ILOC LOGIC REDUCTION and COMESH LAYOUT for THE SP700 DESIGN

William Bricken
May 2003

The objective of this project is to evaluate the performance of the ILOC software and the CoMesh hardware architecture using a specific commercial design, SP700. ILOC is a proprietary circuit design analysis and optimization engine based on novel mathematical techniques. CoMesh is a reconfigurable silicon architecture custom designed for ILOC capabilities. Results are reported for software and for hardware separately. This technical report contains an Executive Summary, a Technical Discussion and an Appendix that includes detailed statistics.

EXECUTIVE SUMMARY

CONCLUSIONS

When measured by the number of standard two-input logic gates, the ILOC software reduces the structural VHDL netlist by 25%. The critical path length of 75% of all submodules decreased as a result of logic minimization.

When mapped onto a four-input LUT architecture, the ILOC reduction is highly sensitive to optimization goals. Current algorithms focus on routing rather than logic reduction. ILOC LUT mapping does not reduce the raw number of LUTs compared to the LUT map generated by the Synplicity software, the ILOC mapping does however reduce the routing required by the LUTs by over 35%. As well, the critical path length of 70% of all submodules decreased. The ILOC LUT mapping algorithms lack the customization specific to the Virtex architecture incorporated in the Synplicity software.

When mapped onto the CoMesh architecture, the entire design can be implemented in a logic fabric area of approximately 111 mm^2 (.13 process geometry), with an expected pipelined cycle time of 180 MHz.

The SP700 design analysis is proof in principle that ILOC can be applied to complex commercial designs. The analysis falls short, however, in providing clearly interpretable results upon which to base business decisions. A more rigorous design methodology and process is currently being developed.

SUMMARY OF RESULTS

Results are reported for two top-level SP700 modules (SAMMEL_SPS and ARITHM). The two modules represent over one-third of the total design; SAMMEL_SPS is a small module covering 2% of the design, while ARITHM is the largest module in the design, consisting of 36% of the entire design.

Software

The two modules represent approximately 70K two-input ASIC logic gates, and are reduced using ILOC algorithms to approximately 52K gates of the same type, *a logic reduction of 25%*. These modules incorporate approximately 2000 registers that are not included in this gate count. Path length reduction varied widely over submodules, from -20% to +24%. In cases of increased path length accompanied by logic reduction, a space/time trade-off choice is possible. Further, applying ILOC path reduction tools to the longest critical path can be applied to reduce the path length to a desired delay, with an accompanying increase in logic area.

To more accurately convey the performance of ILOC algorithms, ILOC settings and parameters were not customized for any particular submodules. In a real-world design environment, a designer may specify different parameters for different parts of the design, emphasizing reduction of path length, for example, along the critical path, while emphasizing area conservation along non-critical paths. By design iteration using different ILOC parameters, almost all particular results can be improved. The *overall average results* show substantial overall average logic reduction across the entire design. Specific results are in the Appendix.

The two analyzed modules require about 15K four-input look-up tables (4LUTs). When logic is mapped into 4LUTs without consideration of wiring requirements, ILOC increases the number of required 4LUTs by 2% compared to the Synplicity log. However, this increase in raw LUT count is accompanied by a conversion of many 4LUTs into 2LUTs and 3LUTs, thus reducing the wires required to route the design. When LUT mapping is weighted using the Synplicity gate counting metric, ILOC reduces the design complexity by 37%. Along with the logic reduction, the critical path length of 70% of the submodules decreased; the critical 4LUT path length reduction varied from -39% to +64%. Again, in cases of increased path length accompanied by logic reduction, a space/time trade-off choice is possible.

LUT mapping results are from simple LUT-mapping algorithm written especially for this report. LUT results do not reflect ILOC capabilities. For example, the LUT-mapper does not use the supporting logic found in Virtex chips, such as carry-chains. As well, ILOC currently separates reset and enable logic embedded in LUT registers. In comparing LUT metrics to the Synplicity log, ILOC results are adjusted for both register and non-LUT support logic. Adjustments are needed only for the LUT technology mapping comparisons. We are currently preparing a suggested methodology for making fair LUT-based comparisons of ILOC and Synplicity performance.

Silicon

A .13u CoMesh block uses .05 mm^2 of silicon for reconfigurable logic and supporting global routing. The two SP700 modules require 740 CoMesh blocks, a total silicon area for logic and routing of 37 mm^2. The overall chip logic density after placement and routing of the design modules is greater than 2500 gates per mm^2.

The CoMesh silicon architecture bears almost no resemblance to the LUT-based architecture of the original design. Each CoMesh block includes pipeline and storage registers as well as five levels of logic, while ILOC software completely retimes the design to fit the pipeline. The CoMesh design is intended to solve the performance problems of current LUT-based FPGAs.

TECHNICAL DISCUSSION

The Technical Discussion includes an introductory description of the SP700 design and the project activities. Comparison metrics are described, followed by the ILOC reduction results for these metrics, the LUT mapping results, and the CoMesh mapping and logic density results. Following the results are sections on timing comparisons, treatment of registers, hierarchical decomposition of the design, structure sharing, and validation. Details of the optimization process are described, as well as constraints and cautions on the results.

DESIGN DESCRIPTION

The sp700_src_lsp_prj folder was used as the source of all design specifications. This folder consists of 63 files, described in the Appendix. The VHDL code for the entire design occupies 14.4 Mb, of which 13.7 Mb (~95%) was parsed into the ILOC internal format. ILOC storage area for the design is 4 Mb, 30% of the VHDL storage area.

The ARITHM module is the largest component of the design. It is supported by various bussing modules, a PCI100 interface, a timing system, and i/o modules. The ARITHM module itself is dominated by a 32-bit multiplier (MULT32), which constitutes over 20% of the module, and 5% of the entire design. The cell library for the design consists of over 400 types, presumably constructed for mapping into Virtex CLBs. The ARITHM module uses about three-quarters of the cell library types.

The Synplicity log reports disconnected design components, the number and type of LUTs required by each design module when mapped to a Xilinx Virtex-II architecture, and a critical path static timing analysis for the LUT mapping.

ANALYZED CODE

Two modules from the SP700 design were analyzed: SAMMEL_SPS and ARITHM. These modules are defined in stand-alone files that contain the complete VHDL structural descriptions of the modules. These files are named

```
sp700-4 Folder/sp700_api__l1s0n0ft_src Folder/SAMMEL_SPS.vhd
sp700-4 Folder/sp700_api__l1s0n0ft_src Folder/ARITHM.vhd
```

The VDHL descriptions in these two files were supplemented by cell library definitions in the files:

`cb10vx_quick_1.vhd` through `cb10vx_quick_8.vhd`

also located in `sp700-4 Folder/sp700_api__l1s0n0ft_src Folder/`

For parsing the structural VHDL code in these files, the complete file was first converted into a text stream; then keywords and their accompanying information from VHDL were converted

into equivalent forms within the ILOC internal data format; and finally the labels identifying specific cell library elements were expanded into the ILOC forms for these functions.

PROJECT ACTIVITIES

1. Become familiar with the design components, cell libraries, and VHDL descriptions.
2. Write a parser to convert structural VHDL into the ILOC internal format.
3. Verify that the parsed ILOC design accurately represents the VHDL design.
4. Apply ILOC reduction algorithms to selected design modules.
5. Measure the reduction in design complexity achieved by ILOC processing.
6. Place and route the reduced design into the CoMesh architecture.
7. Measure the CoMesh performance for reduced design.
8. Write an N-LUT parser and EDIF-output parser for the reduced design.
9. Map the reduced design into a 4LUT architecture.
10. (ongoing) Design a fair and informative methodology for ILOC evaluation.

COMPARISON METRICS

The evaluation strategy for this study was to compare design metrics for the VHDL structural specification from within the ILOC internal format, both prior to any modification or reduction, and after ILOC logic reduction. The reduced design was then technology mapped to LUTs and to the CoMesh architecture for silicon area performance comparisons. Since FPGA layout details are not known, and since the CoMesh global architecture is not yet modeled, only critical path lengths are reported and no timing analysis is included.

ILOC

Five metrics were selected for comparison of design entities prior to and after ILOC reduction:

1. Number of two-input NAND gates: The cell library of the original design was converted into two-input NAND gates, as was the output of the ILOC reduction. FFs are counted separately and not optimized. Inverters are not counted. This metric is a simple way to measure before and after logic reduction.

2. Number of literals: Independent of cell type, the number of input pins to each internal cell is counted. Inverters are not counted. This metric has been shown to correlate highly with overall logic and wiring requirements for ASIC designs.

3. Number of nets: Independent of cell type, the number of separate internal nets connecting cells was counted. Each primary input and output contributes one net. This metric indicates reduction of routing requirements for a design.

4. Raw number of 4LUTs: The Synplicity log provides a count of LUTs generated in mapping the design to LUTs with four or fewer inputs. The reduced design was mapped to 4LUTs (including 3LUTs and 2LUTs) and the total number of LUTs was counted. This metric indicates the logic resource requirements for placing the design into a 4LUT silicon architecture.

