Simulation applies to the *design model* while test vectors apply to the *fabricated circuit*. Neither provide the necessary certainty, a guarantee of correct behavior. Commercial release of a design that contains even the slightest error is vastly expensive, since every faulty chip must be recalled and replaced. In sum, approximate verification is no longer an adequate alternative.

Losp incorporates both approximate verification methods for compatibility only, since the Losp engines are formal and do not directly use approximate verification. The internal Losp model serves as a logic simulator for both complete and partial input vectors. With partial input, Losp reduces the design to a minimal form that excludes the bound inputs. Losp algorithms generate test vectors by determining the priority of unbound inputs in determining the final output.

Formal Verification

Formal verification provides complete testing by deriving behavioral consequences from a complete behavioral description. The behavioral specification serves as the golden standard; attributes of this standard are validated through symbolic proof.

Complete testing by exact (test vector) methods dominated the algorithms of the '70s. These were efficient only for flattened, two-level netlists, the Sum-of-Products (SOP) form, not for multilevel logic networks. Current formal verification techniques that address multilevel logic networks include deductive reasoning and model checking.

In *deductive reasoning*, an axiomatic model of behavior is specified, and correct behavior is proved from these axioms. In *model checking*, formal properties about design behavior are asserted as constraints on the performance of a model.

Loosely, a model is an abstract description for which specific properties are asserted. A model checker exhaustively examines the input vectors of a design to determine if design behavior has the specified properties or meets the specified constraints. An example of a model constraint:

"The signal never takes more than 7 clock cycles to traverse from input to output."

Most new verification tools are model checkers. Although model checking still requires test vectors, the exhaustive set of test vectors can be a much smaller, since model checking verifies classes of behavior, and collections of test vectors rather than individual test vectors.

Specification languages for model checking are new, obscure, and don't scale well. It is difficult to know what the appropriate properties are and what specific locations in the circuit a constraint applies to. In general, model checking systems are very difficult to use, and deductive reasoning systems are even more difficult to use. Identifying appropriate constraints requires a high degree of skill and familiarity with formal specification. Here the EDA industry is discovering what the AI industry discovered 20 years ago: useful rule bases are hard to design.

Losp is a formal, algebraic system. The deductive tools within Losp can provide complete deductive reasoning and model checking. The specification of appropriate constraints to fully characterize the behavior of a design remains a difficulty.

HANDLING COMPLEXITY

A dominant design concern is chip complexity -- millions of gates packed into the size of a postage stamp. Modular design is mandatory, as are scalable tools that are relatively easy to use.

Modularization

Huge architectures, such as SoCs, must be partitioned. Designers define functional modules in order to construct reasonably sized design partitions. Module libraries provide pre-built functions such as multipliers and comparators. The trend is toward large pre-built modules that perform significant tasks, such as bus protocols and signal processing filters. And designs are naturally constructed using top-down refinement.

A unique capability of Losp is automated, *bottom-up modularization*. Due to the simplicity of boundary logic forms, Losp algorithms address both predefined and *discovered* hierarchical partitions, or abstractions. All of the Losp transformations can be applied directly to patterns and modules that have been identified, without necessarily flattening them to the lowest common denominator of logic gates.

Losp provides powerful tools for hierarchical analysis and pattern abstraction. Abstraction comes in many varieties. Specific abstraction methods in Losp include:

- *cell abstraction*: a small pattern is identified and constructed as a separate logic cell.
- *module abstraction*: a large, repetitive pattern is identified and constructed in a separate module.
- *atom abstraction*: symmetries across inputs are generalized.
- *generator abstraction*: functions are generalized to apply to any bit-width, and are generated automatically.
- *vector and matrix abstraction*: signals processed by the same functions are bundled into collections with bit-widths greater than one.
- *partition abstraction*: logic and interconnect clusters are structured from parametric templates that map onto constrained physical resources.

Ease of Design

Losp supports a simple interface that is easy to use. Features of Losp performance that enhance usability include:

- comprehensive and integrated synthesis capabilities
- few iterations to reach design specifications
- course and fine-grain control of engine performance
- predictable output
- fast processing time
- ease of extensibility to new architectures and feature scales
- compatibility with existing techniques and skill sets
- less design work to do

Most of these interaction advantages are achieved by the Losp core engine designs. They have yet to be built into an engineer's interface.