ILOC DELAY OPTIMIZATION ROADMAP

William Bricken
October 2003

ILOC was originally developed as an area minimization tool for combinational and sequential netlists. There is commercial motivation to extend ILOC to delay optimization for two generic capabilities: delay optimization without regard for area, and delay optimization while maintaining low area. Both capabilities require a deep integration of area and timing decisions within the ILOC reduction engines. This memo outlines the steps to achieve competitive delay optimization in ILOC.

EXECUTIVE SUMMARY

Capabilities

The ILOC implementation of Boundary Logic provides an integrated and complete approach to network transformation, including area and delay minimization. ILOC improves optimization performance both through more efficient algorithms and through an integrated theory that organizes the currently haphazard and scattered approaches of competitive products.

Roadmap

1. Extend ILOC logic and network transformations to include delay reduction choices.
2. Add specialized delay reduction tools to the network reduction engine.
3. Incorporate a sophisticated timing model into ILOC.
4. Implement top-level delay reduction control algorithms.

Level of Effort

Two senior and four junior software personnel over a development timeline of eleven months. The product is an alpha release ASIC optimization capability usable by EDA engineers. Timelines and resource estimates are attached.

ILOC COMPARISON TO SYNOPSYS FOR DESIGN AREA REDUCTION

The objective of the ILOC comparison study is to generate reliable data that documents a differential ILOC capability to reduce the area of ASIC circuit designs while holding timing constant.

Preamble

ILOC is a system consisting of hundreds of Boundary Logic algorithms, and dozens of parameterizations. Many netlist manipulation algorithms and all core Boundary Logic reduction algorithms are complete, validated, and extensively tested. Newer algorithms, especially those pertaining to delay optimization, are new, partial, and still potentially buggy. Some techniques are implemented schematically, and require hand steerage for accurate application. Others are purely experimental.

The current study is the first comprehensive test of ILOC against the Synopsys commercially developed and field tested product. Thus it is appropriate to take an incremental approach, seeking to identify and calibrate the comparative strengths and weaknesses of ILOC tools.

In the past, we have made several methodological errors that are corrected in the proposed study. In particular, we need to separate different aspects of what is to be proved. We first need to provide functional equivalence; for this it is necessary that I can verify ILOC transformations within ILOC using test vectors. Another factor is demonstrating that ILOC outperforms Synopsys. This factor has components of area minimization, delay minimization, and combined area/delay minimization. Methodologies include ILOC and Synopsys performance starting from a common non-optimized basis, and ILOC as a post-process after Synopsys delay optimization. Another factor, separate from performance, is application of ILOC to large and complex industrial designs. Yet another factor is the usability of ILOC and ILOC results by others, including common data transfer formats. Another factor is technology mapping, including the selection of cell libraries. Finally, we need to distinguish our current stage of establishing a case from that of empirical validation of that case using random testing.

Scope of Study

We have a few data points comparing ILOC to Synopsys. This study extends the number of data points to a dozen or more. ILOC transforms structure and has the capability of identifying almost all of the common structures used in circuit design. These structures apply to all functionalities; large design is simply combinations of these structures. The set of functions, designs, and partial designs that we ware comparing cover almost all types of circuit structure. Each tested design is included to provide data for a specific purpose.

Study Parameters

Since we are interested in a pragmatic comparison, we will use the full TSMC logic cell library, permitting Synopsys to engage in whatever transformations it chooses to achieve

Methodologies

We have two comparative methodologies: separate optimization by both systems, and post-processing optimization. Separate optimization is good for establishing and comparing absolute capabilities; post-processing is useful in our case because Synopsys can provide its expertise at delay optimization, and ILOC can then apply its expertise at area optimization while holding delay constant. Schematically:

Separate Optimization

```
                              ==> optimized-by-Synopsys
                            /                            \
         non-optimized-RTL                                 ==> compare
                            \                            /
                              ==>   optimized-by-ILOC
```

Post-processing Optimization

```
                               ------------------------
                             /                          \
         optimized-by-Synopsys                            ==> compare
                             \                          /
                               ==>   optimized-by-ILOC
```

At this time, ILOC is not competitive with Synopsys for delay optimization, so post-processing should demonstrate that ILOC can still provide significant area optimization and competitive timing, given a good timing model and delay algorithms.

