Design methodology for IBM ASIC products

by J. J. Engel
T. S. Guzowski
A. Hunt
D. E. Lackey
L. D. Pickup
R. A. Proctor
K. Reynolds
A. M. Rincon
D. R. Stauffer

The IBM ASIC design methodology enables a product developer to fully incorporate the high-density, high-performance capabilities of the IBM CMOS technologies in the design of leading-edge products. The methodology allows the full exploitation of technology density, performance, and high testability in an ASIC design environment. The IBM ASIC design methodology builds upon years of experience within IBM in developing design flows that optimize performance, testability, chip density, and time to market for internal products. It has also been achieved by using industry-standard design tools and system design approaches, allowing IBM ASIC products to be marketed externally as well as to IBM internal product developers. This paper describes the IBM ASIC design methodology, and then focuses on the key areas of the methodology that enable a customer to exploit the technology in terms of performance, density, and testability, all in a fast-time-to-market ASIC paradigm. Also emphasized are aspects of the methodology that allow IBM to market its design experience and intellectual property.

Introduction: Overview of IBM ASIC design

The IBM ASIC design methodology is a process for designing high-density, high-performance, highly testable, fast-time-to-market ASIC chips. It can be used for both standard cell and gate-array designs in chip sizes of up to 1.6 million wirable gates. Sign-off (final approval) for fabrication of an IBM ASIC is based on attaining high-coverage testability through full-scan design and timing verification using static timing analysis.[1] Support for numerous electronic design automation (EDA) tools is provided in a design kit containing model libraries for each IBM ASIC technology. The steps supported in the IBM flow are described below and shown in **Figure 1**.

- *Design entry*

Most customers designing large ASICs today (more than 50 000 gates) enter their designs in a hardware description language (HDL) such as VHDL[2] or Verilog®. This is accomplished via manual HDL language entry, or through the use of HDL entry tools from an EDA vendor. For those customers desiring direct schematic entry of their design, symbol libraries for various schematic editors [1, 2] are provided.

- *Logic synthesis*

Complete library support is provided for logic synthesis, which optimizes the customer's technology-independent HDL and maps it into a gate-level, technology-dependent representation. Component instantiation of gate-level library elements in the high-level description is also

[1] David Lackey, "IBM ASIC Design Methodology Overview," *ASIC Application Note* (IBM internal document), IBM Microelectronics Division, Essex Junction, VT, 1995.
[2] VHDL: VHSIC Hardware Description Language, IEEE Standard 1076.

0018-8646/96/$5.00 © 1996 IBM

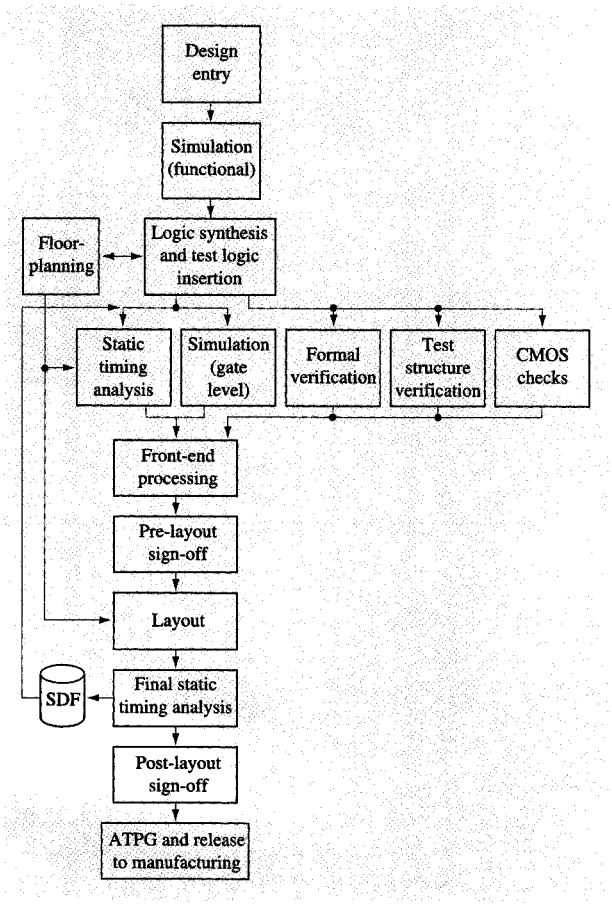

Figure 1

Flowchart of IBM ASIC design methodology.

supported. By using the IBM design methodology, a gate-level design is synthesized from the same functional HDL design representation that is simulated for logical correctness.

Using either IBM internal or external design synthesis tools, the IBM ASIC logic synthesis methodology produces a design that satisfies level-sensitive scan design (LSSD) design-for-test requirements.

- *Simulation*

IBM supports a variety of leading-edge tools for functional and gate-level simulation for designs using VHDL, Verilog [3], and design languages internal to IBM. Included are simulators that allow mixed simulation of VHDL and Verilog.

Delay simulation is supported with back-annotation using standard delay format (SDF) files [4] created by the IBM EinsTimer™ [5] delay calculator. This SDF is based upon the early timing estimator (ETE) rules and nonlinear delay models that have been qualified for the IBM ASIC hardware component libraries. Timing simulation is supported in several tools; however, static timing analysis is the hardware sign-off criterion for timing without the need for simulation test vectors. Delay simulation is subject to the maximum timing accuracy of the simulation tool used, as well as the ability of the SDF to accurately represent the delay data. Thus, EinsTimer, when used as a static timing analysis tool, is needed to fully realize the accuracy of EinsTimer's delay calculations using the ETE and nonlinear delay models.

