

# INTEGRATING LOGIC SYNTHESIS INTO A FULL CHIP ASIC DESIGN SYSTEM

Robert V. Alessi  
Barry Roitblat

Seattle Silicon, Corporation  
3075 112th Avenue NE  
Bellevue, WA 98004

## Abstract

Although logic synthesis provides a powerful tool for creating optimized logic for state machines and combinational logic, other parts of the circuit may be better described in traditional structural terms. In addition, logic will often be more efficiently realized when physical design characteristics are taken into account. This paper describes a design system which integrates logic synthesis with structural tools to give highly efficient design for multiple function ASICs.

## 1. Introduction.

Random logic and state machines, while usually representing a small part of complex designs, may often require a more significant amount of the design time. In addition, unless carefully optimized (and manual optimization is a tedious, error prone process), the logic may consume a disproportionate percentage of the chip area.

Logic synthesis can address both of these problems. Through logic synthesis, specifications of a block's logical behavior can be automatically mapped into structures optimized for area and delay efficiency.

Logic synthesis cannot solve the entire design process alone, however. Memory or LSI elements, such as an ALU, multiplier, or other bus oriented function, are generally more efficiently laid out as higher level blocks than composed from logic level elements. In addition, efficient realization of the logic itself may be dependent on the specific application. Multiple function ASICs can benefit, therefore, from combining the more traditional schematic based design methodologies with logic synthesis. By integrating physical design and, therefore, detailed knowledge of the target hardware into the logic synthesis system, it is possible to assure optimal performance within a given set of area constraints (or vice versa).

The remainder of this paper discusses how logic synthesis can be integrated in a full chip ASIC design system to maximize the design process. The discussion is based around the specifics of ChipCrafter, an ASIC design system offered by Seattle Silicon.

## 2. The Design Environment

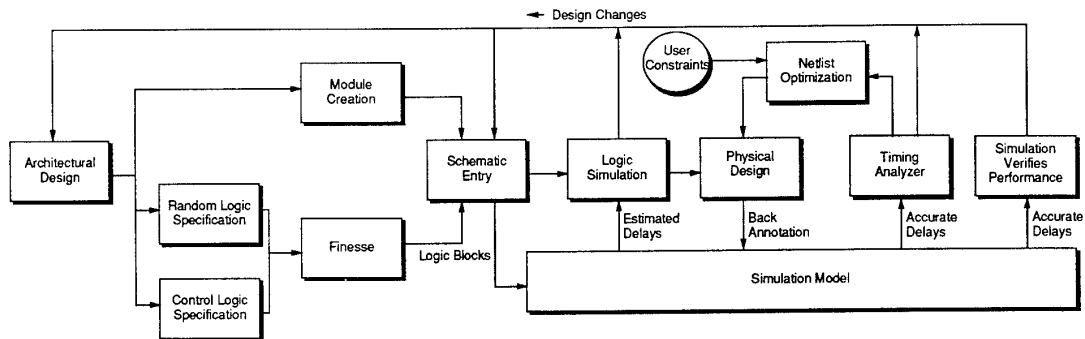
ChipCrafter is a workstation-based design environment combining block and logic level design techniques with full automated physical design and analysis.

The system includes a Design Data Management System that provides a variety of services for managing design data. Each data element may have several views representing the specification, physical geometry, netlist, timing, power, simulation, and schematic symbols of the element. The elements are then organized into a combination of projects and shared libraries.

Designers using the ChipCrafter system may select from a wide library of function blocks from simple gates and flip flops to memories, multipliers and other higher level functions. The blocks are selected functionally through a graphical user interface. Each of the views discussed above is created dynamically as the module is selected. Control logic may be entered behaviorally as state machines or logic equations through the Finesse language, a part of Seattle Silicon's logic synthesis capabilities. The same set of views are generated for a Finesse block.

The function block generators create cells optimized for each of the commercial semiconductor manufacturers' processes available on the system. Layout data is derived from each of over 100 independent design rule micros. Performance data is calculated from process specific parameters provided by the manufacturers.

Figure 1: A Full Chip ASIC Design Process.