DELAY OPTIMIZATION IN ILOC

ILOC uses a transformation system based entirely on Boundary Logic. The ILOC software consists of three independent components:

-- the core logic reduction engine
-- the network reduction application engine, and
-- the technology mapping interface.

For delay optimization, all three components need to be integrated. No new capabilities need to be added to the core engine, however optimization decisions made within the core must be guided by new control systems based on a sophisticated timing model.

Goal-Driven Reduction

The core ILOC reduction engine uses six transformations to achieve network reduction. These formal rules define all functionally invariant transformations of any given network structure, including all possible transformations to achieve delay reduction.

ILOC is an algebraic system, which means that all transformations can be applied in two directions, using deletion for reduction and using creation for construction.

```
                            reduce area <==> increase area
A = X Y Z                       delete  <==> construct
                          increase delay <==> decrease delay
```

Applying Boundary Logic transformations in the reductive direction achieves area reduction; applying them in the constructive direction achieves the full range of existing (and possible) delay reduction transformations. Reducing area increases delay, while increasing area reduces delay. The primary difference between area and delay reduction in ILOC is the underlying model that determines which transformations are to be applied in which directions.

Almost all conventional delay reduction techniques map directly onto reversing the six basic internal ILOC rules used for area reduction. Each rule can be applied independently and locally to part of a previously reduced network in order to reduce network delay. Each rule can be characterized by an area/delay trade-off, the direction of application determines which measure is reduced. The current version of ILOC can be extended into a competitive goal-directed delay reduction tool by connecting a sophisticated timing and signal propagation model to the constructive use of existing transformations. By providing both area and delay goals, ILOC transformations can be applied in both directions to achieve concurrent area and delay optimization that meet specific design objectives.

Thus, to extend ILOC to delay reduction, the existing Boundary Logic equations must be associated with a direction of application, and the choice of direction must be associated with a delay reduction timing model.

General Plan

1. Extend logic and network transformations with delay reduction metrics.
2. Add specialized delay reduction macros to the network reduction engine.
3. Incorporate a sophisticated timing model.
4. Implement top-level control algorithms.

The general plan is to associate each logic and network transformation with a delay-related cost, and then to use each transformation to achieve prespecified timing and area objectives. The internal transformations are few in number, simple to identify, and localized in effect; thus goal-driven transformation to reduce delay, and to reduce delay while maintaining a small area, requires relatively direct and simple control and measurement algorithms. Many decisions about timing interact, however ILOC provides a canonical internal format that can serve as a baseline for selecting globally advantageous transformations.

The logic and network reduction engines first convert a design into simple internal data structures, called parens forms. Then transformations are applied to the parens forms to achieve structural goals such as area minimization. Finally the resulting parens network is mapped back into a conventional gate library. Certain simple cells, such as NAND gates, possess very desirable delay characteristics. These can be pre-selected during logic transformation, thus achieving technology mapping during logic reduction. Network transformation then serves to integrate these pre-selected types. Importantly, this process does not involve search or trail-and-error, as do existing high-performance competitive approaches.

To achieve area reduction, it is sufficient to identify which gate library cells are area efficient, and to map the simplified internal ILOC structures into these cells. This is accomplished as a final step in the area optimization process. For delay reduction, library cells have specific intrinsic delays, however the network topology -- how the cells are interconnected -- also contributes significantly to delay. Thus technology mapping for delay must be integrated into the logic and network reduction decisions, it cannot be applied as an independent terminal process. Network topology becomes more dominant as process sizes shrink.

Top-Level Control

Top-level delay reduction control algorithms will apply the ILOC fine-grain cell transformations to each cell. Each Boundary Logic transformation, applied constructively, changes the connectivity of a single cell in the entire design. Delay minimization can be accomplished by successively selecting each high delay cell in the network, transforming it to decrease its delay, and iterating over the entire network.

