HARDWARE AND SOFTWARE DESIGN Q&A

William Bricken
August 2002

Software

How does boundary logic simplify synthesis and increase efficiency?

Some example algorithmic and data structure features that make boundary logic both simpler and more efficient include 1) a one-to-many mapping from boundary logic to conventional logic, 2) a unary basis with one operator and one ground state, 3) deletion operations that remove rather than rearrange structure, and 4) transparency operators that ignore intervening logic on a path, effecting path transformations independent of the logic along the path.

Unary logic is a central feature of the boundary logic approach. The 1, or HIGH, logical state is represented explicitly; the 0, or LOW, logical state is implicit as the absence of the HIGH state, and is thus not represented. Discarding one logical state into non-representation creates no algorithmic confusion, and greatly improves the efficiency of all representations and processes. In converting back to conventional logic, we identify where explicit 0 states are appropriate. Unary logic is closer to the way physical circuits behave (presence or *absence* of a signal) than is binary logic. Removing the superfluous and redundant *symbolic* structures from binary logic is central to the simplification capabilities of boundary logic.

Why is a "new understanding of circuit structure" important?

All current EDA tools, including those based on BDDs, Boolean factoring algorithms, partitioning algorithms, Boolean minimization, etc., are substantively ineffective since they lack multilevel Boolean modeling capabilities. A new understanding of logic networks that is simpler and more powerful than existing tools improves all EDA processes. In the case of the CoMesh architecture, a new understanding leads to new hardware architectures with dramatic performance improvements at lower costs.

In what ways are designs improved automatically?

The synthesis software provides world class optimization given pre-specified structural parameters. In the context of our CoMesh architecture, the hardware guarantees a minimal timing, while the software automates timing design.

What are the abstractions the tool provides and why are these useful to the user?

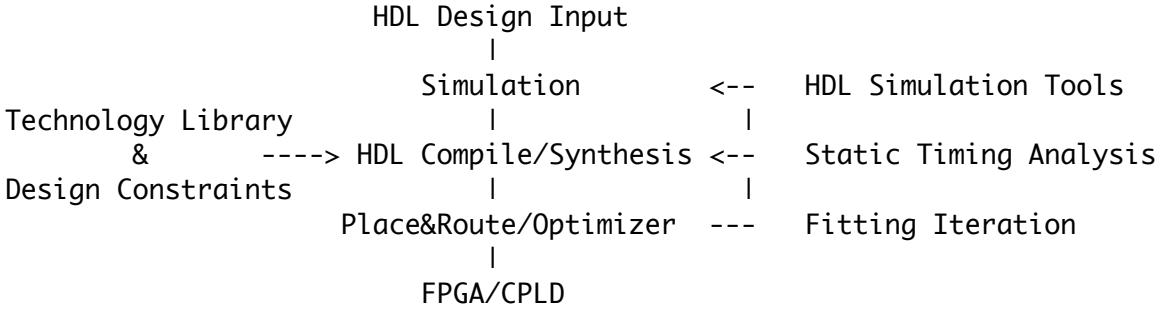
Abstraction capabilities include identification of recurrent functional elements at any grain size, partitioning necessarily sequential from parallel components of the circuit logic, vectorization of functions, control of branchy low level functions such as XORs and MUXes, structure sharing, and hierarchical partitioning. Interactively, a user can work at the function block level to identify mandatory resource requirements, time/space trade-offs, hierarchical design components, available pipelining, available structure and resource sharing, vector parallelization, parametric functional decompositions, partially evaluated structures, and the like. These capabilities give the user an ability to closely control time/space trade-offs in the design; they also provide organizational hierarchy for management of the design.

Are the synthesis tools developed in-house?