ChipCrafter is integrated with standard schematic entry and simulation software offered by workstation vendors Mentor Graphics and Valid Logic. Schematic entry is used to structurally specify a chip net list (Finesse blocks are placed on schematics and treated like any other ChipCrafter block). Once the schematic is entered, the user proceeds to logic simulation to verify the functionality of the design. Errors discovered at this point may drive changes at the architectural or schematic level.

Chip composition is automatically completed by a suite of place and route tools which run on the user's workstation after the functional design has stabilized. There are separate tools optimized for the standard cell, block, datapath, analog, and pad place and route problems, as well as the necessary tools to tie the results together. In addition, power rails are automatically sized and tapered based on power consumption and distribution, and buffers are automatically sized for optimal delay characteristics.

After physical design, the routing parasitics and interconnect delays are automatically back annotated into the simulation database for accurate delay information on the workstation. In addition, Seattle Silicon provides a static timing analyzer for more detailed exploration of a chip's critical paths. The results of timing analysis may be used to drive subsequent passes through the logic synthesis (netlist optimization) and physical design (placement and net weighting). The back annotated database is also available for resimulation to verify performance before releasing the design for fabrication.

Figure 1 summarizes the chip design process using ChipCrafter.

### 3. Logic Synthesis

For logic synthesis, ChipCrafter provides Finesse, a hardware independent high level language for specifying state machines and random logic. The language description is automatically mapped into any of a variety of hardware implementations following the application of logic optimization and minimization techniques.

The Finesse language provides many constructs that allow the user to describe the behavior of their design at a higher level of abstraction. These include special constructs such as symbolic variable and state declarations, default behavior specification, macros, memorized variables, and busses. Finesse inputs are constrained to be deterministic, synchronous finite state machines. Limited asynchronous (initialization) capability is provided for, and combinational logic, expressed as logic equations or symbolic truth tables, is also supported.

Symbolic variables are enumerated types that allow the specification of multiple-valued logic. Symbolic variables may be used with finite state machines, equations, or truth tables and provide a useful level of abstraction from the physical implementation while actually allowing greater facility for logic minimization.

#### Synthesis and Optimization

There are as many as 8 major steps which occur during the process of converting a Finesse language description to optimized target hardware (Figure 2). Optimization is performed at the language, symbolic, structural, and net list levels and may perform multiple optimization steps at each stage. "Don't Care" conditions are recognized or inserted as part of the language synthesis process. Symbolic optimization includes symbolic minimization and optimal state and symbolic variable assignment. Structural optimization includes Boolean minimization, phase assignment, and the

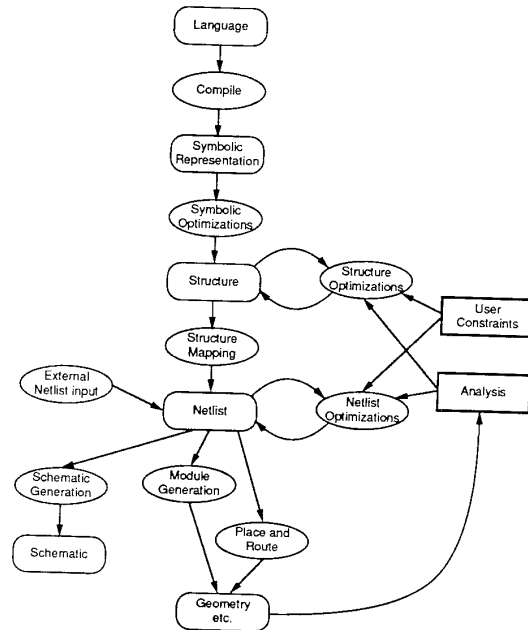


Figure 2: The synthesis and optimization steps.

#### Figure 3: The Finesse hardware model.

Finesse blocks consist of a main control block implemented with a PLA, gates, or ROM, inputs, internal variables, and combinational or memorized outputs. The inputs and outputs may be symbolic, in which case they may be decoded as part of the main control block, or externally with a decoder, pla, gates, or ROM. Symbolic internal variables are always decoded within the main control block.

