DISCUSSION OF PUN-ENCODED CM85A CIRCUITS

William Bricken
November 2001

This memo includes various topics of description and discussion for the 34 varieties of cm85a. Several updated varieties are not included since they were added after these comments were written. Topics include:

STRUCTURAL VARIETIES

Optimization

- SCHEMATIC 1: Two-level Logic (PLD)
- SCHEMATIC 2: Multilevel Benchmark Circuit (cm85a)
- SCHEMATIC 3: Suppress Inverters
- SCHEMATIC 4: Remove Redundancy
- SCHEMATIC 5: Reduce Reconvergence
- SCHEMATIC 6a: Increase Fanin (poor choice)
- SCHEMATIC 6b: Increase Fanin (good choice)
- SCHEMATIC 7a: Enhance Testability (poor choice)
- SCHEMATIC 7b: Enhance Testability (good choice)

Design Constraints

- SCHEMATIC 8: Reduce Critical Path (6 gates)
- SCHEMATIC 9: Pipeline (3 two-input gates)
- SCHEMATIC 10a: Map to Specific Library (poor choice)
- SCHEMATIC 10b: Map to Specific Library (good choice)
- SCHEMATIC 11: Three-level Logic
- SCHEMATIC 12: Map to NAND Gates
- SCHEMATIC 13: Map to FPGA (4-LUTs)
- SCHEMATIC 14: Binary Decision Diagram

Hierarchical Abstraction

- SCHEMATIC 15: Abstract Low-level Components
- SCHEMATIC 16: Abstract for Component Connectivity
- SCHEMATIC 17: Abstract for Sequential Structure
- SCHEMATIC 18: Abstract for Parallel Structure
- SCHEMATIC 19: Abstract for Output Structure
- SCHEMATIC 20a: Abstract Bit-width (recursive)
- SCHEMATIC 20b: Abstract Bit-width (enables)
- SCHEMATIC 20c: Abstract Bit-width (enables, recursive)

BTC Hardware Models

- SCHEMATIC 21: Distinction Network I
- SCHEMATIC 22: Distinction Network II
- SCHEMATIC 23a: Occlusion Array (Dnet 21)
- SCHEMATIC 23b: Occlusion Array (Dnet 22)
- SCHEMATIC 23c: Occlusion Array (two-level Dnet)
- SCHEMATIC 23d: Occlusion Array (raw multilevel benchmark)
- SCHEMATIC 24a: Comesh (multilevel)
- SCHEMATIC 24b: Comesh (two-level)
- SCHEMATIC 25: Bit-stream Simulator

DESCRIPTION OF THE SCHEMATICS (BULLETS)

Optimization

- SCHEMATIC 1: Two-level Logic
  optimized product-of-sums form
  two-level logic (c. 1970)
  three gate types (AND2, OR2, NOT)
  no visual information, hard to read
  a look-up table model
  exponential in number of inputs

- SCHEMATIC 2: Multilevel Benchmark Circuit
  not optimized
  a literal transcription of the benchmark netlist
  multi-level logic network (c 1980)
  three gate types (AND2, OR2, NOT)
  hard to read

- SCHEMATIC 3: Suppress Inverters
  collapse inverters into adjacent logic gates
  additional gate types (AND2, OR2, NOT + NOR2, NAND2)
  hard to read, some structure is visible
  complexity hidden in new gates does not help

- SCHEMATIC 4: Remove Redundancy
  minimize logical structure with Losp
  add new gate type (AND2, OR2, NOT, NOR2 + OR3)
  structure is beginning to appear
  do not identify patterns yet
  size, wiring and performance improvement

- SCHEMATIC 5: Reduce Reconvergence
  reconvergent loops are not testable
  fewer wires, fewer gates
  increased fault-tolerance

- SCHEMATIC 6a: Collect Signals, poor
  condense gates by increasing arity
  add new gate types (AND2, NOT, NOR2, OR3 + OR4, NOR3, NOR4)
  can build 2-input trees after optimization
  gate diversity obscures structure
  finding best design is interactive exploration

- SCHEMATIC 6b: Collect Signals, good
  condense gates by increasing arity
  add new gate types (AND2, NOT, NOR2, OR3 + OR5)
  visual structure is apparent
  easy to identify hierarchical components
  best optimization metrics
  improved transistor network

- SCHEMATIC 7a: Increase Testability
  restructure to homogeneous NOR-gates (NOT, OR2, NOR2, NOR9)
  no reconvergent paths
  most fault tolerant, best for testability
  longer critical path

- SCHEMATIC 7b: Increase Testability, optimize
  restructure to NOR-gates (NOT, OR2, NOR2, NOR3, NOR5, XOR)
  fewest gates
  no reconvergent paths
  most fault tolerant, best for testability
  longer critical path

Design Constraints

- SCHEMATIC 8: Arbitrary Constraint: Depth = 6
  example of critical path reduction
  illustrates CDE parametric constraint
  uses simple gate library

- SCHEMATIC 9: Pipelining Constraint: Delay chunks = 3

- SCHEMATIC 10a: Specific Library, poor
  example of technology mapping
  illustrates CDE parametric constraint
  uses pre-specified gate library (NOT, OR2, NOR2, AND2, XOR2)
  poor choice, still need to explore

- SCHEMATIC 10b: Specific Library, good
  pre-specified library (AND2, NAND2, OR2, NOR2, XOR2)
  no inverters
  clear structure is maintained

- SCHEMATIC 10c [NO LONGER INCLUDED]: Arbitrary Constraint: Mixed
  maximum fan-in = 3
  maximum-fan-out = 2
  wire count = low
  fault-tolerant = yes
  gate library = {AND, OR, NOT, MUX, XOR}
  delay = 7

- SCHEMATIC 11: Three-level Logic
  no maximum fan-in
  no maximum-fan-out
  wire count = high
  gate library = {AND, OR, NOT, XOR}
  delay = 3

- SCHEMATIC 12: Map to NAND Gates
  one homogeneous gate type (NOT, NAND2)
  appears complex, but is simplest at transistor level

- SCHEMATIC 13: Map to FPGA (4-LUTs)
  technology mapping into four-input look-up tables
  illustrates partitioning by different criteria
  clusters of variables rather than logic gates
  partitioning approach doesn't matter to Losp

- SCHEMATIC 14: Map to Binary Decision Diagram
  dominant technology in EDA tools
  one-way tool, circuitry to BDD, not BDD to circuit
  does not indicate structure of circuit
  answers yes/no questions
  does not identify location of faults

Hierarchical Abstraction

- SCHEMATIC 15: Low-level Hierarchical Abstraction
  identify intermediate patterns in circuit
    XOR group depth=1
    OR3-INV group depth=1
    OR5-AND group depth=2
  enhance readability
  identify space/time design trade-offs
  prepare for high-level abstraction
  provides abstraction choices
  wiring tangle due to no input swapping

- SCHEMATIC 16: Emphasize Wiring
  identify patterns that minimize wire tangling
    XOR-OR3 group depth=2
    AND-INV group depth=1
  don't force input swapping
  tangle is in splitting AND-INV output to both OR5 gates

- SCHEMATIC 17: Emphasize Sequential Structure
  identify largest patterns in circuit (alternative 1)
  XOR-OR3-INV group depth=2
  OR5-AND group depth=2
  merge prior abstractions hierarchically
  isolate critical path

- SCHEMATIC 18: Emphasize Parallel Structure
  identify largest patterns in circuit (alternative 2)
  XOR group depth=1
  OR3-INV-OR5-AND group depth=8
  abstracted from fault tolerant configuration
  isolate critical path in a hierarchical component