5. Weighted number of LUTs: The Synplicity log also provides a weighted average over LUT mappings. Each 4LUT is counted as one unit, with 3LUTs counting as 1/2 and 2LUTs counting as 1/4 units. This metric recognizes that LUTs with fewer inputs (i.e. 2LUTs and 3LUTs) require less routing and power resources.

Critical Path Lengths

Maximum path lengths are reported for all hierarchical submodules of the ARITHM module, in two different forms:

1) Critical path length measured by the number of two-input NAND gates, prior to ILOC reduction and after ILOC reduction

2) Critical path length measured by the number of 4LUTs, prior to ILOC reduction and after ILOC reduction.

ILOC includes unique and powerful tools for reducing critical path length. These tools identify locations along the critical path that achieve the best timing and logic area trade-off. No path reduction is included in this report, since exact critical paths and timing points are not known.

CoMesh

The design modules were also mapped into the CoMesh silicon architecture in order to estimate performance expectations for the CoMesh approach to reconfigurable computing.

1. The number of CoMesh blocks required for each design module was counted. The silicon area of each block is known, providing a measure of the silicon area required to implement the design using the CoMesh architecture.

2. The number of CoMesh block cycles was counted, providing a cycle-time for each design component for the pipelined CoMesh block architecture. Since CoMesh is fully pipelined, the clocked cycle-time of 360 MHz is achieved for most designs. For the SP700 design, the

block communication clock cycle-time was halved, to 180 MHz, effectively allowing two sequential blocks to complete logic processing each cycle.

ILOC SOFTWARE LOGIC REDUCTION

The SP700 design is specified in a structural VDHL netlist composed of Xilinx cell library elements. Approximately 95% of the design was directly parsed into ILOC format, without any modification of the library cells or of the design modules. Design entities that included block memory, and those that incorporated library cells customized to the Virtex architecture were not analyzed. As well, a few modules that were specified in behavioral rather than structural VHDL were not analyzed.

Most of the Xilinx cell library elements, including FFs, are complex, each library element represents an average of three simple two-input logic gates. To provide a before and after software logic reduction comparison, the logic complexity measures taken prior to reduction were again applied after ILOC reduction.

As a general strategy, ILOC reduction algorithms were applied uniformly and automatically to all design components. These algorithms were tuned prior to reduction, to conform algorithm parameters to the general types of logic structures in the design. However, no substantive effort was made to locate parameter settings that would provide the best reduction performance, either for individual entities or for the design as a whole. Thus the reduction results are indicative of what a typical user would get using the tool without any parametric customization and without applying specialized algorithms to submodules in the design.

Summary results for the two analysis modules follow. The table shows absolute counts and percentage reduction for the five metrics. "Before" data is from the literal transcription of the VHDL netlist; "after" data is after the ILOC reduction of the netlist. Details for the component entities of these modules are in the Appendix.

ILOC Reduction of Design Components

MODULE	SAMMEL_SPS			ARITHM		
METRIC	before	after	%	before	after	%
Two-Input NAND Gates	2592	2434	6%	69152	52056	25%
Internal Input Pins	4778	4306	10%	94877	63960	33%
Internal Wire Nets	3944	2579	35%	102476	67674	34%
Raw Number of LUTs	669	616	8%	14544	14803	-2%
Weighted LUTs	449	413	8%	12578	7839	37%

For the SAMMEL-SPS module in its entirety, the logic reduction was 6% expressed in NAND gates, 10% expressed in internal pins, and 35% expressed in wire nets. For the ARITHM module, the logic reduction was 25% expressed in NAND gates, 33% expressed in internal pins, and 34% expressed in wire nets. LUT results are discussed in the next section.

The differences in logic reduction between the two modules can be understood in terms of available logic complexity. SAMMEL_SPS is a small module consisting mostly of small functions with little reduction and little interaction with other components. In contrast, ARITHM is relatively large, providing substantial opportunities for identification and reduction of complex and redundant logic across module boundaries.

Results for the three structural metrics are highly correlated, although measured results varied widely across design entity structures. In general, ILOC reduction always improves an entity by reducing its complexity. This is because ILOC is deletion based, so that all changes are simply removal of redundant structure. Reduction gains range generally from a few percent to around 30% for larger complex entities without highly organized internal structure. For example the submodule DIV reduced by 22%, primarily due to a large unoptimized table entity. The barrel shift and mask submodules reduced only by 10%, because of their regular internal structure. Register banks showed little reduction (2% for ACCU_REGS, for example). The 32-bit multiplier showed significant reduction (32%), however this reduction was accompanied by an increase in critical path length of 20%. This multiplier is entirely combinational, not taking advantage of available pipelining.

Substantive ILOC gains in logic reduction occur primarily in:

1. Library cells with redundant logic that may have been included for error checking. These redundant checks are often associated with a specific technology mapping, and are not necessary for the CoMesh architecture.

2. Functions for which ILOC transformations are particularly powerful in exposing redundant and inefficiently designed logic structure. The relative weakness of current optimization algorithms obscures available optimization transformations that ILOC makes transparent in several classes of functional structure (such as deep nesting, distribution, structure sharing, and branching abstraction).

3. Large logic functions for which current software algorithms are simply too slow or inefficient to identify redundancies.

4. Hierarchical module boundaries. Redundant functionality can exist in separate design modules that must be flattened in order to optimize. The multilevel logic of CoMesh permits hierarchical flattening while maintaining deeply nested (and thus highly efficient) logic structures.

5. Flattened functions for which retiming has not been applied. ILOC register retiming and optimization are fully integrated into ILOC logic optimization.

The obtained logic reduction results for SP700 are diluted by three factors:

1. Complex register structures in the Xilinx library were decomposed into simple CoMesh FFs and supporting logic. The supporting logic (such as resets, registered adders, and multiple outputs) was added to the generic module logic, thus increasing the peripheral logic metrics.

2. Technology mapping to NAND gates, LUTs, and other logic structures not native to the CoMesh architecture reverse some optimization gains made by ILOC algorithms intended specifically for the CoMesh architecture. Optimization to CoMesh is reported in a following section.

3. Peripheral logic in the Virtex CLB, such as carry chains and MUXes, is counted separately in the Synplicity log, but not separated in the ILOC reduction analysis.

LOOK-UP TABLE (LUT) MAPPED LOGIC REDUCTION

It is well known that different silicon architectures require different optimization structures and strategies. To output a LUT-based decomposition of the ILOC data structures, it is not sufficient to simply partition optimized results into LUTs, since LUT-based architectures do not resemble the CoMesh architecture. To achieve a competitive LUT partitioning, the optimization settings and computational modules of ILOC would require configuration specifically for LUT mapping. The current Bricken LUT mapping algorithms are not optimized for LUT-based architectures, negating much of the CoMesh specific optimization. These Bricken algorithms are a first-pass and non-competitive version that have many opportunities for improvement. In particular, they would need to be extended to take advantage of the specialized logic within Virtex Computational Logic Blocks (CLBs) that supports the 4LUTs within a CLB. Therefore, the results reported herein for LUTs do not accurately reflect the potential of ILOC capabilities for LUT-mapping.

LUT Layout Area Comparison

The Synplicity log provides a LUT usage figure after layout using two different approaches. In one reporting, LUTs are counted in a weighted metric: 2LUTs are half as expensive as 3LUTs, which are again half as expensive as 4LUTs. Each FF and each 4LUT counts as one "gate". In a different Synplicity log, the raw number of required LUTs (Function Generators) is counted, in recognition that a Virtex-based 2LUT still requires a 4LUT to implement. However, LUTs with fewer inputs operate faster, and do not put as much of a burden on routing. Since there is usually excess logic but restricted routing resources in an FPGA, the weighted average is probably a better predictor of FPGA resource shortages. Both metrics are reported below.

No attempt has been made to optimize ILOC output specifically for 4LUT mapping. As well, various special performance enhancing features of a Virtex CLB extend the simple LUT functionality, making the rationale for mapping solely to 4LUTs less robust.

The table below presents the count of LUT types for each analysis module, both for the Synplicity mapping and for the ILOC mapping. The log results below were taken directly from the Synplicity log included in the design files.

ILOC Reduction of Design Components Expressed in LUTs

MODULE	SAMMEL_SPS		ARITHM	
LUT COUNTS	SYNPLICITY	ILOC	SYNPLICITY	ILOC
Two Input	114	240	1341	2301
Three Input	180	277	1941	4376
Four Input	375	380	11262	11534
All LUTS	669	897	14544	18211
adjusted ILOC		616		14803
% reduction	8%		-2%	
Weighted LUTS	449	503	12578	10416
adjusted ILOC		413		7839
% reduction	8%		37%	

In the above tabulation, ILOC LUT figures have been adjusted in two ways for a more balanced comparison. The Virtex mapping includes usage statistics for various specialized logic within the Virtex CLB. It is assumed that each specialized logic component is on average equivalent to one LUT. The total LUT count for ILOC is reduced by one LUT whenever a specialized logic feature is explicitly used in a CLB. In contrast, the weighted LUT count is reduced by the relative complexity of the specialized logic. For example, a four-input MUX is considered to be equivalent to one 4LUT, while a two input MUX is considered to be equivalent to 1/4 of a 4LUT. This is the same weighting used by Synplicity in the design log.