Identifying which cell to transform is quite simple. The control algorithms can be guided by a few basic principles, applied in order:

-- Many cell types are undesirable, they can be converted into desirable types. Desirability is determined by area and delay goals. For minimization of delay, only XOR2, NAND-N and INV cells are needed, with an occasional NOR-N to accommodate inverters.

-- Almost all inverters can be incorporated into other desirable cells.

-- Cells with high capacitance can be buffered or replicated, depending on area/delay goals.

-- NAND3 and NAND4 cells can be trimmed to provide a more efficient NAND2 critical path.

-- Finally, to shorten critical paths, desirable NAND and XOR cells can be expanded through distribution.

Note that for delay reduction, changing drive strength is not necessary. The top-level control sets all drive strengths at maximum, and then after delay goals have been achieved, those cells not on the critical path can be reduced in drive strength to save area. Similarly for area reduction, all drive strengths are set to minimum, and if delay needs to be improved, the drive strength of cells along the critical path is increased.

The ILOC top-level control sequence replaces the plethora of partial techniques now used in competitive products. Due to the theoretical organization of the transformation rules, these steps are more efficient than existing EDA tools, both for identifying possible transformations that achieve design goals, and in the amount of computational effort required.

ILOC Capabilities To Be Added

A sophisticated timing model for signal propagation must be developed to guide the goal-directed transformations. This model needs to include information about how signals traverse a network of library gates, capacitance and parasitic effects, wire length models, etc. Sophisticated timing models are generally known and available, we will need to attract some expertise in this area.

Delay-driven transformation must be accompanied by new, specialized delay reduction tools within the network reduction engine. These tools are relatively easy to express and to implement in Boundary Logic. In general they are macros, combinations of two or three of the basic ILOC transformations. The technical commentary contains some examples.

TECHNICAL COMMENTARY

Technical Advantages

Boundary Logic is exceedingly elegant and efficient, providing ILOC with a competitive advantage for area reduction. Boundary Logic confers the same advantages to network transformation for delay reduction. The technical advantages of Boundary Logic can be summarized as:

1. Structural patterns are constructed from a single component type, rules for different cell types are not necessary.

2. Gates and wiring are a single concept, permitting path-oriented transformation and fluidity in meeting logic and connectivity design goals.

3. Less computational effort is required to achieve conventional objectives, including simpler representations that permit identification of advantageous transformations, and more powerful and efficient transformations.

4. A simple, integrated, and comprehensive theory guides all possible network transformations. This advantage provides a substantive competitive advantage since current commercial and academic techniques consist of a haphazard collect of unrelated ideas and algorithms.

Boundary Logic Delay Reduction Transformations

The current state of the art in delay minimization consists of a collection of apparently unrelated network modifications that are applied quite haphazardly. No general theory of delay optimization exists. Sutherland's Logical Effort theory, for example, is a set of rules of thumb that can guide good design, however each type of logic gate has a different set of desirable characteristics, the consequences of the Logical Effort rules are calculated separately for each design element, and pre-computed tables that incorporate good design choices are used to guide decisions. In Boundary Logic, there is only one type of gate (the parens), transformation patterns are very limited and easy to identify, and the transformations have quantified local consequences for both area and delay.

A significant aspect of delay optimization is the appropriate placement of inverters. In Boundary Logic, each parens can be interpreted as an inverter. Thus the Boundary Logic transformations incorporate optimal inverter placement directly within the basic boundary transformation equations. For example, the propagation of an inverter through a MUX is:

```
            (((A) (B)) (A (C)))  <==>  ((A) B) (A C)      PIVOT
```

Conventionally, cells with large delays can be buffered, replicated, bypassed or distributed. Each of these transformations is achieved by a particular Boundary Logic transformation already used within ILOC.

```
              ((A)) = A                  buffer     <==>  clarify

               A A  = A                  replicate  <==>  coalesce

            A {A B} = A {B}              bypass     <==>  extract

       ((A B)(A C)) = A ((B)(C))         distribute <==>  collect
```

Buffer insertion is the inverse of the basic Involution rule; replication, the inverse of the Coalesce rule; bypassing, the inverse of the Extract rule; and path shortening, the inverse of the Distribute rule.