- SCHEMATIC 19: Emphasize Output Structure
  identify pattern of entire subcircuit
  OUTPUT group depth=9
  only original inputs to OUTPUT group

- SCHEMATIC 20a-b-c: Abstract Bit-width

BTC Hardware Models

- SCHEMATIC 21: Distinction Network I
  homogeneous logic components
  least complex wiring
  propagates disconnects rather than signals
  asynchronous

- SCHEMATIC 22: Distinction Network II
  homogeneous logic components
  long path but less wiring
  propagates disconnects rather than signals
  asynchronous

- SCHEMATIC 23a: Occlusion Array, optimized
  homogeneous hardware substrate, a modified memory
  optimized circuit expressed as a software array
  functionality of circuit is expressed in memory
  clocking and wiring are in the substrate not in the design
  marks in array represent and abstract wires
  nearly constant time performance
  performance largely independent of complexity of circuit
  different arrays correspond to different circuit structures

- SCHEMATIC 23b: Occlusion Array, testable
  homogeneous hardware substrate, a modified memory
  testable circuit expressed as a software array
  hardware testability is not a relevant concept
  identical performance to other arrays

- SCHEMATIC 23c: Occlusion Array, 2-level
  homogeneous hardware substrate, a modified memory
  2-level circuit expressed as a software array
  2-level shifts array marks to edges
  constant performance over all inputs guaranteed
  array empty space can be reclaimed, not wasted

- SCHEMATIC 23d: Occlusion Array, benchmark
  homogeneous hardware substrate, a modified memory
  benchmark circuit expressed as a software array
  poor design and lack of optimization make array larger
  poor design does not effect processing time of array

- SCHEMATIC 24a: Comesh
  multilevel

- SCHEMATIC 24b: Comesh
  two-level

- SCHEMATIC 25: Binary Bit-Stream
  uses conventional Pentium-class processor and memory
  functionality of circuit is expressed in software
  circuit with inputs is a bit-stream
  evaluation of bit-stream is a single-pass algorithm

DISCUSSION OF THE SCHEMATICS (NARRATIVE)

The Losp/Pun transformation strategy is chronicled by thirty-one circuit structures, all with identical functionality. These schematics have been arranged on the page by hand, in order to emphasize visible structure. The layout of all schematics has been equally careful, so that presentation clarity is not a biasing factor. That some versions are in fact clearer and simpler than others illustrates two important points about circuit design:

1) The space of possible structures for a single circuit is huge and difficult to explore. Those structures with desirable behavioral properties (fast, small, testable, etc.) are in quite different locations in the design space. Compare:

S1	-- designed using a look-up table	(fast, non-scalable)
S2	-- designed using a logic network	(slow, scalable)
S6b	-- designed using Losp reduction	(best overall)
S7b	-- designed for testability	(slow, fault-tolerant)
S12	-- designed to minimize CMOS transistors	(best performance)
S13	-- designed for an FPGA architecture	(reconfigurable)

2) Better designs do exist, however they do not necessarily align with the way a designer may conceptualize the functionality of the circuit. Thus, the way we specify the functionality of a circuit does not necessarily lead to optimal designs. Logic synthesis is a necessary step in the design process.

Optimization

> "Compared to two-level logic synthesis, the problem of optimum multilevel logic synthesis is an impossible dream."
> [Hachtel, *Logic Synthesis and Verification Algorithms*, 1996, p. 407]

The first nine schematics are sequential steps in the CDE optimization process.

S1 is the two-level logic representation. This form was used extensively during the early development of EDA tools in the 1970s. It works only for small circuits, increasing exponentially in size as the number of inputs and registers increase. Two-level circuits are essentially look-up tables. Visually we can see that, like large look-up tables, the structure of a two-level circuit provides no clues as to its functionality. Two-level circuits remove design information from hardware design.

Even today, a two-level representation is required for many EDA tools, such as all known exact techniques for Boolean minimization. The patented Stalmark method (sold by Formal Inc), for example, is very efficient at

processing two-level forms that contain literally millions of terms. However, this is a software analysis process, it is not possible to actually build two-level circuits that large. This means that Formal can answer yes/no questions about the integrity of a design, but it cannot identify which components of the design are in error, nor can it propose hardware solutions.

The CDE uses the opposite of two-level circuits, it uses multilevel versions which are as deeply nested as possible. Deep multilevel circuits are considered to be a disadvantage, since they are the slowest. Worse, no tools exist (other then Losp/Pun) for optimizing multilevel circuits. These circuits are also the simplest and easiest to understand, and thus the best place to begin an optimization process. Delay (depth) can be reduced after the redundant and reconvergent aspects of a design have been removed. Two-level approaches confuse optimization with efficiency. More importantly, the design criteria for today's deep sub-micron circuits are substantively different than those for semiconductor technologies of thirty years ago.

S2 is the ISCAS'91 multilevel benchmark circuit (cm85a) used for all schematics in this paper. It is small, so two-level techniques will work for it, and it is limited to a small library of two-input gates (AND, OR, NOT). However, this design is also complex and confused with respect to the actual functionality it represents. The following CDE transformations remove this unnecessary complexity.

S3 simply increases the types of two-input gates (AND, OR, NOT, NAND, NOR), making the design appear less cluttered at the cost of more effort to understand what each gate does. This process does not improve the circuit, since gate representations are all homogenized at the transistor level. It also hides unnecessary structure and available abstraction by placing processing boundaries in undesirable places. These boundaries become accepted by a designer, effectively freezing the grain-size of the design elements. What is gained in visibility is lost in inflexibility.

S4 is the result of the CDE redundancy removal process. Since no other effective multilevel tools are available, this type of result is unique for EDA tools. We can see structure beginning to emerge.

The clutter of gates at the input have become four highly regular modules. This regularity provides a designer with the capability to trade-off circuit speed and area. Here, the four modules are instantiated with an area of 16 gates, in order to achieve maximum performance speed. Alternatively, when space is at a premium (such as the case of a design being slightly larger than the available area on a chip), four sets of signals can be feed

sequentially into one module, reducing the area to four gates while increasing the processing time fourfold.

The diagonal structure of four 3-input OR gates is also suggestive. This emerging structure was achieved by permitting a greater fanin, here three input OR gates rather than only two-input gates. Gates with higher input are inherently slower, and thus are avoided in some designs. Here again is an example of confusing functional design with efficiency. In effect, multi-input gates are a design expedient which can be removed without cost when the design is completed.

The four clusters of OR/AND gates at the bottom of the schematic provide another example of emergent structure. These clusters embody a horrible design characteristic, they have reconvergent signals which split to enter two OR gates, and then recombine in the AND gate. In this schematic, the reconvergence is simple and clear; however, the same problem existed in **S2**, where it was not as visible. This is an example of redundancy removal exposing poor design.

S5 is the result of the CDE reconvergence removal process. Ordinarily, redundancy and reconvergence (and other undesirable structures) are removed all at once during CDE optimization. Here, we have interrupted the optimization process in order to show desirable intermediate results.

Again we see more 3-input OR gates and a simpler suggestive structure. Further releasing the multiple input constraint will again clarify the circuit structure.

S6a is intended to be the final result of minimization. However, it contains several design flaws. In particular, the 4-input OR gates are still too constrained in the number of inputs. Second, and worse, the circuit has been configured using a diversity of gate types. This effectively destroys the visual and functional symmetries of the circuit.