A second adjustment was made to the ILOC data to compensate for additional logic inside CLB registers that is not counted in the Synplicity log data, but is counted in the ILOC data. No adjustment was made for simple DFF registers or latches. However, when register logic included AND, MUX, or other logic functions, these functions occurred explicitly as separate logic functions in the ILOC model. To bring the two models into alignment, this specific register logic is not included in the ILOC LUT totals. As with specialized CLB logic, total LUT counts are reduced by LUTs expressing the explicit register logic, while weighted LUT counts are reduced proportionally to the amount of explicit register logic.

In the above table, the configuration of LUT types for reporting raw LUT count represents the best results obtained by the ILOC LUT mapper using the objective of minimizing the total number of LUTS, regardless of the type of LUT. The row labeled Weighted LUTs presents the best results using the criteria of minimizing the total number of input pins required by the collection of LUTs. This objective recognizes that routing is far more of a problem than logic density in current FPGAs. The distribution of LUT types for the unweighted results above is different than the distribution of LUT types for the weighted results.

Three entities in ARITHM contained specialized Virtex cell library elements. These elements were not parsed into the ILOC data structure. In counting the LUTs for these entities, the number reported in the Synplicity log was simply used for both the log tabulation and the ILOC tabulation.

The Synplicity log selectively reports the LUT mapping for some lower level design modules, but not for others. The exact counting criteria used by Synplicity, and the specific types and levels of optimization effort are not known. In some cases, registers specifically within an entity were not reported by the Synplicity tabulation and, in some cases, entire entities were not reported. In such cases, the logic actually parsed into ILOC was used as a comparison.

COMPARISON OF PATH LENGTHS

Modification of critical path length is reported for the ARITHM module. The critical paths are measured using a unit delay model, essentially counting the maximum number of logic gates between registers for each submodule in the case of sequential entities and between input and output in the case of combinational entities. Most entities contained multiple outputs; the critical path for an entity is the longest path for any output of that entity. Two technology maps are reported: two-input NAND gates, and 4LUTs. The ILOC tools include path minimization algorithms for the NAND gate format, but not for the LUT format. No attempt was made in any circumstance to minimize the critical path lengths, thus critical path lengths should not serve as a basis for comparative evaluation. The Appendix contains complete data for the entire ARITHM module.

The evaluation strategy was to compare path length metrics for the VHDL structural specification from within the ILOC internal format, both prior to any modification or reduction, and after ILOC logic reduction. As a general strategy, ILOC reduction algorithms were applied uniformly and automatically to all design components.

It should be noted that ILOC reduction incorporates desirable path design features that are not measured solely by critical path length. These include:

1) removal of reconvergent paths,
2) elimination of false paths,
3) logic reduction over entire paths independent of the types of gates on a path, and
4) localized design flexibility to change path lengths with ILOC providing the exact space/time trade-off for any localized modification.

For both formats, ILOC reduction resulted in incidental shortening of critical paths, as would be expected to accompany reduced circuit structure.

Critical Paths for Two-Input NAND Mapping

The cell library of the original design was converted into two-input NAND gates, as was the output of ILOC reduction. The longest path through the NAND gate networks was then measured for both the pre-reduction and post-reduction circuits. FFs served as path terminals for sequential circuits. Inverters are not counted in the path length. This metric is a simple way to measure before and after path length reduction due solely to ILOC logic reduction.

It is possible to decrease path length by increasing gate count using a distributive process. ILOC includes powerful and unique tools to decrease the gate count of a path at specific

locations. These tools identify path locations for which the most efficient trade-off between reduced path length and increased gate count can be achieved. As well, all identified path transformation locations include exact metric information about the trade-off, specifying the number of unit delays removed from the path, and the number of gates added to the circuit to achieve this path reduction. *No path reduction transformations were applied to the reported critical paths.*

Only fully flattened submodules from the ARITHM module are included in the reported data; these cover the entire module. Submodules are separated by type, either combinational or sequential, and classified by whether logic reduction resulted in shorter, longer, or the same length critical paths. When submodules are combined into an entire circuit, only the longest critical path in the circuit is of relevance, since this path establishes the greatest propagation delay for signals through the circuit. All other paths are non-critical. Detailed data in the Supplement Appendix shows exact path length changes for all submodules. Reduction of paths over many specific submodules can be viewed as an indicator of the consistency and power of path reduction due to ILOC logic optimization for a diversity of function types.

Critical Path Reduction Due to Logic Reduction, NAND Gate Technology Map

Entity Type	Number of Cases	% of Entities with Paths		
		Shorter	Same	Longer
Combinational	39	29	7	3
Sequential	17	10	5	2
Total	56	39	12	5

In general, ILOC logic reduction decreases or does not change critical path length for about 90% of all entities. For those entities for which the critical path is longer, it is of course possible to revert to the non-reduced form, making an explicit and known trade of more gates for a shorter path. For example, the MULT submodule increased in path length from 101 unit delays to 121 unit delays, in exchange for a decrease in gate count of 32%, which is over 5000 gates. In such cases, ILOC provides a designer with a powerful set of design option choices. On the other extreme, ILOC reduced the path delay of three large submodules (CONV, CORD, and EXLN) by 25%, resulting in a decrease in over 60 unit delays for each submodule, while at the same time reducing the gate count of the three by an average of over 2000 gates (ranging from 6 to 19%)

Critical Paths for 4LUT Mapping

Comparative critical path length results are reported for 4LUT technology maps applied both before and after optimization, using a unit delay model for which each LUT contributed one unit delay. Critical paths for combinational entities are the longest series of LUT elements from input to output. Critical paths for sequential entities are the longest series of LUT elements between any two registers.

To determine the pre-reduction LUT path length, the VHDL specifications were parsed directly into ILOC format, and then immediately converted to LUT format by the ILOC LUT mapper. To determine the post-reduction LUT path length, the VHDL specifications were first parsed into ILOC format, then ILOC reductions were applied, and finally the resulting circuits were mapped into LUT format.

Critical Path Reduction Due to Logic Reduction, 4LUT Technology Map

Entity Type	Number of Cases	% of Entities with Paths		
		Shorter	Same	Longer
Combinational	39	29	5	5
Sequential	17	11	1	5
Total	56	40	6	10

Approximately 70% of the submodules decreased in LUT path length as a result of logic reduction. Reduction varies from -39% to +64%.

In general, the LUT path length reduction results mirror the NAND gate path length reduction results. The higher variance in LUT results is probably due to the ILOC LUT mapper. Since both before and after measurements of LUT results are based on LUT networks generated by the same LUT mapper, the path reductions are in general not artifacts of the LUT mapper. This is supported by similar reductions in NAND-based path lengths. However, no path length results from the ILOC LUT mapper are comparable to path lengths generated by a commercial mapper, such as Synplicity, since the mapping quality of the LUT mappers is not comparable.

There is one clearly identifiable case for which before and after LUT path lengths are not comparable within the ILOC LUT mapper, this is when the after LUT map shows a large increase in path length. For example, the SQRT entity has a pre-reduction path length of 67 unit delays and a post-reduction path length of 93 unit delays, even though the number of gates in the entity is reduced by more than 25%. This result is even more anomalous when it is observed that the NAND gate path length is only 79 unit delays. Closer inspection of this and two other similar cases (FLAG_BASIC and P_WERKE_REG) shows that the relative increase in LUT path length is due to a preponderance of wide functions.

In conventional FPGA architectures, wide functions are accommodated by special feature MUXes that permit two or more LUTs to be combined into a single functionally limited wide-input LUT. In the Virtex architecture, for example, two 4LUTs can be converted into one 8LUT using special MUX circuitry within the Virtex CLB. CoMesh cells provide a similar option. The present ILOC LUT mapper does not include the wide-input MUX feature. Instead, wide logic functions are converted into narrower chains of LUTs. This conversion significantly increases the critical path length. Pre-reduction LUT paths are shorter because the reduction process creates wide-input functions during reduction. NAND-gate paths are shorter because wide-input NAND gates are converted into a binary tree (rather than a LUT chain) that conserves path length in exchange for more gates. Further refinement of the LUT mapper can remove this problem.

SILICON AREA AND COMESH LOGIC DENSITY

The target process geometries for CoMesh are .13 microns and smaller. Data is reported here for both .13u and .18u process geometries since the CoMesh SPICE simulation is for .18u. Reported CoMesh silicon areas are for logic and local routing only, as if the CoMesh reconfigurable block were an embedded component of a larger chip. The design of CoMesh global routing is not yet fully simulated, nor are the i/o pads, clocks and other peripherals for a complete CoMesh chip. It is believed that the area for global routing and i/o will be less than two times the area of the logic blocks. Block interconnect, when placed on a separate metal fabrication layer, may not add area to the logic footprint. A conservative estimate of the area requirements for block neighborhood and global interconnect is two times the logic area.