Several commonly used techniques are simple combinations of the basic network transformations used in ILOC. A buffer tree is the repeated application of Replication and Involution.

```
            A ==> A A ==> ((A)) ((A))                  BUFFER TREE
```

The conventional generalized select algorithm is considered to be a complex, somewhat advanced technique. A slow internal signal is used as the selector to a MUX of the Boolean difference of paths for which that signal has been eliminated. This effectively advances the slow signal to the top of a path. In ILOC, generalized select is the Distribute rule followed by the Pervasion rule and the Distribution rule again in the other direction.

```
          (A  ) ((      B) (          C))
    ==>         (((A  ) B) (      (A) C))       distribute
    ==>         (((A B) B) ((A B) (A) C))       pervade (A B)
    ==>   (A B) ((      B) (      (A) C))       collect
```

Here, signal B begins nested two levels deep on the critical path, it ends up both one and two levels deep. In the case that B=1, a 0 result is returned immediately from the shallower B signal. When B=0, the result (A) is again returned immediately from the shallower instance of B. In all cases, the signal B immediately determines the path result, rather than having to propagate from the lower portions of the path.

Boundary Logic makes this complex transparent quite understandable, and quite accessible for implementation, by putting it into the theoretical structure of ILOC rules.

The generalized bypass algorithm is another advanced technique motivated by the bypass adder architecture. The Boolean difference formula suppresses a particular slow signal, while the difference itself is used to select between the suppressed signal or the critical path with the suppressed signal eliminated. This eliminates a logic chain by making it a false path, swapping the chain for two copies of that same chain with the slow input omitted. ILOC reduces this rather complex structure to the conjunction of the signal value and the network value when the particular signal is irrelevant.

$$((B)\ (F/B)) \;=\; ((b)\ (F/b{=}0\ \ F/b{=}1)\ ((F/b{=}0)(F/b{=}1)))$$

where F/b=0 is the network with signal b deleted (the wire cut) and F/b=1 is the network with the cell for which the signal b is an input deleted. That is,

F/b=0: cut the wire b
F/b=1: cut the output wire of any cell that wire b enters.

Rather than requiring an elaborate calculation of Boolean differences MUXed together, the ILOC model requires only an AND and two versions of the network with deleted signals.

These examples illustrate that Boundary Logic and the ILOC implementation place current delay reduction techniques into a broader framework of a small set of transformation rules that are easily implemented. The net result is that ILOC can achieve competitively superior delay reduction and area reduction at the same time.

ROUGH DRAFT OF REQUIRED RESOURCES

Estimates include personnel with expertise in building timing models, and in developing comprehensive synthesis testbeds. The deliverable is an alpha release software product for ASIC logic synthesis with highly competitive area and delay reduction capabilities.

Software Level ff Effort

TASKS		person-months senior	person-months junior	total person-months
system design	SUM	3	1	4
software		2		
integration		1	1	
software	SUM	14	20	34
parsers			4	
ILOC synthesis		6	6	
application interface		1	4	
user interface		7	6	
integration	SUM	5	3	8
configuration suite		2		
tool suite		3	3	
testing and documentation	SUM		11	11
design configuration test suite			1	
software			10	
TOTALS		22	35	57

Development Effort

Software development is projected for 11 months and will require six employees: 2 senior programmers, 3 junior programmers, and one quality control/documentation person.

For the two lead programmers and the four junior programmers, we expect an average productivity of fifty lines of functional code per day, from an average of 4 persons. The other 2 allocations consist of

1 full-time junior person for testing and documentation
1/2 full-time junior person for integration
1/2 full-time senior person for management duties.