S6b is the final result of CDE optimization. It provides the best overall functional design metrics. The two 5-input OR gates provide structural organization. The wires entering the eight AND gates are not tangled in a maze (compare **S17**). We can see the gate level circuit structure clearly. There are essentially three types of modules, the four at input, the four 3-input OR gates followed by inverters, and the two 5-input OR-AND clusters. With effort, it is now possible to reverse-engineer the circuit, to identify the functionality solely from the structure. **S17** abstracts these components into hierarchical library modules. **S21** converts this circuit into a distinction network.

S7a is a design generated by the CDE which emphasizes testability. A testable circuit has no reconvergent paths. The cost is a longer path delay. The design is achieved by reducing the circuit to a distinction network, then manipulating it to contain fewest wires.

This example illustrates the substantial differences between designs with different design objectives. The CDE can generate designs which meet pre-specified criteria, moving freely around the design space in order to identify structures which achieve desirable performance characteristics.

S7b is the fault tolerant design optimized for gate count. This circuit has the fewest gates all.

Design Constraints

> "Given a set of transformations, it is difficult, if not impossible, to claim that all equivalent network configurations can be reached by some sequence of transformations....Different sequences of transformations lead to different results that may correspond to different local points of optimality."
> [DeMicheli, *Synthesis and Optimization of Digital Circuits*, p.356]

The next eight schematics illustrate a diversity of design constraints asserted as parameters in the CDE. All constraints are applied to the reduced circuit (**S6b** and **S7b**). The CDE generates functionally equivalent designs based on values of pre-specified parameters. At the moment these parameters include:

- signal propagation delay
  the length of longest path through the circuit
- silicon area
  count of the number of logic gates in the design
- wiring
  count of the number of nets connecting circuit gates
- generic quality of design
  a composite metric,
  the sum of gates and wires, multiplied by the delay
- fanin and fanout
  limits on the number of inputs and outputs for a logic gate

With slight modifications, the following metrics can be added to the CDE parametric space:

- power consumption
  count of the number of gates transitioning for an exhaustive set of input vectors
- transistor area
  count of transistors specific to each gate type, modified by the transistor width required to maintain signal strength
- noise
  count of the number of concurrent gate transitions
- pipelining
  subdivision of the circuit into equal delay segments
- N-input look-up tables
  partitioning into tables with limited inputs

The EDA industry has built elaborate models of physical circuitry in order to better estimate design performance prior to the costly fabrication process. The CDE does not include these models, however, the minimization architecture is easily modified to accommodate different semiconductor models. The CDE has the capability for highly accurate modeling, but it does not have the actual models, which tend to be trade secrets.

S8 illustrates design changes when the CDE delay parameter is fixed. One complexity is that different types of gates have different delays, so that an accurate estimate of delay must take into account the transistor network rather than the bulkier logic gate model. Suppose for compatibility with other board components, the circuit delay must be 6 gates.

One approach to reducing delay would be to begin with **S6b**, which has a delay of 8. Changing the 3-input OR along the critical path is, however, a bulky solution. As well, **S6b** uses several different types of gates, so that the accuracy of a gate count model is suspect.

In contrast, the fault-tolerant structure **S7b** has a maximal path length (propagation delay) of 11 gates. Beginning with the more homogeneous structure may imply a greater number of changes, however each change is modular and simple to implement. As well, the 2-input NOR gates are easy to modify, and maintaining the testability of the resultant circuit would be desirable.

The CDE includes incremental transformations which allow specific trade-offs between circuit complexity (area) and delay. These incremental changes can be applied and evaluated one at a time. This process is probably best seen as an interaction between the CDE and a designer. The designer would first ask for the current critical path to be identified, then s/he would identify a particular part of that path to reduce, and this process would iterate

until the desired structure is generated. Alternatively, a simple rule-based automated implementation may be sufficient for most design problems.

When the critical path is expressed as a nested parens form, delay is simply the maximum depth of the nesting. As in S1, application of distribution in the distribute direction will successively reduce the length of the path (i.e. the depth of nesting).

S9 illustrates pipelining. Here, the objective is to identify modular delay components of the same length, so that signals can be processed in waves. This constraint is simply a multiple application of delay shortening. Specifying the maximum delay for the entire circuit becomes specifying the maximum delay for components along the critical path.

With the pun form, pipelining is easy to achieve. Cells are expanded until a pre-specified depth is reached. When all cells have the same depth of nesting, pipelining partitions have been identified.

S10a is a different type of design constraint, that of phrasing the circuit structure using a pre-specified set of gate types. Specific gate libraries may be an enforced limitation of the design environment, they may be associated with limited fanin and fanout in order to manage power consumption, they may be a preference of the designer in order to better understand the circuit, or they may be associated with other performance characteristics of the target technology. This example is one of technology mapping as well as design constraint.

As with all circuit transformations, the number and variety of possible structures are huge. The cell library in this circuit is {NOT, OR2, AND2, NOR2, NAND2, XOR}. Although appearing to be reasonable, this selection permits too much gate diversity, in effect undermining the symmetric structures in the circuit.

S10b uses a slightly different library, {OR2, AND2, NOR2, NAND2, XOR}. Simply by suppressing the inverter gates, the symmetric structure of the circuit is maintained. The selection of library components is an interactive exploration.

S10c [NO LONGER INCLUDED IN COLLECTION] provides a complex example of parametric design. Here, a circuit structure is requested by setting the CDE parameters to the following values:

- maximum fan-in = 3
- maximum-fan-out = 2
- wire count = low
- fault-tolerant = yes
- gate library = {AND, OR, NOT, MUX, XOR}
- delay = 7
- design quality = not important

The schematic shows that the CDE parameters are all interactive and can be set concurrently. This does not imply that there exists a circuit configuration which meets these constraints. For example, if a delay of 2 is requested, then a two-level circuit similar to **S1** is generated. There is no flexibility in this structure for other considerations.

The CDE interface provides ranges which establish the relative importance of a particular constraint. Some parameters, such as delay, do not require a range, since maximal path length is always best for achieving other design constraints. Some parameters, such as wire count, are only roughly quantifiable. Some, like gate type, fanin and fanout, are strict limitations. The interactive values of most interest to a designer are not at this time known.

In general, a set of constraints falls into one of three categories:

- significantly underspecified
- significantly overspecified
- tightly specified

The meaning of the types of specification (under, over, and tight) depend on the specific circuit, since some functional designs permit a wide diversity of structural solutions, and some, such as a systolic array multiplier, do not. Within the space of possible structures for a specific functionality, significantly underspecified constraints mean that there are many potential solutions. In this case, the CDE can easily identify one. The approach of finding one possible solution when there are many is called *satisficing*, the approach of finding one exact single solution is called *optimization*. The CDE is a satisficing system, since exact optimization has horrendous computational complexity.

Significantly overspecified constraints are also computationally easy, it is easy to determine that there are no possible solutions. In the case of a tight specification, logic synthesis systems have a choice between very long processing times, or rapid achieved close but not optimal solutions. The CDE is of the second type.

S11 is a mapping to 3-level logic.

S12 provides a solution to the constraint that all gates be two-input NAND gates or INVERTERS. This particular constraint is reasonable since the transistor count and the computational effort for NAND gates is minimal. Although this circuit has more gates than others, when NAND2 gates are converted into a transistor network, they are optimal. thus this circuit is best from a hardware performance perspective.

S13 provides a mapping to a radically different type of architecture, that of FPGA reconfigurable look-up tables. Instead of logic gates, the CDE has generated a circuit decomposition composed of tables which are constrained to take up to four inputs. FPGA architectures vary widely in types of reconfigurable components. Four-input look-up tables is a common simple case.