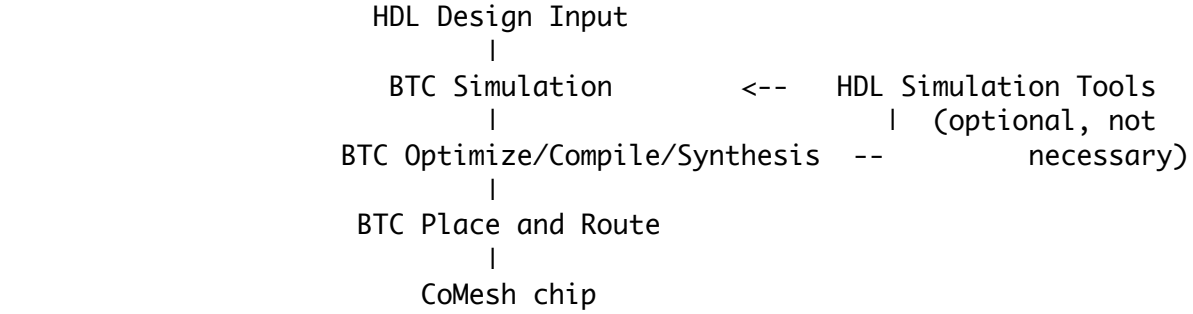
All BTC software has been developed in-house, is completely unique, and no other companies or academic environments have or know of these algorithms.

What is the complete design flow? Compare to conventional FPGA/ASIC flow.