CoMesh Logic Block Area

Geometry	Block Dimensions	Block Area	Area Including Routing
.18u:	260u x 118u	30680 u2 (~.03 mm^2)	~.09 mm^2
.13u:	188u x 85u	16000 u2 (~.016 mm^2)	.05 mm^2

CoMesh blocks are designed to have 32 registers terminating the five levels of asynchronous logic. Combined with the five levels of logic and the ILOC restructuring algorithms, the number of registers per block is sufficient to implement a total design pipeline, and to accommodate register intensive design elements such as FIFOs, counters and shift registers. Each CoMesh block accommodates approximately 100 two-input ASIC gates, after placement and routing. Conventionally, each register is counted as six logic gates, thus a CoMesh block is equivalent to up to 300 conventional logic gates.

Silicon area results for the design modules are reported below in mm^2 for both .18u and .13u geometries. Silicon areas are computed by multiplying the number of blocks required to implement the module by the area of each block with routing area included. Design component details are in the Appendix.

CoMesh Silicon Area

	SAMMEL_SPS	ARITHM	BOTH	
Blocks Needed	27	713	740	
Silicon Area .18u:	2.43	64.2	66.6	mm^2
Silicon Area .13u:	1.35	35.7	37.0	mm^2

Based on these results, the extrapolated area required for the entire design is approximately three times the area of the ARITHM module, which would be 111 mm^2.

CoMesh Logic Density in Gates/mm^2

	SAMMEL_SPS	ARITHM	BOTH	
Logic Gates	5792	87622	93414	
Logic Density .18u:	2380	1370	1400	gates/mm^2
Logic Density .13u:	4290	2450	2520	gates/mm^2

Logic density for the SAMMEL_SPS module is based on 2592 two-input ASIC gates in the design, plus 10 gates for each of the 320 registers in the module. The ARITHM module density is based on 69152 gates and 1847 registers.

These densities are for a stand-alone CoMesh chip application. For an embedded CoMesh application for which peripheral support, i/o, and global routing are not required, the expected logic density could approximately double the figures reported above.

The CoMesh silicon/software codesign is highly robust to logic changes and additions, since timing across blocks and pin-out locations that can change during engineering redesign do not change the CoMesh timing or logic placement. Such changes can be accommodated using ILOC place and route software. This architectural model is substantively different than conventional reconfigurable silicon, and has been achieved through intimate coupling of software and silicon capabilities. Unlike LUTs, CoMesh blocks can share unrelated logic, allowing CoMesh block resource utilization to approach 90%.

TIMING COMPARISONS

Separate entities may not be on the same critical path as others, so that delay times do not add. The SAMMEL_SPS module is not listed as being on the critical path for the entire design, while the ARITHM module is implicated in all critical paths listed by the Synplicity log for the SP700 design. The Synplicity Clock Frequency Report indicates an achievable frequency of 2.3 MHz for the ARITHM module, with many critical paths incurring a delay of 333-336 us.

The CoMesh architecture takes a significantly different approach to timing than does the Virtex architecture modeled in Synplicity. In CoMesh, the entire design is pipelined, using the delay through a single block as the significant path delay. The number of blocks in the critical path thus represents the pipeline set-up time, not the path delay. Similar to the non-additivity of delays in an FPGA architecture, the CoMesh block delays do not add. CoMesh timing is associated only with the pipelined layout structure of the entire flattened design.

At a .18u process geometry, the CoMesh pipeline SPICE model operates at 300 MHz (3.3ns cycle delay). This delay includes the block delay of 2.2ns and the global long-line delay of 1.1ns. Thus the register bank that terminates each logic processing block is clocked across the CoMesh chip at 300 MHz. For a .13 process, the chip pipeline is expected to operate at 360 MHz or more.

The flattened ARITHM module spreads across 713 CoMesh blocks, and is expected to incur no global communication delays. Thus the achievable cycle time for this module is the local inter-block-neighborhood pipeline delay. At .18u, this delay is 2.2ns for the block delay and 1.1ns for the inter-neighborhood routing delay synchronized with global routing delays, resulting in a cycle time of 300 MHz. At .13u, the total delay for a single cycle would be in the range of 2.8ns, or 360 MHz.

The significant cases when the CoMesh baseline pipeline processing rate is changed are when designs have inner loops and outer loops, such that the outer loops must wait for more than one processing cycle of the inner loops prior to passing processed data to the next block. In the ARITHM module, a cursory analysis indicates no such loops. Specific techniques for loop-unrolling can be applied when necessary, swapping area for time.

Allowing for two blocks to complete processing prior to clocking would permit an overall clock frequency of 180 MHz at .13u. This is the frequency used for the current analysis.

REGISTERS

Modification of the ILOC reduction algorithms to accommodate the dozens of different register types in SP700 was not attempted. Instead, all SP700 registers were converted into one simple standardized format compatible with the CoMesh architecture. In order to achieve comparability to FPGAs, it is assumed that any LUT has one register that can be of any register type. Thus for the Virtex model the FF storage function, and the logic associated with resets, enables, and the like, are counted as part of the mapped LUT without incurring additional logic overhead. In contrast, the initial ILOC register module counts all peripheral register logic as part of the standard logic routed through CoMesh blocks.

Statistics and metrics are adjusted for the different register models. In particular, for the ILOC optimization data, measurements include the external register control logic. Thus, two-input ASIC gate counts, etc. include separately counted gates for register logic. Thus the CoMesh cell area usage is overestimated to the extent that CoMesh register control logic is separate from cell logic. For LUT mapping comparison, register control logic is explicitly subtracted for LUT counts, through a technique of assigning separate LUTs to control logic, and then reducing the total LUT count by the number of LUTs used solely for register control. See the details of LUT mapping comparisons in the Appendix.

HIERARCHICAL EXPANSION

The ARITHM module contains five levels of design hierarchy, with the higher levels containing very little logic, and serving solely to connect lower level components. This programming style incurs huge port and pointer overheads. It is not known if this overhead is a cause of routing problems for the design, however, the ILOC system eliminates nearly all of the port identifiers while still maintaining modularity. In addition, ILOC has the capability to abstract repetitive patterns in a design; abstraction can occur across logic patterns and functions, or across wire patterns and groupings (busses, for example).

One significant difference between ILOC and VHDL coding styles (and supposedly FPGA mapping tools) is the approach to representation. ILOC is primarily functional rather than either procedural or behavioral. A functional style achieves modularity by "in-line substitution" as opposed to ports, connectors, and wires. Thus to connect two components, the structure of one component is substituted for the name of that component in the data structure of the other component. This approach is a consequence of the Boundary Logic model that combines logic functionality and logic connectivity into a single concept. (This concept is the "boundary", for which crossing over a boundary represents wire connectivity, and encountering a boundary represents logic functionality.)

In ILOC, any boundary is a potential hierarchical subdivision. Structures with the same boundary configuration, but with different collections of names are replicated modules. These replicated functionalities are identified dynamically during ILOC optimization, without reference to the intent of the designer or the modular expression of the design within VHDL. The general approach is that once the design is functionally valid, how it looks is solely an issue of technology mapping. In conventional FPGA design, "conceptual" modules are placed and then routed. In ILOC technology mapping, the conceptual structure that dictates the design decomposition is the CoMesh block rather than the design components conceptualized by the designer.

The Appendix shows the gains due to hierarchical flattening of the ARITHM module. In general, such gains are minor, 1-2%. And flattening gains are not cumulative, so that submodules flattened at a lower level of the design hierarchy do not continue to show gain at higher levels of the hierarchy. Flattening significantly increases logic reduction processing time, since all algorithms must address larger data structures. Thus, flattening is not a particularly good use of processing time for this design.

STRUCTURE SHARING

The ILOC structure-sharing algorithm first reduces a module to simple elements, and then iteratively reconstructs the structure of the module from these elements. A physical analogy of this process would be decomposing an object into the atoms composing it, and then optimally reassembling the atoms into molecules.

This algorithm is relatively slow, but it is not intractable for large circuits. More iterations successively reduce a circuit to a minimal form. Thus, a particular circuit, given sufficient processing time, can often be reduced to fit into limited resources.

As an example, the following chart maps processing time against amount of reduction for the ARITH component of the ARITHM module. To begin, the module is first reduced by all efficient (N log N) ILOC algorithms. Then the module is successively decomposed and restructured by the structure-sharing tools. Each cycle below represents about 10 CPU minutes for this module (on a relatively slow machine). The algorithm currently has no speed-up optimizations, and is expected to increase in speed by ten to fifty times when optimized.

ILOC structure sharing is not a random search similar to techniques such as simulated annealing. Rather, structure sharing is goal directed, stopping when reduction is not achieved within a

single cycle. The irregular reduction behavior of the algorithm, as illustrated below by reduction gains per cycle, is a result of successive structural organization that reaches critical stages of organization. The organizational structure then cascades throughout the remaining circuit structure, resulting in a global reduction.