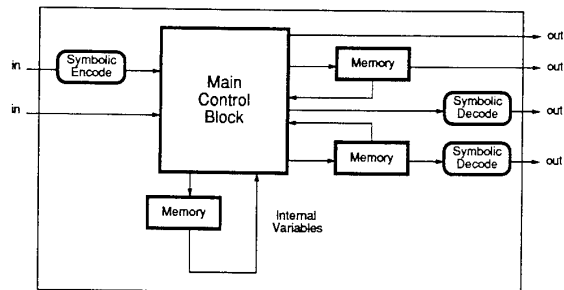
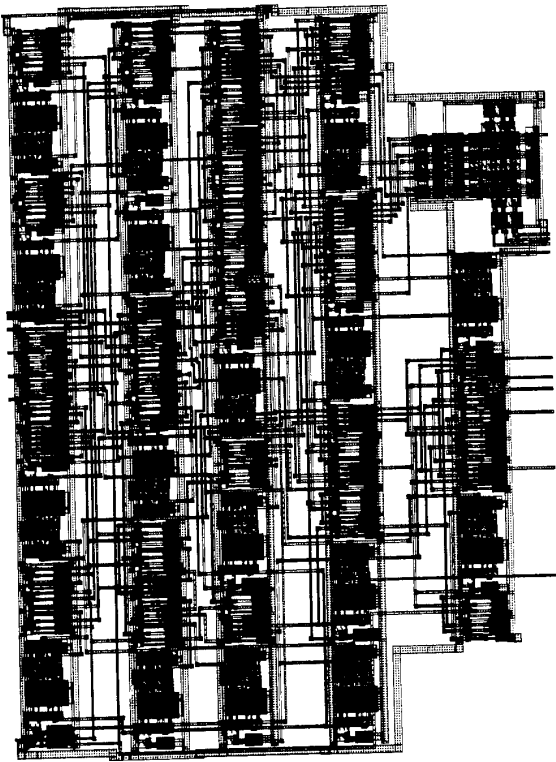
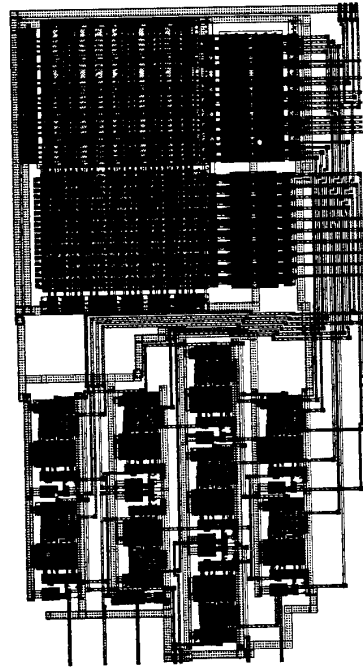


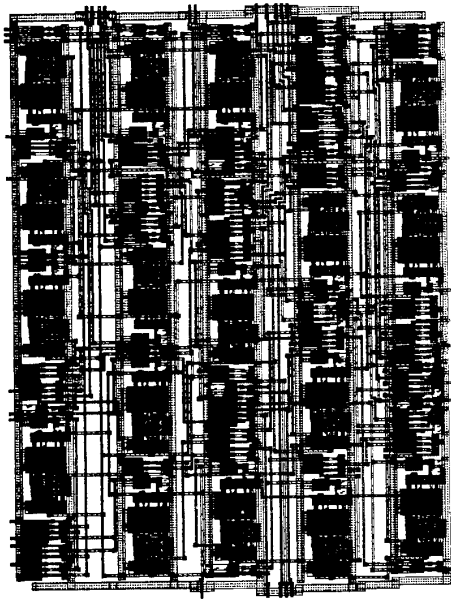
Figure 4: A small microprocessor example.