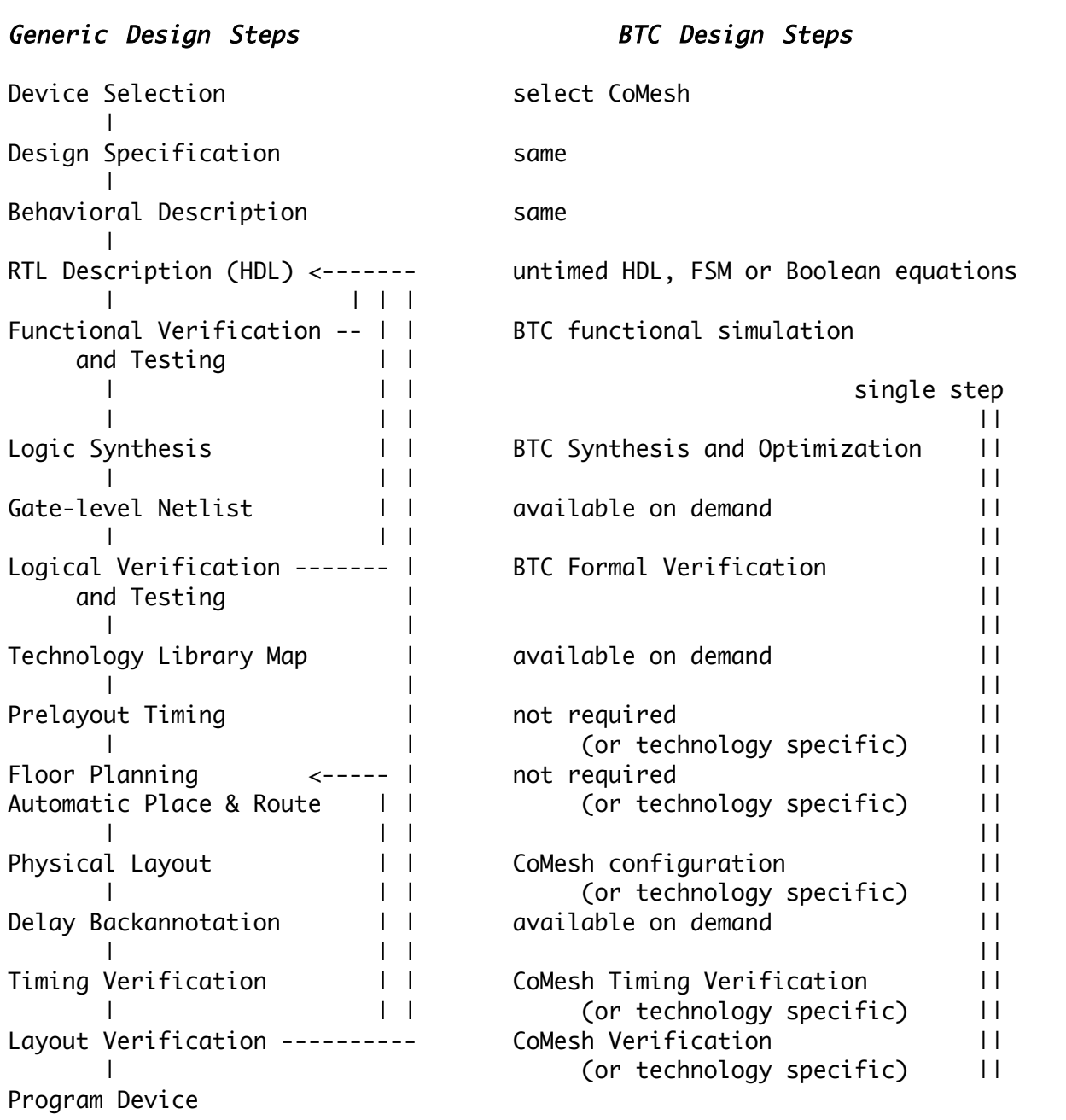
Standard Design Flow for PLDs



Comesh Design Flow



BTC Tool-chain Integration (ASICs and PLDs)



Which low level design steps are removed?

Within the context of a golden circuit specification, a specified technology library and structural circuit performance parameters, the software generates a conforming circuit automatically.

How does the user guide the design and architectural choices the tool makes? How much control does he have?

The software is completely modular; we have not yet decided upon particular interface styles and options. At this time the user provides the desired functionality and sets parameters such as the technology library, fanin and fanout constraints, desired critical path length, testability, logic density, and level of processing effort. The software then generates a circuit that fits within the parameters, given such a circuit is possible.

At a finer grain, the software consists of around 200 independent functions that an experienced designer can mix and match. Each modular transformation achieves specific design trade-offs, so that a designer could specify a subcircuit of interest and apply transformations to that subcircuit that achieve known trade-offs, such as lowering fanout by increasing path length, lowering area by increasing routing, and lowering path length by increasing area. These trade-offs are available at any grain size.

Our architectural strategy is to automate design so well that user choice and control is moot for the majority of users. There are no necessary obstacles to giving the user control over every fine grain step taken by the software (or some reasonable mixture in between). The greatest difficulty in giving the user control is that the algorithms are completely unfamiliar, so that the user would have no cognitive model of what is going on or how automated design decisions are being made. Our experience is that the software can outperform all but the top few percent of designers in arriving at designs that meet specifications. We mitigate the unfamiliarity of the algorithms by providing the user with functional choices expressed as familiar design parameters, such as desired timing and area.

For the CoMesh architecture, fine-grained user control is not preferred. Rather, the user names the functional and performance requirements, and the software generates and routes a hardware solution. It should be understood that portions of this co-designed strategy are fully implemented, and portions are simulated. CoMesh hardware has not yet been fabricated.

What is the design entry language?

We currently have parsers for EDIF 2 0 0, BLIF, KISS, a very limited subset of Verilog, and we can easily enter logic equations, FSM specifications and other formal models. The BTC strategy is to be compatible with industry standard design entry languages, particularly HDLs. HDL parsers have not yet been written; no complications are expected since timed boundary logic maps directly onto timed conventional logic.

Is design entry behavioral or structural?

Design entry is primarily structural at this time, preferably an EDIF netlist. Behavioral parsers can easily be written, specifically for HDLs. We have created proof-of-concept parametric circuit generators for adders, multipliers, comparators and some other functions, and intend to create an extensive library of automated circuit generators for behavioral design entry.

What is the simulation capability of the BTC tools?

Currently we have simulation capabilities for combinational and sequential test vectors with a limited variety of flip-flop types. We also provide *partial simulation*, in which some inputs can be locked-down to specific values. The software then generates the residual circuit given the specified inputs. Thus far, static timing simulation has been developed only for simple unit-delay models and for aspects of the CoMesh architecture. We envisage no complications extending our simulation capabilities given appropriate timing models.

How does one co-simulate HDL and BTC designs?

We have no provision for co-simulation at this time, and anticipate no technical difficulties interfacing BTC tools with conventional CAD tools. BTC's tools do not address design specification. BTC's tools do optimize specified designs in the context of a specific target technology, using standard netlists and test vectors generated from HDL design specifications. In the context of the CoMesh architecture, timing simulation will be provided by the BTC software. In other contexts, BTC synthesis results can be output as a standard netlist at any time, for coordination with other EDA tools.

How does one integrate HDL-based IP into BTC designs?

Many options exist, depending on security, boundary integrity, size, etc. The simplest approach is to provide a netlist of the IP to the BTC software and let the software place it within soft boundaries, with little to no differentiation from non-IP functionality.