Designers have learned to think in terms of logic gate networks. The design qualities of a circuit are hidden when look-up tables are used, the two-level design **S1** being an extreme example of one look-up table for the entire circuit. (In fact, if the interior of the four-input look-up tables were exposed, they would look like smaller versions of **S1**.) Thus, there is no structural information in this schematic, a design trade-off in order to achieve reconfigurability.

Partitioning by number of different variables is again easy for Pun. The fully expanded circuit can be chunked from the deepest space outward simply by counting the number of variables. When four (or N) is reached, a new cell is constructed and labeled. The partitioning process then iterates with the new 4-input cell locked.

S14 is not a circuit, but a binary decision diagram (BDD). Almost all current EDA tools use the BDD as a primary data-structure. This example visually illustrates the primary difficulty with BDDs: they provide yes/no answers to functional questions about a circuit (such as: Are two structures functionally identical?) but they do not provide information about the structure of a circuit. BDDs are one-way streets, you can turn a circuit design into a BDD, but you cannot turn a BDD into a circuit design. The BDD technology is pervasive simply because better methods are not known. All problems in circuit minimization are exponentially complex, which means that they do not scale as the circuit becomes larger. Thus, in an era of million gate structures, current EDA tools have reached a breaking point. The generally espoused solution is formal verification, using theorem proving techniques to address families of circuit structures.

The CDE uses a representation which maintains a close fidelity to logic gates, is scalable, and as well provides formal verification. It is the first known fully functional tool for multilevel circuit optimization.

Hierarchical Abstraction

> "Library binding is the key link between logic synthesis and the physical design of semicustom circuits."
> [DeMicheli, *Synthesis and Optimization of Digital Circuits*, p.546]

The next eight schematics are examples of hierarchical abstraction. Abstraction is mandatory as designs grow large, since human designers find it exceedingly difficult to extract information from logic networks containing tens of thousands (or more) of gates. To accommodate this restriction, design itself has embodied modular principles. A primary example of this is bit-width: an 8-bit adder differs little from a 16 or 32-bit adder, except for duplication of components. Thus modular abstraction already exists as a top-down design practice.

What current tools do not provide is bottom-up abstraction, the ability to identify modular patterns in a circuit which have not necessarily been put there during design. The primary advantage of bottom-up abstraction is that designers can identify available trade-offs between circuit area and circuit delay.

Another (currently unavailable) advantage of bottom-up abstraction is that modular components can be partitioned parametrically. Circuits can be sliced up in different ways, some of which may be more desirable for a particular design context. Not only does the CDE identify available modular components, it can move across different modular groupings, customizing the level and degree of abstraction.

S15 abstracts the low-level components which can be seen by visual inspection of **S6b**. The CDE builds customized library elements which have the functionality of each component, and can be reused as modular components. Here, all gates have been encapsulated in hierarchical modules.

S16 abstracts different components in order to avoid the tangle of wires generated by **S17**. This tangle is an artifact of abstracting the two OR-AND groups. Each group has nine inputs. In order to maintain structural similarity within the OR-AND groups, inputs must be similarly sequenced for each. Often designers permit input rearrangement. This removes the tangle of wires, but makes the two modules into different subcircuits. Thus the abstraction is lost. The CDE simply partitions the circuit elements to reduce the ordering of inputs. Here, each module has only three inputs.

S17 illustrates an abstract view of the circuit which highlights the critical path and suppresses other detail. A designer can consider these intermediate modules to be encapsulated mini-circuits composed solely of logic gates. Alternatively the intermediate components can be treated as hierarchical abstractions composed of circuit elements and other lower-level abstractions.

S18 illustrates an abstract view which highlights the parallel portion of the circuit. Sequential paths are encapsulated inside modular components, exposing that portion of the circuit which propagates signals in parallel. The ability to partition sequential and parallel components of a logic structure is unique to the CDE.

S19 continues to emphasize parallel components, but it forms modules which encapsulate an entire output tree. Thus, the inputs to the InputChain modules are all primary inputs. What is then isolated is the third output.

S20a-b-c illustrates a common abstraction across bit-width. This provides the capability of designing generic components which can be generated to fit any width of parallel binary inputs. An N-bit adder, for example, can be instantiated in one part of a circuit which uses only 4-bit data as a 4-bit adder, while being instantiated as a 16-bit adder in other parts which use 16-bit information. Since the CDE can generate parameterized components, designs can be customized for particular uses, and even for particular input configurations.

An example of bit-width configuration combined with other CDE parameters is a decimal-to-binary converter. In order to accommodate decimals with 10 input varieties, a binary system needs 4 bits which can be configured into 16 (2^4) varieties. However, 6 of these binary configurations (which would represent the decimal numbers 10 through 15) are never used. Thus the input to a binary-to-decimal converter is biased. The lowest bits occur probabilistically 50% as 0 and 50% as 1. The highest bit, which covers the decimals numbers 8 through 15 when set to 1, is most often a 0 (specifically four times out of five a decimal number will be 0 through 7, and one time out of five it will be 8 or 9). Thus the circuit can be optimized to take this biasing into account. Biased input is quite common in many applications, however current design tools enforce a 50/50 expectation for input values.

BTC Hardware Models

The remaining nine schematics show BTC hardware designs. These designs are provided for contrast. They are not part of the CDE functionality, rather they illustrate radically different hardware architectures which embody the software innovations within the CDE. This approach is of obvious advantage:

rather than using the CDE to improve existing circuit designs, a designer can configure existing designs for hardware which performs the circuit functionality at hardware processing speeds while including all of the design advantages of the CDE's unique approach. BTC hardware can be contrasted with other architectures generated by the CDE tools:

- **S1** -- two-level look-up table architecture
- **S6** -- optimized multilevel logic network (ASIC) architecture
- **S7b** -- fault-tolerant architectural variation of ASICs
- **S13** -- FPGA reconfigurable architecture

S21 is the distinction network version of **S6b**. The unique circular "gates" are distinction nodes. They do not act like normal gates in that they do not propagate signals. Internally each dnode processes by disconnection.

S22 is a variant distinction network structure which mimics the fault-tolerant architecture of **S7** (although the concept of fault-tolerance which applies to ASICs is quite different than distinction network fault-tolerance).

S23a is an Occlusion Array, BTC's FPGA-like reconfigurable architecture. The circuit expressed within the array is **S6b**.
S23b is the Occlusion Array for **S7a**.
S23c is the Occlusion Array for **S1**.
S23d is the Occlusion Array for **S2**.

Although the array configuration is recognizably different for each of these four circuits, the functioning of the array is the same. Occlusion arrays process in almost constant time, regardless of array configuration. thus the design differences over these four circuits are irrelevant to the efficiency of the array operation. Bad design uses a larger array area (**S23d**), but does not effect either the timing or the wiring complexity, since Occlusion Arrays abstract both timing and wiring into spatial configurations of marked memory locations.

S24a-b are CoMesh architecture mappings; **S24a** to **S1**, and **S24b** to **S7**.

S25 is a Bit-stream software architecture which represents the functionality of a circuit as a software bit-stream. This approach simulates the circuit functionality in software, using conventional Pentium-class processors. Circuit elements are bond to input values and then transcribed into a bit-stream. The bit-stream is evaluated in a one-pass operation.