Structure-Sharing Reductions

CYCLE	ASIC GATE COUNT	CHANGE/CYCLE	
0	39281		
1	37441	1840	
2	37305	136	
4	37290	8	
6	37284	3	
9	37251	11	
10	36752	499	
12	36720	16	
14	36402	283	
16	36284	59	
20	36237	12	
24	36233	1	
26	36166	34	
28	36131	17	
30	36117	7	
32	36113	2	
33	36112	1	process completion

Total reduction: 3169 gates (8%)

VALIDATION

Lacking test-vectors, the parsing from VHDL to ILOC and from ILOC to LUT-based EDIF could not be rigorously tested. However, since ILOC is a formal theorem prover, several internal sanity checks are available to provide some reasonable confidence that all large and significant errors have been avoided.

1. All algorithms and transformation tools internal to ILOC are validated against a test file of over 200 circuits with extensive test vectors.

2. ILOC includes many traps and filters to catch errors. These include checking each circuit for disconnected, isolated, and unused wires and logic, verifying that no logical or structural contradictions are shorting or degenerating the circuit structure, and functional equivalence checking for all internal transformations.

3. Each design entity was checked by hand for correctness of port connections and logic flow. Unusual circumstances such as large changes in critical path length or number of logic elements were filtered and warnings posted.

4. Periodic printouts of intermediate reduction steps, frequency of calls to algorithms, and processing times provided grounding for internal process verification.

5. Before and after entity structures were verified for functional equivalence. This type of checking was not possible across hierarchical design modules.

6. Hierarchical and flattened versions were cross-checked for composition and consistency.

Several validation cautions are in order:

1. Since many processes, both computational and human, were applied to over 130 separate entities, it is possible for some errors to have occurred. These errors would probably effect the functional correctness of the design, but would be very unlikely to effect the general accuracy of the logic density and timing results.

2. A few library components were difficult to model. These included some registers and some Virtex components. The sample ARITHM module contains a minimum number of these difficult elements, and when their modeling was at all dubious, they were isolated from the reduction processes. Three entities in ARITHM had cells that were excluded from analysis since their behavioral specification was difficult to interpret.

3. The most difficult aspect of transforming the design was assuring that the connectivity and wiring model from VHDL was aligned with the connectivity model within ILOC. The two languages take substantively different approaches to the idea of connectivity, however these differences were largely accommodated. The greatest source of risk of error was in modeling the very minor details of how signals were passed between modules, particularly between modules that were not directly connected. Three examples:

3A) Signals in VHDL could be passed from input to output without traversing (symbolically) the interior of an entity. Thus formal i/o parameter names could be subverted.

3B) Names were not unique across VHDL entities, yet these names were often used by the designer to identify the same wire in different contexts. This provided the opportunity for syntactic parsers to become confused.

3C) Naming conventions across the two systems were significantly different, however in two separate cases a naming conflict occurred, for which an internally generated ILOC name conflicted with an explicit VHDL name.

4. Another source of compatibility difficulty between VHDL and ILOC was that VHDL did not support a strong or dynamic typing system, whereas ILOC is strongly and redundantly typed to protect against transformational errors. Thus, VHDL can pass a signal coming from a register into another module without maintaining the information that the signal is in fact generated as a register output.

LUT MAPPING and the VALUE PROPOSITION

The ILOC algorithms do not yet include sophisticated engines for LUT technology mapping or for LUT path reduction. All LUT results are from a naive 4LUT mapper written specially for this project and in no way customized for competitive performance. In particular,

1) The LUT mapper translates ILOC wiring reduction into LUT input reductions, and not into a reduced number of LUTs.

2) The LUT mapper is not a LUT optimizer. It does not reduce the raw number of LUTs in an entity, nor does it reduce the critical paths of a LUT-based design.

3) The LUT mapper does not include any of the special features found in Xilinx CLBs and used by Synplicity LUT mapping, including carry-chains, wide-input MUXes, and specialized XOR functions. As a consequence, Synplicity LUT mapping results should not be considered to be comparable to ILOC LUT mapping results.

The ILOC reduction engine has been designed to minimize ASIC circuits in general and to generate optimized configuration files for the CoMesh silicon architecture in particular. It should be recognized that due to the interaction of synthesis optimization and technology mapping, gains in optimization targeted for one technology (such as ASIC design) do not transfer into gains in optimization for a non-targeted technology (here, for example, LUT mapping).

The ILOC value proposition is an integrated silicon/software suite (CoMesh/ILOC) that provides superior competitive performance for reconfigurable logic, as is documented in the main report. The ILOC software shows exceptional potential when added to the standard synthesis tool-chain for ASICs. We believe that the ILOC software also has undeveloped potential when applied to the standard tool-chain for FPGAs. The main report and this supplement are intended to provide preliminary evaluative information to support our value proposition, using the SP700.

REDUCTION DETAILS

1. The parsing from structural VHDL to ILOC was structurally and functionally exact, no reductions were applied. The Synplicity log includes flags for unused design components, as did the ILOC code. These were verified to match by hand. As well, the ILOC code is rigorously sensitive to design flaws such as missing connections, inappropriate gates, and logical contradictions. The final parse triggered no ILOC error filters.

2. Two different generic FF models were used.

2A) The FF library provided within the design was used without modification or optimization.

2B) All FFs were converted to a simple homogeneous model used within CoMesh. Resets, enables, and other control signals were separated from the basic FF and combined with the other logic of the circuit directly.

3. Modeling the Sp700 design FF library was difficult, since ILOC has not been customized for a variety of FF models. Since the ILOC code required significant modification for generic simulation of the expanded FF library, and since the optimization routines do not

deconstruct the library FFs, results are reported only for the simple CoMesh FF model. Expressing the internal complexity of the FFs as part of the design logic increases the area of the ILOC design relative to the VHDL model, while using simple FFs permits automated retiming, pipelining and other CoMesh related optimizations not directly available in VHDL or Synplicity.

4. After direct parsing into ILOC, four simple reductions were applied:

4A) Ground values were removed and propagated.
4B) Redundant and unused elements were removed.
4C) Inverters were removed by combining them into ILOC gates using a technique analogous to conventional bubble-logic.
4D) All cells with no fanout were conglomerated, so that all nodes in the logic network had fanout. These conglomerate nodes were reduced using iconic logic.

5. Two reductions were applied and selected based on positive results:

5A) Distribution of logic elements.
5B) Local expansion of logic elements.

These techniques are unique to ILOC. Although they are known transformation rules for logical analysis, these transformational rules are generally not applied in conventional systems due to their complexity within those systems.

6. All entities were flattened completely to primary inputs and outputs using the ILOC canonical form of deepest possible nesting. This permits bottom-up automated identification and analysis of abstract logic structures, available modular decompositions, and existing data vectors, independent of designer-provided information. Some conventional analysis uses the shallowest possible nesting (two-level logic, or SoP forms) for generic logic analysis, which results in large data structures with simple logic structure and complex connectivity. Conventional BDD forms do not provide accessible models of actual logic layout. The ILOC canonical form uses small complex logic data structures, the simplest connectivity, and maintains a strict mapping to the final circuit structure.

7. XOR and MUX structures were abstracted across all entities. Abstraction of elementary branching components controls the explosive growth of logic and wiring. Conventional techniques cannot identify abstract logic patterns globally.

8. Structure sharing was maximized across all entities to reduce logic area. Conventional silicon architectures must manage fanout during structure sharing, while the CoMesh architecture is not sensitive to fanout. Performance in conventional reconfigurable silicon is significantly reduced by fanout from the fine-grain LUT logic structures. In CoMesh, between cell wiring within each block is pre-optimized, so that the worst-case block traversal time is 1.8ns at .18u. Due to pipeline coordination, CoMesh blocks cannot be individually timed, even though signals may propagate through one specific block much faster than through another. CoMesh blocks perform very much like simple PLAs, but with five levels of logic instead of two. Block logic organization is quite different than conventional product term approaches in that logic is organized in deeply nested groupings that minimize wiring and

functional expressability. Block logic mapping is quite different than mapping to LUT structures in that cells within the same block can be assigned to functionally independent components, alleviating the need for both specialized logic processing structures such as carry chains and for constrained logic packing in LUT-sized groupings.

9. The resulting logic forms were technology mapped to three models: two-input NAND gates, inverters, and FFs only; N-LUTs and FFs only; and CoMesh blocks and block-neighborhoods. These technology maps were then used for comparison metrics. Since ILOC and CoMesh are intimately interconnected, the mapping to NAND gates and to LUTs significantly decreases the optimization advantages of ILOC. Since NAND gates and LUTs impose more constraints than CoMesh on a design layout (such as fanout limits, depth of nested logic limits, and library cell limitations), these metrics decrease the performance advantages of CoMesh.

The logic placement and routing process for CoMesh is determined entirely by the ILOC software. During optimization, logic structures are first completely decomposed into elementary relational units. These units are then clustered to form pattern-based groupings accommodated by an individual CoMesh cell. Then cells in a block are filled with no consideration of the specific logic or the wiring relations between the logic. Finally, filled cells are relocated to fit into the 16x5 cell blocks, again paying no attention to the logical functionality. Valid cell relocation moves do not change logic functionality, and are thus determined solely by available block cell and wiring resources.