What is the timing closure process?

The BTC software/hardware architectural strategy is to make timing closure easy for the designer. In general, the process for the CoMesh chip is to submit the functional design, set the preferred delay parameter for the design, and evaluate the modeled performance of the generated circuit. Should some performance measures be unsatisfactory, those specific performance parameters would be reset and the design resubmitted for generation.

Within a guaranteed lower performance limit (2.4 ns per block of five levels of logic, plus a maximum of 1.1 ns cross-chip busing delay for about 180K gates at 100% utilization), CoMesh timing closure is *independent* of logic and layout. Multiple block delays can be reduced by flattening the design, automatically trading area for time. We have not yet designed timing software for non-CoMesh architectures.

For designers not wishing to engage in timing specification or design, a functional non-timed specification can be submitted to the software, and the software will time, place and route the specification, reporting the number of block delays in the design, or when a development board is attached, reporting the timing performance of the physical circuit. The designer can then elect to lower the number of block delays by using more blocks, should they be available.

How does the software support multiple clocks?

As our target technologies become more defined, we intend to create the appropriate tools and extensions support multiple clocks as desired. and see no difficulties in doing so. We envisage supporting at least four clock domains for the CoMesh hardware.

What does "software/hardware co-design strategy" mean?

Commonly, the performance of synthesis and layout software is degraded by fixed and limited hardware resources. Co-designing hardware that optimizes the performance of synthesis and layout software, while co-designing software algorithms that take into account specialized hardware resources, removes or minimizes traditional problems in reconfigurable architectures such as routing congestion and wasted logic capabilities.

How scalable is the synthesis tool to larger devices?

Since all algorithms are both simpler than existing algorithms, and polynomial in complexity, scalability is not a problem. Our current implementation will require a rewrite to reach maximal efficiency for very large devices, since much of the slower cross-verification code can be eliminated.

Hardware

Define what you mean by deterministic timing. What are the limitations and exceptions?

Any block of 300 user gates of standard combinational or sequential logic with five or fewer levels of logic, and with up to 16 inputs and 32 outputs, can be processed in 2.4 ns or less, including clocking of registers. Cross-chip communication between blocks takes less than 1.1 ns for a 180K gate chip, worst case. We anticipate no exceptions to this performance.
Thus, deterministic timing is a guarantee that block to block processing, including block registers, will incur a delay of 3.5 ns, worst case. We do not yet have timing models for i/o pads.

Explain contents of the cell.

We have a number cell designs that are variations on a basic theme. Our current SPICE model is based upon the following cell structure: SRAM-based selectable polarity for inputs, a small reconfigurable logic network that expresses all three-input functions using one or two cells, steerable output to cell neighbors and across multiple cell layers, and register feed-in to any cell. Cells are composed of standard transistor logic elements.

Is the cell 3-4 gate metric the number of gates in the cell or the expected number of routed gates?

The 3-4 gates per cell metric refers to the expected number of routed gates. A cell requires about 300 square microns of silicon area in a .18 micron geometry. A block of 80 cells is the primary logic element, handling around 300 gates at 100% utilization. Average utilization is expected to be over 70%. Blocks can mix different logic and FSM functions.

Explain how gate density numbers were arrived at in the comparisons. Does it include I/O and pad ring for instance?

All BTC gates are available user logic gates at 100% utilization. BTC uses no "system gate" metrics. Competitors' gates are logic gates as claimed in specification sheets, with exceedingly high expected logic densities. For example, we are quite skeptical of Xilinx's claimed 100% utilization expectation of 12 logic gates per 4LUT. Experienced FPGA designers expect fewer than 6 gates per 4LUT for hand-crafted designs, based on 100% utilization of multiple LUTs.

How does CoMesh support reconfigurable processing? Is there special H/W support for reconfigurable processing? How does software support reconfigurable processing?

CoMesh blocks are reconfigurable to any standard combinational or sequential functionality. The software automatically generates configuration files. Currently JTAG standards are assumed for reconfiguration. The SRAMs that support reconfiguration are both readable and writable. Within-block routing is supported by SRAM switches at statistically optimized locations. Global routing is supported by conventional switching matrices. Reconfiguration can be achieved in less than a microsecond, independently in any block, and on-the-fly during dynamic processing, should we elect to include circuitry to provide for this capability. Our chip density calculations do not include specialized circuitry for non-standard reconfiguration options.