METRICS FOR THE STRUCTURAL VARIETIES

Six simple metrics were used to calibrate the effect of each transformation:

- Devices: the number of gates and i/o ports
- Wires: the number of wires, both internal and external
- Pins: the number of pins, in and out, connected to devices
- Area: the sum of device and pin count
- Delay: the length of the longest path through the circuit
- AxT: a compound measure, the product of the area by delay.

The Devices metric gives a rough indication of circuit area. In sub-micron technologies, wire count is a more accurate measure. Area, the sum of gates and pins, is a slightly more robust measure.

Delay is a critical metric independent of circuit complexity. It measures how many clock ticks are required to evaluate an input. A lower delay means fast circuit. Pipelining obviates this measure.

The compound measure AxT is preferred by experienced engineers, since it balances design trade-offs between speed and area.

The transformations and their accompanying statistics are charted below.

The amount of removable redundancy in the original circuit specification is rather typical of quickly designed circuitry. Even circuits of this size are a challenge for CAD tools, primarily because they uniformly lack Boolean transformation algorithms. Current commercial systems have weaker tools which reduce simple special cases of redundancy, but cannot handle the sheer diversity of logic network structures for the same functionality.

4-LUT and abstraction transformations, as well as alternative architectures such as the occlusion array (S13-24, excepting S21 and S22), are not comparable on these metrics.

	circuit	parts	wires	pins	area	delay	A*T	gate2
1	two-level logic	67	69	294	361	2	722	174
2	multilevel benchmark	90	87	210	300	8	2400	44
3	suppress inverters	69	66	168	237	8	1896	44
4	remove redundancy	59	56	140	199	7	1393	36
5	reduce reconvergence	51	48	120	171	6	1026	32
6a	increase fanin, poor	49	47	116	165	6	990	35
6b	increase fanin, good	49	46	116	165	6	990	32
7a	increase testability	50	48	114	164	9	1476	32
7b	testable, optimized	50	48	118	168	9	1512	32
8	delay, depth=6				0		0	0
9	pipeline, delay=3				0		0	0
10a	specific library, poor	53	50	124	177	10	1770	32
10b	specific library, good	46	43	110	156	8	1248	32
11	three-level logic				0		0	0
12	nand gates	78	76	174	252	9	2268	32
13	4luts	25	29	62	87	4	348	0
14	bdd				0		0	0
15	low-level abstraction	24	34	74	98	5	490	0
16	emphasize wiring	26	35	74	100	6	600	0
17	emphasize sequential	20	26	58	78	5	390	0
18	emphasize parallel	22	24	62	84	2	168	0
19	emphasize output	34	31	88	122	3	366	0
20	bit-width abstraction				0		0	0
21	dnet	50	46	126	176	11	1936	40
22	dnet, less wiring	51	48	122	173	11	1903	34
23a	occ array, optimized				0		0	0
23b	occ array, testable				0		0	0
23c	occ array, two-level				0		0	0
23d	occ array, benchmark				0		0	0
25	bit-stream simulator				0		0	0

Comments below are constrained to Schematics S1-12, S21-22. For convenience, the table is repeated with relevant items only:

	circuit	parts	wires	pins	area	delay	A*T	gate2
1	two-level logic	67	69	294	361	2	722	174
2	multilevel benchmark	90	87	210	300	8	2400	44
3	suppress inverters	69	66	168	237	8	1896	44
4	remove redundancy	59	56	140	199	7	1393	36
5	reduce reconvergence	51	48	120	171	6	1026	32
6a	increase fanin, poor	49	47	116	165	6	990	35
6b	increase fanin, good	49	46	116	165	6	990	32
7	increase testability	50	48	114	164	9	1476	32
10a	specific lib, poor	53	50	124	177	10	1770	32
10b	specific lib, good	46	43	110	156	8	1248	32
12	nand gates	78	76	174	252	9	2268	32
21	dnet	50	46	126	176	6	1936	40
22	dnet, less wiring	51	48	122	173	9	1903	34

Discussion

All metrics are intended as rough approximations which allow comparison across the different versions. They are not intended to be accurate physical models.

Parts: Includes 14 i/o pins and counts inverters.

Gates2: A better indicator of logic complexity, this counts only logic gates, with all gates having only two inputs. S10a and S10b use a more complex XOR gate, so they should have 4 added to their gate2 count for comparison.

Wires: The number of nets does not reflect the wiring complexity since it does not count extra for fanout.

Pins: A better wiring metric than wires. The quantity (pins - wires) is an indicator of the fanout in the design. Unused pins in S13 were not counted.

Area: Devices plus pins estimate the silicon area of the design.

Delay: The longest topological path in optimized designs is also the critical path. In poorly optimized designs, false paths, those that never contribute to the output, may make the actual critical path shorter than the longest topological path. Counting gates to measure delay is very crude. The many factors which contribute to delay (type of gate, fanin, fanout, driving voltages) require a more complex propagation model. As well, the

transistor network itself most be modeled. Inverters were not included in delay counts.

Area * Delay: This is an ad hoc composite measure which balances relative importance of delay and topological complexity.

Ranges

	Range	Best	Worst
Devices:	46 - 90	S10b	S2
Wires:	43 - 87	S10b	S2
Pins:	110 - 294	S10b	S1
Area:	156 - 361	S10b	S1
Delay:	2 - 10	S1	S10a
A*T:	722 -2400	S1	S2
Gates2:	32 - 174	many	S1

many=(5,6b,7,10a,10b,12)

CONSTRUCTIVE ANALYSIS OF CM85A

S1, two-level logic, is obviously in an unreadable form. Similarly, **S2**, the raw multilevel benchmark, poses difficulties for analysis, even with its small size and underlying structure. We can see that the signals cluster in four groups of two {<a,b>, <c,d>, <e,f>, <g,h>}. Inputs {i, j, k} are less interpretable.

S6b and **S7a** both provide a clearer picture of the functionality of this circuit.

Examining **S7a**: CM85A is a four-bit processor, with three auxiliary inputs. Signals {i,k} go directly to an output OR gate. Thus they will dominate the results when hi, and have no influence when low. This indicates that they permit ganging of the functionality, perhaps with other identical four-input processors. Signal j serves the same purpose for the third output. Since j is inverted prior to the NOR9, when j enters low, it determines the value of out1 as low. As well, j effects out2 and out3. When j is low, the last gates of the NOR-chain will be low, leaving the results of out2 and out3 to depend only on i and k.

Thus we have the following table:

Signal	*Effect*
i=low	none
=hi	out2=hi
k=low	none
=hi	out3=hi
j=low	out1=low, out2=i, out3=k
=hi	none

The four groups of two signals are all symmetric, call them <X,Y>. Each XOR group produces low signals on both lines whenever X=Y. The only situation in which a positive signal comes from an XOR NOR gate is when the accompanying X is low and Y is hi, Thus, positive signals indicate a comparison, a difference between X and Y. Any such positive signal will make the connected NOR in the NOR chain low. When any pair is the same, the connected NOR will ignore it, checking if any other pairs are the same. The NOR chain returns low only when all signal pairs match.

Examining **S6b**: whenever a pair is different, one line will be hi, making the OR3 gate hi. This is then inverted to low and sent to an AND gate, which returns low. The AND result s are accumulated by the final OR5; a low signal from the AND is ignored. The XOR paired wires also go directly to the AND gates. Whenever X is hi and Y is low, out2 will be hi and out3 will be low. And symmetrically when X is low and Y is hi. Thus out2 collectively indicates that at least one X is different than Y and is hi.