For routing between blocks, each block is treated as a single "black-box" unit. Blocks that share input/output signals are placed directly adjacent to each other. Inter-block routing congestion is avoided by a software vectorization process prior to technology mapping that identifies groupings of signals and bundles them into a single unit. Thus CoMesh routing is quite different than conventional routing in that switching matrices are avoided by course grain logic module grouping and direct wiring between blocks within a 16-block neighborhood.

CONSTRAINTS ON THE ANALYSIS

1. RAM entities and those entities that incorporated them were not parsed into ILOC or modeled (21 entities). None are contained in the SAMMEL_SPS or ARITHM modules.

2. The parsing of a few entities was questionable, because of missing references, possible linking errors, and cell library elements that were not modeled structurally. These were not included in the analysis (fewer than 10 entities). None are in the SAMMEL_SPS or ARITHM modules.

3. Files that included behavioral VHDL were not parsed into ILOC, with the exception of cell libraries, which were translated from behavior VHDL into internal ILOC patterns by hand.

4. The type hierarchy and records were flattened to standard logic vectors.

5. The wide variety of FFs were not optimized, but rather modeled as is, and then treated as rigid design elements. For the CoMesh analysis, all FFs were converted into a single format.

6. No specialized, or targeted local, reductions were incorporated, such as minimizing a particular critical path or a particular cell type. No attention was paid to pin locations and i/o configurations.

7. The 363 entities parsed into ILOC were hierarchically expanded as components of the largest entities using them. This resulted in 203 separate entities that covered the entire structural design. This design was atypical in that very few entities were used multiple times. The design had almost no modularization, thus it was reasonable to hierarchically expand all component entities.

8. The SAMMEL_SPS and ARITHM modules were selected for initial analysis using criteria of representivity, tractability and ease of processing. The Synplicity log for SAMMEL_SPS incorporates only LUTs, making comparison easier. The log for ARITHM includes mapping to various specialized features within the Virtex architecture, such as MUXF5, CCU_ADD and CCU_AS. These features provide a Virtex CLB with capabilities, such as wide functions and carry chains, that are not efficiently achieved using LUTs alone. The cost is often wasted silicon resources. None of these features are modeled in the ILOC parsing to LUTs, and they are not necessary for the CoMesh architecture.

9. The ILOC software is not optimized for performance, and in fact is highly redundant in order to guard against implementation errors. CPU time for the analysis should decrease at least an order of magnitude when the code is industrialized. Total CPU time for all analyses (parsing and reducing 95% of the SP700 design, and technology mapping 35% of the design) was under 6 CPU hours.

CAUTIONS

1. ILOC reduction may have removed some components of the design that were technology specific.

2. ILOC reduction may have removed some components of the design that are not used, but may have been intended to be used later, or are intended to be included redundantly.

3. Since no test-vectors were available, reduction was verified only between ILOC transformations. The parsers to and from ILOC have not been verified.

4. Comparison metrics that require forms not native to ILOC and CoMesh significantly undermine the advantages of ILOC and CoMesh. The ILOC tools solve design problems by casting them into a different form. Back-translating the ILOC and CoMesh solutions into formats that are the source of the original problems reduces the benefits of the tools.

5. The underlying model for the LUT comparison metrics from the Synplicity log is not known. In SAMMEL_SPS for example, several entities are not included by Synplicity, while much

control logic is embedded into specialized structures within the Virtex CLB. Thus the comparison of performance is somewhat inexact.

6. The compile logs use different Virtex mappings, each with slightly different results. Thus different comparative statistics are used depending on the basis of comparison.

7. Several minor anomalies were encountered while processing the complex ARITHM structure. Although the Synplicity log identified dangling and disconnected components, the optimization process apparently did not carry the implications of these disconnects to their conclusion. ILOC did so, resulting in selective deletion of trees of logic structure that propagated through the entire design module.

WORK IN PROCESS

About 95% of the design has been parsed into ILOC and placed-and-routed on the CoMesh architecture without applying any optimization transformations. Detailed reduction and analysis of the entire design has not been completed. The SAMMEL_SPS and ARITHM analyses are prototypical, intended to clarify and validate comparison techniques and metrics.

Demonstration and validation of ILOC reduction capabilities can be achieved quickly and efficiently by applying the prototypical analytic processes developed for SP700 to the post-synthesis EDIF formats of more well-understood designs. In this light, the internal test validation suite for ILOC consists of over 200 diverse circuits with test vectors. These circuits consist of most common design elements (arithmetic functions, register structures, FSMs, and commonly used larger design macro functions), as well as a rigorous collection of pathological and difficult logic structures for synthesis tools.

The SP700 design analysis is proof in principle that ILOC can be applied to complex commercial designs. The analysis falls short, however, in providing clearly interpretable results upon which to base business decisions. A more rigorous design methodology and process is currently being developed.

APPENDIX: SUPPLEMENTARY DATA

The Appendix includes supporting detailed information about reduction of the SP700 design modules. Tables in the Appendix include:

Description of the SP700 Design
Details of LUT Mapping
Before and After Reduction Data for Component Entities
Pre and Post Reduction Data for Submodules, Including Critical Paths
    ARITHM Sequential Entities and Submodules
    ARITHM Combinational Entities and Submodules
Reduction Gains Due Solely to Flattening the Design Hierarchy
CoMesh Block and Silicon Area Requirements

DESCRIPTION OF THE SP700 DESIGN

The top-level modules of the design, and their storage requirements are:

FILE	MB	KB-ILOC	ENTITIES	INSTANCES	% OF DESIGN
ADR_BUSS_UNIT	.5	116	24	2599	
ADV	.5	96	15	1530	
ARITHM	4.3	1200	116	26251	36
BEFADRRW	.1	32	10	600	
BUSSWITCH	.3	116	13	1587	
COR100	2.1	712	84	12250	
OPADRRW	2.6	832	39	13273	
PIPE_012	1.6	444	38	8346	
RAM	0	24	6	12	
SAMMEL_SPS	.3	72	16	1359	2
SAMMEL1	.1	28	8	563	
ZEIT_SYSTEM	1.1	336	40	5356	
SUM	13.5	4008	409	73726	

The design files consist of

17 structural VHDL design files
5 behavioral VHDL design files
35 cell-library and type hierarchy files
6 project summary and log files

The 17 structural VHDL design files consist of

409 entities
375 cell-components
73726 gate instances, including complex gates and FFs

DETAILS OF LUT MAPPING

The tables below show a summary of the Synplicity log generated by technology mapping to the Virtex architecture for the two analysis modules. Data is from file sp700_api__l1e0n0j9.sum.

The Synplicity log reports area in terms of "gates", which have the following conversion table:

```
one   FF = 1    gate
one 4LUT = 1    gate
one 3LUT =  .5  gate
one 2LUT =  .25 gate
```

For the SAMMEL_SPS module, the number of FFs, 2LUTs, 3LUTs, and 4LUTs to implement the design entity are presented. The GATES total is a weighted sum of the LUT total. The Synplicity log also reports the number of LUTs in each type used to implement the design, and a raw total number of LUTs used. The raw LUT total reflects the layout constraint that any LUT, regardless of number of inputs, uses a physical 4LUT on the Virtex chip. The weighted sum, in contrast, tracks routing usage, since a 2LUT requires two, not four, input ports, even when the logic is implemented by a 4LUT. Both raw and weighted LUT totals are reported for the ARITHM module.

Synplicity counts one gate for each register, regardless of the complexity of the register (for example, some registers are simple DFFs, while some incorporate enables and resets). In the ILOC analysis, registers were decomposed such that every register was a simple DFF, while the additional logic internal to a register (such as reset and enable logic) was moved to the general logic of the circuit. This choice was made since ILOC is customized to optimize to the CoMesh architecture, and the model of registers for that architecture treats register logic separately. As a consequence, the LUT mapping by ILOC includes a disproportionate number of 2LUTs and 3LUTs that are actually register logic. In the table below, the actual LUT count for the ILOC LUT mapping is presented, and then adjusted by removing the register logic from the gate totals. Similarly, the gate totals from the Synplicity logs have been reduced by the register count. The net result is that the gate totals in the LUT comparison tables below compare logic to logic, without including registers for either Synplicity or ILOC.

For the ARITHM module, Synplicity log results are reported for DFFs and GATES. One gate is one 4LUT or one FF. In the table, the DFF and the logic gate counts have been kept separate. The PRE-GATE column lists the results from the Synplicity log. The POST-GATE column lists results from ILOC reduction followed by 4LUT mapping.

Utilization of the Virtex CLB special logic features, primarily MUXes, has been adjusted by adding a small number to the pre-reduction gate count. The GATES total has been reduced by the number of FFs, so that the entry represents solely logic implementation using LUTs. The FF-adjustment column shows the additional logic that was required to move register control logic outside of registers. Gains are listed in both absolute and percentage terms. Absolute gain is the difference between the PRE-GATE count (with the extra-CLB logic added in) and the POST-GATE count. The FF-ADJUSTMENT has already been subtracted from the POST-GATE count.