The Chief Scientist and the lead programmers are expected to be capable of a sustained effort of about 100 lines of functional code per day. Each programmer should generate their daily code allowance for three days each week, the other time going to planning, meetings, etc. Thus

4 programmers at 50 lines/day at 3 days/week = ~600 lines/week

Total development effort is

13,000 lines core	= 23 weeks x 6 persons = 138 person/weeks
15,000 interface	= 24 weeks x 6 persons = 144 person/weeks
28,000 ILOC system	= 47 weeks x 6 persons = 282 person/weeks

Software   Development   Timelines

```
======================================================================
ACTIVITY            PERSON-MONTHS            TIME-LINE

SOFTWARE DEVELOPMENT     38
                              0  1  2  3  4  5  6  7  8  9 10 11 12
system design             4   x-----x
parsers                   4      x--------x
ILOC synthesis           11      x-----------------------x
application interface     3               x--------x
user interface           13               x--------------------x
design test suite         3         x--------x        x--------x

======================================================================
INTEGRATION/TESTING      19
                              0  1  2  3  4  5  6  7  8  9 10 11 12
integration               8                     x--------------x
design test suite         1      x--------x
testing                  10         x--------------------------x

======================================================================
OVERALL PROJECT          57
                              0  1  2  3  4  5  6  7  8  9 10 11 12
software                 38   x--------------------------------x
integration               8               x--------------------x
testing                  11         x--------------------------x

======================================================================
```

Projected Software Personnel

```
====================================================================
                              0  1  2  3  4  5  6  7  8  9 10 11 12
                      MONTHS

senior software I       11
      design                  O-----x
      synthesis                  x--------------------x
      interface                        x--------------------x
      integration                               x--------------x

senior software II      11
      design                  O-----x
      timing                     x--------------------x
      interface                                 x--------------x


junior software I       10
      parsers                    O--------x
      configuration                 x-----x        x-----x
      testing                       x--------------------------x
      integration                               x--------------x

junior software II      10
      synthesis                  O-----------------x
      interface                                 x--------------x
      application                                        x-----x

junior software III      8
      interface                     O--------------x
      synthesis                           x--------------x
      testing                                   x--------------x

testing/documentation    7
      synthesis                  O--------------x
      interface                              x--------------x
      testing                          x--------------------x
      manuals                                         x--------x

TOTAL                   57
                              0  1  2  3  4  5  6  7  8  9 10 11 12
====================================================================
```

Estimated level of effort for the junior software III and testing/documentation positions is less than full-time; however it is necessary for these people to be available over the duration of the project.

NOTE ON INTERNAL DEVELOPMENT ENVIROMENTS

The software level of effort, timeline and personnel attached above are predicated on some assumptions about development languages, environments, and personnel. In particular, they assume

1. A functional LISP code base with an object-oriented user interface

2. A UNIX/LINUX/MAC OSX development environment

3. Personnel who have worked with the Chief Scientist on prior projects. Further training is already included in the above levels of effort.

4. A dedicated systems administrator/web site coordinator not included in the project staffing

5. A version control and coordination system administered through an internal Company internet.

The above timeline and level of effort schedules are aggressive and allow little room for misjudgment, and no room for mid-course modification of the development environment.

It should be understood that the internal code development environment imposes no restrictions on the user base. The interface design incorporates the general principle that a user would perceive no difference at the interface between ILOC and conventional tools from Synopsis or Synplicity (except, naturally, better performance).

Internal development environments do have business implications, some of which include:

1. Prior to hiring, the internal tools must be decided, so that hired personnel are skilled in these tools. The risk associated with training software personnel in new tool sets, or in changing internal tools mid-way through a project, would be to approximately double development time.

2. Internal tools that are exotic (i.e., not Windows and C based) restrict the available hiring base.

3. Internal tools that are not object-oriented or functional (i.e. are C language based) incur the risk of obsolesce and of not being upgradable.

4. Object-oriented/functional environment images (i.e. the code infrastructure to be shipped) are approximately 20% the size of C/Windows code.

5. For a non-exotic C/Windows development environment, development time and personnel estimates should be doubled.

6. The risk of hiring lead programmers who have not previously worked with the Chief Scientist is to double development time.

7. The risk of adopting development environments not familiar to the Chief Scientist and to the Lead Programmer is complete failure.