The table for this:

Signals		*Effect*
X=low	Y=hi	out2=low, out3=hi
X=hi	Y=low	out2=hi, out3=low
X=Y		none; when all X=Y, out2=low, out3=low

We can now summarize: CM85A is a four-bit comparator. When all four bits are equal (and when the ganging signals are low), out1=hi, out2-low out3=low. When any pair does not match, if the higher bit is hi and the lower bit is low, then out2=hi and out3=lo. Reversed otherwise.

Out1 is four-bit equality.
Out2 is four-bit greater-than.
Out3 is four-bit less-than.

COMPARISON TO PUBLISHED RESULTS (DATED)

In comparison with recent (1996) published results for the CM85A circuit, our techniques performed well. [Reference: Tsai (Mentor Graphics) and Marek-Sadowska (UCSB), Multilevel Logic Synthesis for Arithmetic Functions, DAC'96 p242]. Three different systems are compared below.

- SIS: UC Berkeley optimizing tool used by many CAD companies
- DAC: 1996 academic research results
- Losp: void-based techniques

METRIC: number of wires

Wire count indirectly measures circuit complexity, since the dominant fabrication problem is connecting functional gates. In deep sub-micron technologies, wire count rather than transistor count measures chip area as well.

The first set of results is after logic synthesis but before physical layout. Thus the first results are mathematical and independent of semiconductor technology. This measurement technique assumes that the results of logic synthesis are not deconstructed when the abstract functionality is laid out on a physical substrate. Most CAD tools *do* deconstruct the logic optimization because synthesis and layout have traditionally been seen as independent sequential tasks often done by different engineers.

source	*wire-count*	*CPU time*
SIS	80	1.7 sec in C on Sparc 5
DAC	84	1.48 sec
Losp	64	0.52 sec in LISP on PowerPC 8100

The second set of results is after layout, and is more pragmatically based. Losp result reflect an integrated synthesis and technology mapping strategy, since both are achieved by the same transformation algorithms.

source	*gate-count*	*wire-count*
SIS	33	77
DAC	41	84
Losp	23	47

COMMENTS ON ALGORITHMS

SCHEMATIC 1: Two-level Logic

The distribution theorem is

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

Continuing application of Distribute will convert a parens form to two-level logic.

Example:

```
                  ((a) ((b) ((c) (d))))              3 levels
                  ((a) ( ((c (b)) (d (b)))))         distribute (b)
                  ((a) (c (b)) (d (b)))              clarify,  2 levels
```

SCHEMATIC 2: Multilevel Benchmark Circuit

This circuit is a direct parse from EDIF to pun format. The standard conversion table is used:

```
            FALSE                          <void>
            TRUE                           ( )
            (NOT a)                        (a)
            (a OR b)                       ((a b))
            (a AND b)                      ((a)(b))
            (IF a b)                       (((a) b))
            (IF a THEN b ELSE c)           (((a) b)(a c))
            (a XOR b)                      ((a b)((a)(b)))
```

SCHEMATIC 3: Suppress Inverters

A cell in a pun file consists of

```
      (id parens-form)
```

Expanding a pun file consists of substituting the parens-form for the id wherever it occurs within the rest of the pun body. Example using a circuit fragment:

```
            (~1  (a) )
            (~2  (b ~1) )
```

Expand ~1:

```
            (~2 (b (a)) )
```

For this circuit, all of the inverters have been expanded into the cells which call them.

SCHEMATIC 4: Remove Redundancy

Each cell is handed to the Losp minimization engine, after being fully expanded. This circuit represents a part of Losp functionality, that of using the theorems Dominion, Involution, and Pervasion to minimize a parens form. PF-REDUCE accesses the Losp minimization functionality.

SCHEMATIC 5: Reduce Reconvergence

The distribution theorem is applied to all cells, in the Collect direction. This is equivalent to factoring the Boolean expression.

```
           ((a b)(a c)) ==>  a ((b)(c))              DISTRIBUTION
```

Factoring Pass 1:

```
 ((~29    (H G (j) (F))   )
  (~28    (H G (j) (E))   )
  (~31    ((k ((j) (H))))   )
  (~30    ((i ((j) (G))))   )
  (~10    (E F A B H D G C (j))   )
  (~40    ((~28 ~30 (E F H D G C (j) (A)) ((C) E F H G (j)))) )
  (~41    ((~29 ~31 (E F H D G C (j) (B)) ((D) E F H G (j)))) ) )
```

Factoring Pass 2:

```
 ((~10  (A B C D E F G H (j))   )
  (~40  ((i ((G)(j)) (G H (E)(j)) (C D E F G H (A)(j)) (E F G H (C)(j)))) )
  (~41  ((k ((H)(j)) (G H (F)(j)) (C D E F G H (B)(j)) (E F G H (D)(j))))) )

 ((~10  (A B C D E F G H (j))   )
  (~40  ((i ((j) (G (H (E)) (F H (C (D (A)))))))) )
  (~41  ((k ((j) (H (G (F)) (E G (D (C (B)))))))) ) )

 ((~10  (A B C D E F G H (j))   )
  (~40  ((i ((j) (G (H (E (F (C (D (A))))))))) )
  (~41  ((k ((j) (H (G (F (E (D (C (B))))))))) ) )
```

SCHEMATIC 6a: Collect Signals, poor choice

Cells are expanded in order to create gates with a greater fanin. Example:

```
           (~1 ((a)(b)) )
           (~2 ((c)(d)) )
           (~3 (( ~1 ~2 ((e)(f)) )) )
```

~3 has an irregular form. Expanding ~1 and ~2 into it produces a collection (OR) of three ands:

```
           (~3 (( ((a)(b)) ((c)(d)) ((e)(f)) )) )
```

In this circuit, three 4 input NOR gates are created.

SCHEMATIC 6b: Collect Signals, good choice

Cells can be expanded in many ways. Some choice of expansion are poor, others are good, with respect to the simplicity of the entire circuit. Here, two 5-input NORs are created, leading to a more elegant circuit.

In general, what cells are expanded and when is an interactive design choice.

SCHEMATIC 7: Increase Testability

The fully expanded version of this circuit is given to the function BUILD-DISTINCTION-NET, which creates a structure composed solely of NOR gates. This has the effect of removing reconvergent paths and thus enhancing testability.

SCHEMATIC 8: Specific Delay (6 gates)

no comment

SCHEMATIC 9: Pipeline (3 gates)

no comment

SCHEMATIC 10a: Specific Library, poor choice

Cells can be manipulated to be composed of specific collection of gate types. Here, these gates are used: NOT, NOR2, AND2, OR2, XOR2. This collection produces a clumsy circuit.

SCHEMATIC 10b: Specific Library, good choice

Here the circuit is mapped to this collection: NAND2, NOR2, AND2, OR2, NXOR2. This collection produces a circuit with nice structure.

SCHEMATIC 10c: Arbitrary Mapping

no comment, no longer in collection

SCHEMATIC 11: Three-level Logic

no comment

SCHEMATIC 12: Map to NAND Gates

Here the circuit is mapped to the cell library: NOT, NAND2. NAND2 gates are very efficient wrt transistor usage. Thus this circuit is best for saving area on a chip.

SCHEMATIC 13: Map to FPGA (4-LUTs)

Some FPGA architectures have lookup tables rather than logic gates. A lookup table with N inputs can simulate the functionality of any collection of logic gates for those inputs. Here the circuit is partitioned not by type of logic gate, but by groups of 4 inputs.