1LUTs, INVERTERs, and GROUNDs are not reported. Any inconsistencies in these data tables other than clerical error are probably due to the interpretation of how data was accumulated hierarchically in the Synplicity log.

SAMMEL_SPS

ENTITIES	DFF	2LUT PRE/POST	3LUT PRE/POST	4LUT PRE/POST	LUTS PRE/POST	GATES PRE/POST	FF/MUX ADJUST
GRAND TOTAL							
SAMMEL_SPS	320	114 240	180 277	375 380	669 897	449 503	254/27
adjusted					616	413	
% reduction					8%	8%	
SAMMEL_SPS	85	18 62	45 53	47 35	110 150	70 75	49/6
adjusted					95	57	
INTCON	136	63 102	56 88	155 154	274 344	172 169	134/5
adjusted					205	131	
KLC	99	33 76	79 136	173 191	285 403	207 259	71/16
adjusted					316	225	

ARITHM

	DFF	2LUT PRE/POST	3LUT PRE/POST	4LUT PRE/POST	LUTS PRE/POST	GATES PRE/POST	FF/MUX ADJUST
GRAND TOTAL							
ARITHM LOG	1847	1341	1941	11262	14544	12578	
1706/1710							
ILOC		2301	4376	11534	18211	10416	
adjusted					14803	7839	
% reduction					-2%	37%	
ENTITIES							
ARITHM LOG	74	57	61	235	353	202	
74/1							
ILOC		63	80	172	315	184	
adjusted					240	163	
WERKE LOG	227	334	404	1246	1984	1402	
117/646							
ILOC		307	576	1277	2160	1310	
adjusted					1397	954	
ARI LOG	405	311	588	1980	2879	2302	
405/174							
ILOC		399	907	2077	3383	2201	
adjusted					2804	2000	
ARITH LOG	1141	639	888	7801	9328	8672	
1110/889							
ILOC		1532	2813	8008	12353	6721	
adjusted					10362	4722	

The above tables report numbers of 2, 3, and 4LUTs generated by the Synplicity and by the ILOC LUT mappers. Data is provided for several sub-modules to illustrate the relative consistency of results. ILOC data is provided in raw form, prior to adjustments to remove register logic and to compensate for use of specialized CLB logic. Data is also provide after these adjustments. The FF/MUX ADJUST column shows the number of FFs and the number of wide MUXes used for ILOC adjustments

BEFORE AND AFTER REDUCTION DATA FOR COMPONENT ENTITIES

The tables below report specific results for different submodules using the ASIC gate and internal wiring metrics, before and after optimization. The before data is measured directly from the structural VHDL netlist for the design. The after data is measured after ILOC reduction algorithms have been applied. Algorithms were applied to the design as a whole, no algorithms were applied separately to submodules in the design.

In the tables below, care should be taken reading the nested hierarchical components of the design. The design hierarchy is indented on the left, with composite entities farthest to the left. The hierarchical components have been flattened in the composite entities, so that the results for each component represents the *sum* of the entities it contains.

The reason for reporting the lower-level entities is to show that even without logic reduction, ILOC optimization reorganizes the entity structure to have fewer internal pins and fewer wires. A significant part of the wire net gain is in eliminating the redundant port connections between each individual entity.

SAMMEL_SPS

DESIGN ENTITIES		NAND GATES		INPUT PINS		WIRE NETS	
	DFFS	PRE	POST	PRE	POST	PRE	POST
SAMMEL_SPS TOTAL	346	2592	2434	4778	4306	3944	2579
INTEXT	52	173	120	399	266	381	194
INTCON	132	1022	894	1987	1655	1487	1047
INTPRI		198	174	331	310	328	264
KLC	100	1265	1127	2309	1864	1917	1125
KLC_STW	9	148	136	273	250	248	160
KLC_CNT	16	539	439	968	543	764	359
KLC_CNT_DW01_ADDSUB_16_0		284	204	472	216	316	172
KLC_LTE		60	47	116	89	101	72
KLC_LTE_DW01_DEC_5_0		13	13	26	18	31	22
KLC_REG	75	344	388	727	711	642	525
KLC_REGS		33	33	72	72	80	74
KLC_DBOUTMUX		165	165	300	293	385	356
KLC_CAS		202	170	381	266	355	210
KLC_CAS_DW01_CMP2_16_0		61	61	118	109	151	11
KLC_CAS_DW01_CMP2_16_1		61	61	119	109	151	112

The ARITHM logic reduction tables reports before and after reduction measures on three metrics, internal pins (literals), internal wires (nets), and two-input ASIC gates. These hierarchical tables are cumulative. Data rows that are carried forward in duplicate is marked by the word SUM, together with the hierarchy level the row entity comes from.

The "fully flattened" row is a separate accounting of logic complexity when the entire group of entities is combined into a single entity with no hierarchical or structural boundaries. The ILOC algorithms applied to flattened groups of entities are more efficient in all cases and for all metrics. The following table is the raw comparison date for the ARITHM module.

ARITHM

	PRE/POST PINS		PRE/POST WIRES		PRE/POST ASIC GATES		
ARITHM	179	143	484	330	132	99	
ARI_STEU_REG	1323	1071	1189	924	798	693	
WERKE	9923	7219	10749	8440	7064	5911	SUM-3
ARI	83452	55527	90054	57980	61158	45353	SUM-4
ARITHM GRAND SUM	94877	63960	102476	67674	69152	52056	
LEVEL 4 HIERARCHY							
ARI	4	4	313	240	2	2	
FP_DECODE	74	61	56	44	44	36	
ANZG	396	243	432	249	267	182	
EX	2299	1574	2380	1379	1668	1237	
ari/RS	1253	784	2106	1410	911	769	SUM-1
ari/AL	2033	1539	2541	1407	1465	1151	SUM-2
ari/SH	4029	3017	4738	3769	2770	2437	SUM-2
ari/AC	5089	4202	6397	5232	3188	2932	SUM-3
ari/ARITH	68275	44103	71089	44250	50843	36607	SUM-3
ARI	83452	55527	90054	57980	61158	45353	SUM
LEVEL 3 HIERARCHY							
WERKE	44	39	223	211	33	29	
KL_OUTMUX	99	77	121	114	66	63	
WERK_OUTMUX	208	186	313	326	150	147	
P_WERKE_REG	622	524	649	427	375	336	
W_PHI2	223	156	273	143	148	109	
werke/W_PHI1	1181	942	1364	1150	851	758	SUM-2
werke/CONV	7546	5295	7806	6069	5441	4419	SUM-2
WERKE	9923	7219	10749	8440	7064	5911	SUM
Fully Flattened	9337	6759	8625	6216	6706	5751	
ari/AC	0	0	189	187	0	0	
AC_MX	53	50	88	77	34	32	
SPL_IMMED	217	147	301	178	153	116	
ari/ac/AKKU	4819	4005	5819	4790	3001	2784	SUM-2
ari/AC	5089	4202	6397	5232	3188	2932	SUM
Fully Flattened	5068	3918	4527	3559	33182	2784	

ari/ARITH	183	129	675	363	149	99	
FLAG_BASIC	424	343	445	312	337	312	
FP_RET	253	200	287	209	182	153	
PRI	854	635	1040	764	622	520	
MX_SH_ADD	1041	833	1672	1373	813	766	
SL_BUSAB	80	71	113	93	54	49	
MX_MD	9693	2602	8013	2855	6944	2259	
ARI_STEU	738	651	683	461	458	397	
MANT_LATCH	67	99	67	67	32	32	
SQRT	1689	1069	1795	921	1329	924	
STEU_ADSU	36	27	31	17	17	12	
STEU_SQRT	55	50	59	31	32	28	
CORD	10491	8349	10232	7289	7232	6517	
ari/arith/MULT	19501	11575	20075	11275	16116	11022	SUM-1
ari/arith/FINIEEE	2541	1692	3322	2246	1940	1408	SUM-1
ari/arith/SH_ADD	2702	2158	3113	2350	2713	1825	SUM-2
ari/arith/DIV	9190	6300	9135	5879	6177	4933	SUM-2
ari/arith/EXLN	8741	7320	10332	7745	5696	5351	SUM-2
ari/ARITH	68275	44103	71089	44250	50843	36607	SUM
Fully Flattened	66193	43357	63011	38695	48986	36075	

PRE AND POST REDUCTION DATA FOR SUBMODULES, INCLUDING CRITICAL PATHS

The table below reports specific results for submodules of the ARITHM module, before and after optimization. *It has been extended to include critical path lengths*, for all submodules. Critical paths are provided for two different technology maps: two-input NAND gates, and 4LUTs. Submodules are sorted into combinational and sequential categories.

Pre-optimization entries in the two tables that follow differ slightly (by 1-2% total) from those reported directly above. In this tabulation, grounds, 1LUTs, and duplicate logic have been removed from the pre-reduction statistics, under the assumption that such simple transformations would occur in any reduction system.