The block architecture has a flexible design, blocks can be fabricated in wider and in deeper configurations. Wider and deeper blocks are slower; thinner and shallower blocks are faster. We expect that mixing different structural block types on the same chip will accommodate a wider diversity of user logic more efficiently.

Define what you mean by "scalable". What are the limitations and exceptions?

Performance of the software scales polynomially with circuit size. At all scales, we anticipate at least 70% logic utilization using automated software partitioning, placement, and routing. The software can partition large designs with exceptional efficiency, anticipating and configuring bus utilization to avoid routing congestion. Due to the very large logic block size, and the hierarchical routing scheme, we expect software place and route capabilities to maintain 70% utilization without incurring timing degradation due to routing inefficiencies.

According to our SPICE models, all hardware blocks are expected to process 300 logic gates (100% utilization, .18 geometry) in 2.2 ns, plus .2 ns for register output. Communication between blocks in a block-neighborhood is expected to be a maximum of about .4 ns. Thus, each block-neighborhood can evaluate approximately 5000 gates of logic (100% utilization) in 2.8 ns when all blocks are processing in parallel. This performance is consistent for all block-neighborhoods on a chip.

Our design permits up to 20 block-neighborhoods (100K gates at 100% utilization) to communicate over the hierarchical local bus within a delay of .7 ns. Thus the design scales to 100K gates with a block-to-block processing time of 3.1 ns.

Using the global bus, our design permits up to 36 block-neighborhoods (180K gates at 100% utilization) to communicate within a delay of 1.1 ns. Thus the

design scales to 180K gates with a block-to-block processing time of 3.5 ns. For larger gate counts, the block-to-block communication delay over the global bus degrades gracefully, with a cross-chip block-to-block processing delay of approximately 4.5 ns for 500K gates (100% utilization, .18 geometry).

The architecture is designed to scale down to below .1 microns while incorporating the proportional improvements of smaller geometries. We expect to handle all conventional functionalities with the automated partitioning software without encountering routing congestion. We emphasize that the large logic blocks will rarely require more than local routing to accommodate very large designs.

What does multilevel logic mean? Does it mean each logic cell holds multiple values or that there is hierarchical routing?

Multilevel logic refers to logic resources in series. ASICs use multilevel logic to combat the exponential explosion of routing required by two-level logic.

No, not multivalued logic or hierarchical routing. There *is* hierarchical routing in the BTC design, but not at the block level. One five-level CoMesh block is analogous to five 4LUTs in a logic path. The block width is analogous to about five 4LUTs, so a CoMesh block is functionally analogous to a square array of about 25 4LUTs, although architecturally there is nothing in common between CoMesh blocks and 4LUTs.

One 4LUT can reasonably be expected to accommodate about 6 logic gates, thus a square array of 25 4LUTs might accommodate around 150 logic gates at 100% utilization. Assuming the utilization ratios of CoMesh blocks and 4LUT arrays are equivalent, the CoMesh logic capacity of 300 gates/block doubles the logic density of 4LUT architectures. Since CoMesh cells are more fine grain that 4LUTs, and since CoMesh blocks can easily mix logic functionalities, the utilization of CoMesh blocks is expected to be significantly higher than an analogous 4LUT array.

How does the routing interact between cell, block, and block neighborhoods?

Cell routing is dedicated to the block a cell is in. Block routing is optimized for fast communication within the block-neighborhood a block is in. Block to block routing speeds are hierarchically enhanced for close block-neighborhoods, within about 3 mm. The worst case 1.1 ns block-to-block global routing delay is cross-chip, using the global routing facilities.
The software configures all routing interaction at the functional level. The cell routing architecture within blocks is statistically optimized to the software partitioning capabilities and to the expected average types of functional usage. The block routing architecture within block-neighborhoods

is similarly optimized to the software partitioning capabilities. The block-neighborhood routing is designed for efficient use of the global bus.

Does the routing delay increase in a step function when moving from block to block and neighborhood to neighborhood?