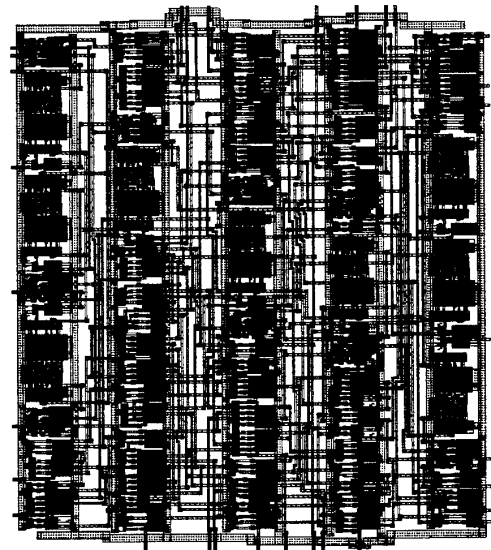
(a) Manual design



(b) Optimized for Area



(c) Optimized for Delay



(d) Including Scan Logic

removal of trivial or redundant signals. Factoring, technology mapping and buffer sizing comprise net list optimization.

After all the synthesis and optimization steps have been completed, Finesse generates the actual logic elements for the selected implementation and invokes a final place and route phase to complete the physical design. Finesse may also automatically insert scan logic (for testability) as part of the structure generation. In addition to the physical layout, simulation, timing, and schematic symbol models are created for the Finesse block.

Finesse can generate many different implementations from the same language description (see Figure 3). Symbolic variables may be implemented either in the main control block, or externally as PLAs, gates, encoders or as decoders. The main control block can be implemented as a PLA or in random logic (plus storage elements for memoried variables). Other target implementations such as microcode architectures are also possible.

Each of the possible implementations may have relative advantages for different sets of inputs and constraints. Integrating the hardware implementation decisions with a full ASIC design system allows effective trade-offs to be made based on area and performance of the target logic elements and actual routing delays for the specific application.

The routing related information is particularly important since interconnect geometry may consume a significant portion of chip area and delay. Fan-out costs and routing parasitics may easily dominate the intrinsic gate delays. Estimates range as high as 70% of a typical critical path delay may be contributed by routing (1.25 micron process). By generating and analyzing detailed geometry, the optimization algorithms may be much more effective in considering and controlling such factors as fan-out, buffer sizing, net weighting and levels of logic.

#### 4. Example

Figure 4 shows an example implementation of a simple, 8 instruction, microprocessor. The state machine for this processor included 5 inputs, 17 outputs, 12 states, and 5 internal flip-flops. The first picture (a) is a straightforward manually specified gate level implementation of the processor (the logic was entered schematically, the layout, place, and route were still fully automatic). It is approximately 23.4 by 31.7 mils (741.8 square mils) and has a critical path delay of about 17.6 nano-seconds. It took nearly 2 days to complete.

The other examples (b) through (d) were automatically synthesized from a Finesse description. They have the same number of inputs, but the synthesis automatically chose to encode the internal variables in 3 flip flops, and the outputs have been collapsed down to 11. Example (b) was constrained to result in the minimal area. The synthesis completed in 50 cpu seconds (Apollo 4500) and resulted in a layout, using a PLA, that was 27.6 by 15 (413.9) with a delay of 41.2 nsec. Example (c) was optimized for delay. Run time was 150 cpu seconds for a layout 24.9 by 19 (474.1) with a delay of 11.6. The constraints for example (d) were chosen for a result in the mid-range of both area and delay and with the addition of scan logic for testability. It completed in 125 seconds and is 22.9 by 20.4 (467.6) mils. It runs in 14.6 nano-seconds.

#### 5. Conclusion.

Logic Synthesis can speed the implementation of glue or control logic and allow the user to explore the design space for optimal tradeoffs between area and delay. When coupled with a full chip design system, the synthesis tools can weigh detailed information about actual area and delay, including interconnect area and loading, for more accurate design tradeoffs.

Logic blocks account for only a portion of most designs. It is not enough to optimize the logic in isolation. An integrated system, which may consider the logic in the context of the entire design, may be able to achieve a significant improvement over chip layout or logic synthesis accomplished separately.

Table 1: Summary of Statistics for Example Designs

Implementation	Area	%	Delay	%	Design Time
Manual Design	741.8	100	17.6	100	2 days
Optimized for Area	413.9	56	41.2	234	50 cpu seconds
Optimized for Delay	474.1	64	11.6	66	150 cpu seconds
Including Scan Logic	467.6	63	14.6	83	125 cpu seconds