The critical paths are measured using a unit delay metric, essentially counting the maximum number of two-input NAND gate unit delays (or 4LUT unit delays) between registers, or between input and output, for each submodule. The tables also report before and after reduction measures on three metrics: internal pins (literals), internal wires (nets), and two-input NAND gates.

Although module entities are hierarchical, within each category, no submodules contain duplicate entities. Sequential entities do contain some combinational entities, so that summing across both tables will result in counting some combinational entities more than once. The totality of all submodules and entities completely covers the entire ARITHM module. Combinational entries marked with an asterisk are not included in the sequential gate counts, all others are. Submodules with prefixed hierarchical nesting information (such as `ari/...`) are composite; those entities without hierarchical nesting prefixes are single entities at the lowest level of the ARITHM entity hierarchy. Composite submodules have been fully flattened so that critical path lengths are for the entire submodule.

ARITHM Sequential Entities and Submodules

ENTITY	PRE/POST PINS		PRE/POST WIRES		PRE/POST NAND PATH		PRE/POST LUT PATH		PRE/POST NAND GATES	
werke/W_PHI1	1171	911	1199	973	29	23	25	14	839	752
werke/CONV	6998	4798	6368	4514	232	167	132	110	5109	4199
ari/AC	5068	3918	4527	3559	113	92	118	94	3182	2784
ari/arith/DIV	8957	6059	8422	5345	244	223	120	119	6075	4795
ari/arith/MULT	19501	11575	18682	11568	101	121	55	70	16078	11022
ari/arith/EXLN	8283	7101	8405	6267	262	200	218	196	5403	4968
ARI_STEU_REG	1308	1071	1165	924	13	15	8	10	785	693
P_WERKE_REG	622	524	626	427	52	31	23	26	375	336
W_PHI2	215	156	269	143	4	4	3	2	144	109
ANZG	394	243	430	249	14	12	12	8	266	182
EX	2290	1574	2367	1379	33	29	21	18	1661	1237
ARI_STEU	726	651	651	461	64	64	62	51	451	397
SQRT	1630	1069	1706	921	79	79	67	93	1271	924
CORD	10368	8349	10070	7289	275	210	226	227	7213	6517
MANT_LATCH	67	67	67	67	2	2	2	2	32	32
STEU_ADSU	29	27	26	17	6	6	6	4	12	12
STEU_SQRT	54	50	50	31	14	12	11	9	31	28

ARITHM Combinational Entities and Submodules

ENTITY	PRE/POST PINS		PRE/POST WIRES		PRE/POST NAND PATH		PRE/POST LUT PATH		PRE/POST NAND GATES	
ari/RS*	1253	750	1776	1107	17	13	11	13	911	769
ari/AL*	2059	1502	2256	1203	78	77	38	49	1452	1185
ari/SH*	3983	3003	4187	3065	48	37	34	24	2759	2525
ari/arith/FINIEEE*	2481	1684	2858	1805	39	31	35	31	1913	1401
ari/arith/SH_ADD*	2643	2132	2852	2098	88	86	35	80	1966	1839
werke/w_phi1/TIZA	332	276	337	320	13	12	10	9	244	232
werke/conv/BCD	1475	919	1264	845	26	21	16	15	1143	832
werke/conv/BIN	1169	672	1013	648	16	14	9	11	931	668
FP_DECODE*	74	61	54	44	6	5	12	8	44	36
KL_OUTMUX*	99	77	117	114	6	5	11	4	66	63
WERK_OUTMUX*	208	186	312	326	6	6	11	5	150	147
AC_MX	53	50	87	77	4	4	3	3	34	32
FLAG_BASIC*	424	343	445	312	15	10	13	15	337	312
FP_RET*	253	200	287	209	11	9	9	7	182	153
PRI*	835	635	1031	764	28	28	18	17	618	520
MX_SH_ADD*	1041	833	1662	1373	17	10	11	7	813	766
SL_BUSAB*	80	71	110	93	7	7	9	5	54	49
MX_MD*	8853	2602	7642	2855	8	9	5	5	6537	2259
BIT_NETZ	258	188	242	189	11	9	8	6	187	150
NUEUS_BIT	278	216	340	243	18	14	13	11	209	183
BCD_FEHLER	260	179	335	212	13	9	9	6	192	148
CONV_IN_MUX	272	183	368	178	8	7	7	4	190	154
CONV_OUT_MUX	399	359	649	549	9	7	7	4	311	309
P_BANZ	55	32	75	29	10	6	7	4	41	27
P_MOD_A	245	207	329	258	13	9	8	7	189	177
P_S5T2T	202	178	276	232	6	5	5	4	157	141
P_T2S5T	180	140	235	156	8	6	7	5	133	118
AC1_MOD	594	440	714	510	12	13	10	8	460	356
OMUXA	575	450	904	745	5	3	5	3	417	417
OMUXB	388	277	358	454	4	3	3	2	245	245
EXLN_MD	273	271	463	460	6	6	5	4	196	193
EXLN_MX	432	312	770	476	4	4	3	2	337	257
P_HEXCONV	154	154	285	284	4	4	4	3	120	120
B_DT2D	307	226	249	198	13	9	8	6	210	203
B_SUB28	545	429	449	410	11	9	7	7	373	363
D_NETZ	5432	3552	5315	3345	70	86	158	129	3816	2999
TAB_EXPLN	637	581	701	538	8	7	7	5	429	405
ADD40_1	640	490	636	451	85	83	42	42	445	435
ADD40_2	640	490	636	451	85	83	42	42	445	435

REDUCTION GAINS DUE SOLELY TO HIERARCHICAL DECOMPOSITION

The following data is taken from the previous table, comparing logic complexity prior to reduction, after reduction of each entity separately, and after reduction of a group of entities combined to remove hierarchical boundaries. Gains are generally very modest.

	PRE/POST-ASIC		FLATTENED	GAIN	%GAIN
WERKE	7195	5994	5751	253	4
ari/AC	3316	3060	3040	20	1
ari/ARITH	50807	36671	36075	596	2
werke/W_PHI1	851	758	752	6	1
werke/CONV	5441	4419	4199	220	5
ari/AL	1465	1151	1148	3	0
ari/SH	2770	2437	2432	5	0
ari/ac/AKKU	3001	2784	2637	147	5
ari/arith/SH_ADD	1979	1825	1824	1	0
ari/arith/DIV	6177	4933	4811	22	0
ari/arith/EXLN	5696	5351	4968	383	7
werke/w_phi1/TIZA	252	231	232	0	0
werke/conv/BIN	955	667	668	0	0
werke/conv/B_HEXCONV	1982	1787	1724	23	1
werke/conv/BCD	1279	891	832	59	7
ari/RS	911	769	769	0	0
ari/al/E_ALU	1364	1060	1057	3	0
ari/sh/SHIFT	1749	1528	1508	20	1
ari/ac/akku/ACCU_REGS	1394	1365	1365	0	0
ari/arith/MULT	16116	11022	11022	0	0
ari/arith/FINIEEE	1940	1408	1401	7	0
sh_add/SHIFT_TR	1455	1321	1322	0	0
ari/arith/div/D_ARRAY	3979	3067	2956	111	4
ari/arith/exln/PSEUDO	2593	2513	2148	365	15

COMESH BLOCK REQUIREMENTS

Cumulative CoMesh block requirements for the ARITHM entity hierarchy are presented below. The CONV entity is marked (nominal) due to specialized Virtex elements. Its data is an average for entities of similar size. Data is provided both before CoMesh placement and routing optimization and after routing optimization to increase logic density.

ENTITY	BEFORE	AFTER	
TOTAL BLOCKS	1116	713	
ARITHM	1	1	
ARI_STEU_REG	10	8	
WERKE	160	75	(nominal)
ARI	945	629	
LEVEL 4 HIERARCHY			
ARI		629	SUM
FP_DECODE	1	1	
ANZG	5	3	
EX	25	14	
ari/RS	13	8	
ari/AL	34	25	
ari/SH	49	35	
ari/AC	48	32	
ari/ARITH	770	511	
LEVEL 3 HIERARCHY			
WERKE		74	SUM
KL_OUTMUX	2	1	
WERK_OUTMUX	6	4	
P_WERKE_REG	7	4	
W_PHI2	4	2	
werke/W_PHI1	14	10	
werke/CONV	120	53	(nominal)
ari/AC		33	SUM
AC_MX	2	1	
SPL_IMMED	5	3	
ari/ac/AKKU	51	29	
ari/ARITH		510	SUM
FLAG_BASIC	9	3	
FP_RET	4	2	
PRI	13	8	
MX_SH_ADD	12	9	
SL_BUSAB	2	2	
MX_MD	48	24	
ARI_STEU	7	5	
MANT_LATCH	2	1	
SQRT	39	23	
STEU_ADSU	1	1	
STEU_SQRT	1	1	
CORD	110	78	
ari/arith/MULT	205	141	
ari/arith/FINIEEE	26	19	
ari/arith/SH_ADD	52	30	
ari/arith/DIV	150	98	
ari/arith/EXLN	89	65	