Assignment Sequence (bottom up condensation):

Pass 1:

```
            #1= (C (D (A)))
            #2= (D (C (B)))
            #9= A B C D
           #10= E F G H

  ((~10  (#9 #10 (j))   )
   (~40  ((i ((j) (G (H (E (F #1)))))) )
   (~41  ((k ((j) (H (G (F (E #2)))))) ) )
```

Pass 2:

```
            #3= (E (F #1))          3-var
            #4= (F (E #2))          3-var
           #11= ((j) #9 #10)        3-var

  ((~10  #11   )
   (~40  ((i ((j) (G (H #3)))) )
   (~41  ((k ((j) (H (G #4)))) ) )
```

Pass 3:

```
            #5= ((j) (G (H #3)))
            #6= ((j) (H (G #4)))

  ((~10  #11   )
   (~40  ((i #5)) )
   (~41  ((k #6)) ) )
```

Pass 4:

```
            #7= (( i #5 ))          2-var
            #8= (( k #6 ))          2-var

  ((~10  #11   )
   (~40  #7 )
   (~41  #8 ) )
```

Final LUT Assignment:

```
 #1= (C (D (A)))
 #2= (D (C (B)))
 #3= (E (F #1))           3-var
 #4= (F (E #2))           3-var
 #5= ((j) (G (H #3)))
 #6= ((j) (H (G #4)))
 #7= (( i #5 ))           2-var
 #8= (( k #6 ))           2-var
 #9= A B C D
#10= E F G H
#11= ((j) #9 #10)         3-var
```

SCHEMATIC 14: Binary Decision Diagram

BDDs do not represent circuits, although they can answer come questions about circuits. The function BDD builds BDDs from circuits. These in turn can be used to generate an exhaustive set of test vectors for a circuit.

SCHEMATIC 15: Low-level Abstraction

Hierarchical clustering, or library abstraction, permits a designer to control complexity by grouping circuit portions with similar logical structure. Pattern-matching across parens in cells identifies abstract patterns for hierarchical abstraction. Examples follow

SCHEMATIC 16: Emphasize Wiring

A different set of patterns emphasizes wiring.

SCHEMATIC 17: Emphasize Sequential Structure

Different patterns identify different circuit characteristics. This partition emphasizes the critical path through the circuit.

SCHEMATIC 18: Emphasize Parallel Structure

A different set of patterns emphasizes components which trigger in parallel.

SCHEMATIC 19: Emphasize Output Structure

Here the patterns isolate entire subcircuits connected to each output.

SCHEMATIC 20: Abstract Bit Width

no comment

SCHEMATIC 21: Distinction Network I

BUILD-DISTINCTION-NETWORK is called on circuit 6b, creating a structure composed solely of NOR gates.

SCHEMATIC 22: Distinction Network II, less wiring

BUILD-DISTINCTION-NETWORK is called on circuit 7, creating a structure composed solely of NOR gates.

SCHEMATIC 23a: Occlusion Array, optimized circuit

The Occlusion Array is a hardware architecture based on boundary logic. It displays circuit functionality as a pattern in memory. The occlusion logic reads this memory, returning the functional evaluation (the output) of the circuit (rather than returning literally what is in memory). This array is of circuit 6b.

SCHEMATIC 23b: Occlusion Array, testable circuit

Occlusion arrays are built from distinction networks. Different networks produce different occlusion arrays. This array is of circuit 7.

SCHEMATIC 23c: Occlusion Array, 2-level circuit

This array is of circuit 1.

SCHEMATIC 23d: Occlusion Array, multilevel benchmark

This array is of circuit 2.

SCHEMATIC 23e: Occlusion Array, clean multilevel benchmark

no comment

SCHEMATIC 24a: Comesh (multilevel)

no comment

SCHEMATIC 24b: Comesh (two-level)

no comment

SCHEMATIC 25: Bit Stream Simulator

Bindings and Conversion to Bit-stream:

Each output consists of a fully expanded parens form. Full expansion means no internal cell names, only inputs and parens in the form. Input variables are bound to their values. This translates into a pure parens form, only nested parentheses. Using the encoding (=1 and)=0, these can be converted into bit-streams. The function PROCESS-BITPF evaluates these bit-streams in a single pass, thus simulating the circuit functionality on conventional hardware.

Stack process:

```
(BITPF "11100110011001101100011011000110100111000110100111000")
(BITPF "1            110    0110    011  001    011  001    00")
(BITPF "1                                1    0      1    00")
(BITPF "1                                1                00")
(BITPF "                                                   ")
(VALUE= FALSE TICKS= 4 ENTROPY= 27)


(BITPF "11111001110100111100011101001111000111011000110110110000000000000")
(BITPF "111    111  0011    0111  0011    01110    0110110    0000000000")
(BITPF "111    1      11    01      11    01        110        000000000")
(BITPF "111    1      11    01      11    01                    00000000")
(BITPF "111    1      11    01      11                          00000000")
(BITPF "111    1      11    01                                    000000")
(BITPF "111    1      11                                          000000")
(BITPF "111    1                                                    0000")
(BITPF "11                                                            00")
(BITPF "                                                                ")
(VALUE= FALSE TICKS= 9 ENTROPY= 32)


(BITPF "11111001111000111010011110001110100111011000110110110000000000000")
(BITPF "111    11    0111  0011    0111  001110    0110110    0000000000")
(BITPF "111    11    01      11    01      1        110        000000000")
(BITPF "111    11    01      11    01      1                    00000000")
(BITPF "111    11    01      11    0                            000000")
(BITPF "111    11    01                                          00000")
(BITPF "111    11                                                00000")
(BITPF "111                                                        000")
(BITPF "1                                                           0")
(VALUE= TRUE TICKS= 8 ENTROPY= 31)
```

LOSP GENERATED STATISTICS FOR CM85A EXAMPLES

Optimization

- SCHEMATIC 1: Two-level Logic (PLD)

```
(cm85a ((i-o 11-03)(cell 52)(lits 226)(nets 63)(path 2))
 ((inv 11)(or 38)(and 1)(nor 0)(nand 2)(eq 0)(xor 0)(lib 0)(mix 0)(gates 41)))
```

- SCHEMATIC 2: Multilevel Benchmark Circuit (cm85a)

```
(cm85a ((i-o 11-03)(cell 76)(lits 120)(nets 87)(path 8))
 ((inv 32)(or 18)(and 26)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 44)))
```

- SCHEMATIC 3: Suppress Inverters

```
(cm85a ((i-o 11-03)(cell 55)(lits 99)(nets 66)(path 8))
 ((inv 11)(or 2)(and 21)(nor 16)(nand 5)(eq 0)(xor 0)(lib 0)(mix 0)(gates 44)))
```

- SCHEMATIC 4: Remove Redundancy

```
(cm85a ((i-o 11-03)(cell 45)(lits 81)(nets 56)(path 7))
 ((inv 13)(or 16)(and 8)(nor 8)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 32)))
```

- SCHEMATIC 5: Reduce Reconvergence

```
(cm85a ((i-o 11-03)(cell 37)(lits 69)(nets 48)(path 6))
 ((inv 13)(or 8)(and 8)(nor 8)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 24)))
```

- SCHEMATIC 6a: Increase Fanin

```
(cm85a ((i-o 11-03)(cell 35)(lits 67)(nets 46)(path 6))
 ((inv 11)(or 3)(and 6)(nor 15)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 24)))
```

- SCHEMATIC 6b: Increase Fanin

```
(cm85a ((i-o 11-03)(cell 35)(lits 67)(nets 46)(path 6))
 ((inv 13)(or 6)(and 8)(nor 8)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 22)))
```

