STATUS OF THE ILOC IMPLEMENTATION

William Bricken
April 2004

CONTENTS

This memo includes a brief description of the ILOC logic synthesis system and its capabilities. Technical discussion is suppressed.

DESCRIPTION

The Iconic Logic Optimizing Compiler (ILOC) is an EDA logic synthesis software tool that provides an integrated and complete approach to network transformation, including area and delay optimization. ILOC takes a netlist circuit design as input and returns a functionally equivalent netlist of the same design with improved delay and area performance. ILOC 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.

ILOC synthesis improves upon the current state-of-the-art by incorporating innovative mathematical techniques and algorithms that are unlike any found in other commercial products. The Iconic Logic used within ILOC provides competitively more efficient network transformation algorithms within an integrated theory that organizes the currently haphazard and scattered approaches of competitive products.

ILOC is a system consisting of hundreds of Iconic Logic algorithms, and dozens of parameterizations. The netlist manipulation algorithms and all core Iconic Logic reduction algorithms are complete, validated, and extensively tested.

CAPABILITIES OF THE CURRENT ILOC PROTOTYPE

ILOC has been demonstrated to improve delay performance by approximately 10% for a wide variety of designs. When timing is not critical, ILOC can decrease silicon area by around 20%.

The ILOC software suite includes the following:

- independent reduction engines for logic, area, and delay reduction
- netlist parsers for HDL languages such as Verilog and VHDL
- 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 both reduces design iterations and provides competitive design performance with little effort.

The ILOC delay reduction capabilities are completed as alpha prototype code. They are not tightly integrated with the ILOC area reduction capabilities.

The comprehensive Iconic Logic network transformation model within ILOC confers a hidden value: although not fully implemented, several other design optimization capabilities are inherent within the ILOC architecture, for example hierarchical abstraction, design partitioning, and functional modularization.

CODE DESCRIPTION

ILOC uses a network transformation system based entirely on Iconic Logic. The ILOC software consists of five relatively independent engines and a global parameter-based control structure.

Parens Engine	2000 lines
Parens Stack Engine	2000 lines
Logic Reduction Engine	8000 lines
Delay Reduction Engine	8000 lines
Generator Engines	2000 lines

The Parens Engine and the Parens Stack Engine duplicate functionality.

An entire non-redundant core system consists of about 20000 lines of LISP code. The ratio of lines of C code to lines of LISP code is generally over 10 to 1. Thus were ILOC to be translated from the functional language model of LISP to a procedural model such as C, or to an object-oriented model such as JAVA, the core transformational algorithms, engines, and interface language parsers would consist of around 200,000 lines of code. Streamlined rewriting could reduce this to about 120,000 lines of code.

Current Implementation Status

- The core synthesis code is complete, alpha-prototype version
  - -- logic reduction engine
  - -- area reduction engine
  - -- delay reduction engine
  - -- EDA language interfaces
- The engines are not integrated
- There is no user interface; the programmer's interface is extensive.
- All code must be rewritten for application use

Current Implementation Strategy

The ILOC reduction strategy at this time is fairly rigid. About 30 local cell reduction transformations are organized hierarchically into five sequentially applied groups. Whenever a transformation group succeeds in making a reduction step, the reduction control algorithm begins again with the new design incorporating the small change. Some cell transformations are filtered to allow only local delay improvement, while others are applied at all times. None of these algorithms simultaneously considers both delay and area effects. The function of each transformation group follows:

Group One:	Greatly reduce the number of cell types.
Group Two:	Condense adjacent cells that always benefit.
Group Three:	Condense adjacent cells pending delay improvement.
Group Four:	Reduce fanout loads.
Group Five:	Terminal critical path changes that always benefit.

Future versions will optimize the selection of particular transformations without hierarchical organization, and will include top-level control of reduction rules.

SEQUENTIAL CIRCUITS AND REGISTER LOGIC

Designers need timing tools for assembling design components into a system. Currently ILOC handles register logic (mainly DFFs and a few other types used in FPGAs). This capability has been validated using approximately 80 sequential benchmarks, including finite state machines, multipliers, controllers, minmax, encryption, and bus interfaces.

Aspects of register logic not yet implemented in ILOC include:

-- register retiming, reconfiguration, and pipelining: in general, algorithms that move and place registers with a design.

-- a good sequential timing model, and separate analysis of set, reset, and enable logic.

-- tools for abstract optimization of sequential logic, such as state encoding, FSM decomposition, and register-based constraint modeling.

-- pragmatic design tools for Register Transfer Logic, such as timing specification for separate sequential blocks, waveform timing analysis, and timing-based layout.

ABSTRACTION TOOLS

ILOC includes designed but not yet implemented tools for component abstraction. Abstraction applies particularly to register management. These tools include:

-- identifying repetitive components (for library construction and timing partitioning)

-- identifying wide bitwidth signals (for bus, memory, and other routing and coordination)

-- function generators (for handling RTL specifications)

-- vector identification (for parallel processing design)

-- netlist partitioning (for register placement and signal encoding)

BETA RELEASE CODE REWRITE

The current ILOC version is an alpha prototype. ILOC development includes a beta-release rewrite of the logic and delay reduction tools, capable of being used by designers.

Several capabilities have not been fully implemented in the alpha version. Each will require deep integration with existing capabilities, especially within the transformation rule decision structures. ILOC is modular, consisting of about 40 transformation tools for each of area and delay reduction. In the alpha version, each set is independent. The decision rules for selecting these transformations to include measurement of the impact of the transformation on the entire (delay and area and power and ...)

current state need to be added. Adding register management also requires extension of the ILOC decision structures.

The existing ILOC architecture supports decision structure extension and integration. Design domain knowledge will be needed for the beta version, including:

-- expertise in knowing how the decision structure should change

-- an excellent behavior model

-- knowing what designers expect and want in configuration flexibility and freedom

-- knowing how important each transformation parameter is for the design process.

The current code is in LISP. There are several choices about implementation languages for the beta version. Some of the choices include:

-- re-implementation in C (an older industry standard)

-- re-implementation in an object-oriented language (C++ or Java)

-- automated parsing of LISP into C or Java

-- incorporation of LISP as an internal intermediate language (LISP is written in C, so it can viewed as "just some fancy C code that makes development much easier")

-- hooks through the API into C code of the host EDA system

-- assembly language for inner loops