Blocks within neighborhoods have dedicated local routing; at this time we intend to provide up to 16 inputs per block, and up to 32 outputs. We can provide up to 64 input wires into block-neighborhoods, and up to 128 output wires, more if statistically justified. These dedicated local routing resources do not incur the switch box delays of conventional reconfigurable architectures. Instead, the design statistically distributes SRAM routing switches between blocks in order to optimize the software placement and routing capabilities.

Neighborhood to neighborhood routing will be supported by both local and global hierarchical busses. Due to the size of the neighborhoods and the partitioning capabilities of the software, we anticipate bussing groups of vectorized signals into and out of larger functional blocks with high efficiency, and with few switching matrix delays.

Abundant routing implies larger die size. Is this true? What are the tradeoffs?

Routing congestion can be avoided by exceptionally efficient partitioning, placement and routing algorithms processing exceptionally condensed multilevel logic data structures for exceptionally efficient hierarchical routing resources. We believe our co-design strategy can assure abundant routing while using less silicon area for routing than conventional reconfigurable hardware. The most sensitive trade-offs are defined by the statistical analysis of average user configurations in order to maximize the potential hardware design efficiency across a broad possible customer base.

The 300MHz performance number is related to what chip density? How would such performance degrade with density?

300 MHz refers to the minimum time a signal takes to enter a CoMesh block, traverse five levels of logic, settle into a register, and then transfer to any other block on the CoMesh chip consisting of up to approximately 36 block-neighborhoods (about 180K gates at 100% utilization), based on a .18 micron process geometry. Performance degradation is described about under scalability.

Why is the CoMesh architecture excellent for embedded reconfigurable processing? Which part of the architecture makes it more suitable for embedded applications?

The primary advantages of the CoMesh architecture are in the block and block-neighborhood designs, as co-designed with the software capabilities. These fully reconfigurable logic cores can be embedded on chips with other silicon features without effecting their performance. The block design can be highly customizable for prespecified applications. For example, speed can be increased by using fewer logic levels or narrower blocks, and density can be increased by using deeper but slower blocks. As well, the block architecture has been designed to accommodate embedded shift registers, FIFOs, arithmetic functions, counters, and the like without negative impact on the performance of the reconfigurable block.

Technology

Please explain how boundary logic works and how the technology helps achieve better logic and die area optimization?

This question is probably best answered in person. Boundary logic can be thought of as a mathematical system that is "underneath" Boolean logic. It has a simple mapping to conventional logic, but also has simple and powerful transformations that are not available to conventional logic. Boundary logic is an algebraic system with one operator and one ground value. The operator is variary, taking any number of arguments, and spatially disjoint in that associativity and commutativity are not relevant concepts. Boundary logic is a rigorous formal system, so it "works" from an axiomatic basis. A formal system that is structurally more powerful and more compact than logic yet can be interpreted as logic confers optimization benefits upon all areas of logic, including the minimization of logical expressions and their embodiment in hardware.

In particular, the boundary logic formalism combines logic functionality and wiring into a single concept. This makes partitioning for logic and routing resources particularly easy. Since the formalism is algebraic, simple match-and-substitute algorithms are all that are needed for circuit transformation. Match-and-substitute algorithms are well understood, and all are of polynomial complexity, so that the implementation is both efficient and scalable. Because there is only one operator, parsing and keeping track of different types of gates is not necessary. Having no grouping or ordering concepts makes moving wires and changing fan-in and fan-out relations particularly easy. Since every step is formal, the functionality of the circuit is never changed by transformations, and equivalence checking is not needed. Since every formal step is characterized by a specific structural trade-off, each transformation achieves a specific known design objective. Since the formalism is compact, there are only five transformations in total, each one sharply delineating the effect of moving around the design space of circuits

with fixed functionality. Because there is always a simple map between boundary logic and Boolean logic, it is always easy to move between the abstract data structure and an actual circuit netlist. And perhaps most importantly, boundary logic maps one-to-many into conventional logic, making it *always* smaller and simpler than any existing conventional EDA approach. In technology mapping from boundary logic to a target architecture, for example, a single boundary logic data structure represents a family of equivalent circuit configurations; choosing between them is simply a matter of reading the boundary logic structure using a different interpretation.

In sum, boundary logic tools do the same as any other EDA tools, using less structure, less mechanism, and more powerful and succinct transformations in all cases.