- SCHEMATIC 7a: Enhance Testability

```
(cm85a ((i-o 11-03)(cell 40)(lits 72)(nets 51)(path 9))
 ((inv 13)(or 2)(and 0)(nor 21)(nand 0)(eq 4)(xor 0)(lib 0)(mix 0)(gates 27)))
```

- SCHEMATIC 7b: Enhance Testability

```
(cm85a ((i-o 11-03)(cell 36)(lits 68)(nets 47)(path 9))
 ((inv 11)(or 2)(and 0)(nor 23)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 25)))
```

Design Constraints

- SCHEMATIC 8: Reduce Critical Path (6 gates)

```
(cm85a ((i-o 11-03)(cell 40)(lits 80)(nets 51)(path 6))
 ((inv 15)(or 4)(and 0)(nor 21)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 25)))
```

- SCHEMATIC 9: Pipeline (3 two-input gates)

```
(cm85a ((i-o 11-03)(cell 43)(lits 75)(nets 54)(path 9))
 ((inv 11)(or 9)(and 0)(nor 23)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 32)))
```

- SCHEMATIC 10a: Map to Specific Library

```
(cm85a ((i-o 11-03)(cell 39)(lits 79)(nets 50)(path 10))
 ((inv 7)(or 10)(and 9)(nor 8)(nand 1)(eq 0)(xor 4)(lib 0)(mix 0)(gates 32)))
```

- SCHEMATIC 10b: Map to Specific Library

```
(cm85a ((i-o 11-03)(cell 32)(lits 64)(nets 43)(path 9))
 ((inv 0)(or 2)(and 4)(nor 12)(nand 10)(eq 4)(xor 0)(lib 0)(mix 0)(gates 32)))
```

- SCHEMATIC 11: Three-level Logic

```
(cm85a ((i-o 11-03)(cell 23)(lits 67)(nets 34)(path 4))
 ((inv 8)(or 2)(and 9)(nor 0)(nand 0)(eq 4)(xor 0)(lib 0)(mix 0)(gates 15)))
```

- SCHEMATIC 12: Map to NAND Gates

```
(cm85a ((i-o 11-03)(cell 64)(lits 96)(nets 75)(path 9))
 ((inv 32)(or 0)(and 0)(nor 0)(nand 32)(eq 0)(xor 0)(lib 0)(mix 0)(gates 32)))
```

- SCHEMATIC 13: Map to FPGA (4-LUTs)

```
(cm85a ((i-o 11-03)(cell 11)(lits 57)(nets 77)(path 9))
 ((inv 0)(or 2)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 9)(gates 2)))
```

- SCHEMATIC 14: Binary Decision Diagram

```
(cm85a ((i-o 11-03)(cell 3)(lits 49)(nets 75)(path 9))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))
```

Hierarchical Abstraction

- SCHEMATIC 15: Abstract Low-level Components

```
(cm85a ((i-o 11-03)(cell 10)(lits 0)(nets 0)(path 0))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 10)(mix 0)(gates 0)))
```

- SCHEMATIC 16: Abstract for Component Connectivity

```
(cm85a ((i-o 11-03)(cell 12)(lits 12)(nets 0)(path 0))
 ((inv 2)(or 2)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 8)(mix 0)(gates 2)))
```

- SCHEMATIC 17: Abstract for Sequential Structure

```
(cm85a ((i-o 11-03)(cell 6)(lits 0)(nets 0)(path 0))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 6)(mix 0)(gates 0)))
```

- SCHEMATIC 18: Abstract for Parallel Structure

```
(cm85a ((i-o 11-03)(cell 7)(lits 9)(nets 0)(path 0))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 6)(mix 1)(gates 0)))
```

- SCHEMATIC 19: Abstract for Output Structure

```
(cm85a ((i-o 11-03)(cell 3)(lits 17)(nets 0)(path 0))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 2)(mix 1)(gates 0)))
```

- SCHEMATIC 20a: Abstract Bit-width (recursive)

```
(1bit-magcomp ((i-o 02-03)(cell 3)(lits 8)(nets 14)(path 2))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))

(nbit-magcomp-recursive ((i-o 02-03)(cell 3)(lits 15)(nets 20)(path 3))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))
```

- SCHEMATIC 20b: Abstract Bit-width (enables)

```
(0bit-magcomp-enable ((i-o 03-03)(cell 3)(lits 4)(nets 6)(path 1))
 ((inv 0)(or 2)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 2)))

(1bit-magcomp-enable ((i-o 05-03)(cell 3)(lits 13)(nets 25)(path 2))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))

(nbit-magcomp-enable ((i-o 05-03)(cell 3)(lits 8)(nets 15)(path 3))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))
```

• SCHEMATIC 20c: Abstract Bit-width (enables, recursive)

```
(nbit-magcomp-enable-recursive ((i-o 05-03)(cell 3)(lits 20)(nets 31)(path 5))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))
```

Novel Hardware Models

• SCHEMATIC 21: Distinction Network I

```
(cm85a ((i-o 11-03)(cell 36)(lits 76)(nets 47)(path 6))
 ((inv 14)(or 0)(and 0)(nor 22)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 22)))
```

• SCHEMATIC 22: Distinction Network II

```
(cm85a ((i-o 11-03)(cell 36)(lits 70)(nets 47)(path 9))
 ((inv 11)(or 0)(and 0)(nor 25)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 25)))
```

• SCHEMATIC 23a: Occlusion Array (Dnet 21)

```
(cm85a ((i-o 11-03)(cell 36)(lits 76)(nets 47)(path 6))
 ((inv 14)(or 0)(and 0)(nor 22)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 22)))
```

• SCHEMATIC 23b: Occlusion Array (Dnet 22)

```
(cm85a ((i-o 11-03)(cell 36)(lits 70)(nets 47)(path 9))
 ((inv 11)(or 0)(and 0)(nor 25)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 25)))
```

• SCHEMATIC 23c: Occlusion Array (two-level Dnet)

```
(cm85a ((i-o 11-03)(cell 52)(lits 226)(nets 63)(path 2))
 ((inv 11)(or 0)(and 0)(nor 41)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 41)))
```

• SCHEMATIC 23d: Occlusion Array (raw multilevel benchmark)

```
(cm85a ((i-o 11-03)(cell 120)(lits 164)(nets 131)(path 8))
 ((inv 76)(or 0)(and 0)(nor 44)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 44)))
```

• SCHEMATIC 23e: Occlusion Array (clean multilevel benchmark)

```
(cm85a ((i-o 11-03)(cell 62)(lits 102)(nets 73)(path 8))
 ((inv 22)(or 0)(and 0)(nor 40)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 40)))
```

• SCHEMATIC 24a: Comesh (multilevel)

(4bit-magnitude-comparator-with-enables
 ((i-o 11-03)(cell 36)(lits 70)(nets 47)(path 9))
 ((inv 11)(or 0)(and 0)(nor 25)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 25)))

• SCHEMATIC 24b: Comesh (two-level)

(cm85a ((i-o 11-03)(cell 52)(lits 226)(nets 63)(path 2))
 ((inv 11)(or 0)(and 0)(nor 41)(nand 0)(eq 0)(xor 0)(lib 0)(mix 0)(gates 41)))

• SCHEMATIC 25: Bit-stream Simulator

(cm85a ((i-o 11-03)(cell 3)(lits 49)(nets 75)(path 9))
 ((inv 0)(or 0)(and 0)(nor 0)(nand 0)(eq 0)(xor 0)(lib 0)(mix 3)(gates 0)))