- *Floorplanning*

Floorplanning, which allows the customer to introduce estimates for the placement of functional blocks early in the design cycle, is supported in the design methodology. Without floorplanning, estimates of capacitance for global nets can be highly inaccurate. Floorplanning can provide accurate estimates to guide synthesis of the logic design and, later, to guide the layout process. Floorplanning also provides indications of wiring congestion on the chip, allowing the designer to minimize congestion problems early in the design. Integration of floorplanning with logic synthesis [6] is supported, allowing the customer to iteratively improve design performance using the improved timing characterization enabled by the layout data.

Additionally, a "bit-stacking" process is provided whereby various dataflow blocks in the design are better optimized for timing and area and are fed into the floorplanning/synthesis cycle for optimal design integration.

- *Test structure verification*

IBM requires that all ASICs pass test structure verification (TSV), which uses the IBM internal test structure verification and automatic test pattern generation (ATPG) tool. This ensures compliance with LSSD requirements for scan, clock control, and random-access memory (RAM) and read-only memory (ROM) access requirements. A design that meets these requirements contains fully testable, race-free hardware [7]. TSV forms the basis for the unique ability of the IBM foundry not to require test vectors from the customer.

- *Static timing analysis*

The sign-off tool for timing analysis is EinsTimer. EinsTimer accepts both Electronic Data Interchange File (EDIF) and the IBM VLSI integrated model (VIM) netlist formats. Timing assertions [5] are provided by the customer to characterize the design in terms of its primary inputs (PIs), primary outputs (POs), expected time of arrival (ETA), clock phases (PHS), and don't-care-and-adjust (DCA) data that specify multicycle and "false" paths in the design.

Additional static timing analysis tools from external EDA vendors are supported for use in the customer's design environment, but EinsTimer is required for sign-off.

- *Formal verification*

The IBM ASIC methodology supports a formal verification method known as Boolean equivalence checking (BEC) [8, 9] which is used at various steps in the design flow to ensure that changes made to the netlist, including those made in test insertion, front-end processing, and chip layout, still preserve the original functional HDL design of the chip. This process is far more thorough and cost-effective than simulation. Requirements for expensive and time-consuming gate-level simulation late in the design cycle are drastically reduced. Simulation resources can be focused instead on the original functional HDL design early in the design cycle, where simulation is most effective and the design least costly to correct.

- *CMOS checks*

All IBM ASICs must pass a series of CMOS rules checks (CMOS checks). These checks include electrical connectivity checks such as fan-out, and checks for conformance to I/O requirements, including IBM boundary scan. The CMOS checks utilize rules available in the design kit. CMOS checks provide the remaining checks not otherwise covered in the design methodology to ensure design success in IBM CMOS processes.

- *Design handoff*

When a design team is ready to turn its design over to IBM for layout and fabrication, it provides the IBM design center with a netlist, timing assertion files, an I/O placement list, and specification of any preplaced macros. If floorplanning was used by the customer, placement specifications, region constraints, and parasitic data are provided.

- *Front-end processing*

By customer agreement[3], the IBM ASIC design services organization provides certain netlist transformations (transforms) [10], such as clock tree and test logic insertion. After the transforms have been applied, the netlist is verified by using TSV to check design compliance with LSSD guidelines, and with static timing analysis to ensure that performance targets are met. An SDF file produced by the EinsTimer tool is then provided to the customer for delay simulation, and for any further timing analysis or synthesis optimization that may take place.

- *Pre-layout sign-off*

After all transforms, static timing analysis, and TSV have been run on the customer's netlist, the updated netlist, the verification reports, and the SDF are returned to the customer for simulation and timing analysis, as well as formal verification for those customers with that capability. This step enables the customer to verify that no errors were introduced into the design as a result of the transforms. It is also a checkpoint for the designer and IBM to track how closely the results from EinsTimer match the timing results of the customer's front-end tools. The customer must be satisfied with the EinsTimer timing analysis results before pre-layout sign-off can be completed[4].

- *Layout*

After pre-layout sign-off is achieved, IBM begins the detailed layout of the customer's design. If the customer is designing in the IBM CMOS 4 technology, the layout is done on the IBM mainframe-based physical design system. If the customer is designing in one of the IBM CMOS 5 technologies, layout of the chip is done using workstation-based tools [11].

An IBM ASIC design can be laid out utilizing either a flat or a hierarchical process. Whereas the hierarchical design process may be necessary for a large-gate-count design, the flat design process is often more efficient for smaller designs. Layout is driven by the performance objectives of the design, using capacitance targets generated by static timing analysis, and by linking static timing analysis to the optimization steps within the layout process.

The final netlist and timing delay files are written in the EDIF and SDF languages respectively, and returned to the customer in either flat or hierarchical formats, depending upon whether a flat or hierarchical physical design methodology was used.

- *Post-layout sign-off*

The post-layout netlist, the SDF file, and the timing reports are returned to the customer for final logic and timing verification. The customer can verify that the integrity of his logic function was not corrupted by the physical design process by using either post-layout simulation of the design or formal verification. The customer can use the SDF to re-examine timing for critical paths, using either static timing analysis or delay simulation, and to correlate the results to the path data in the post-layout timing report. The timing reports from EinsTimer are used for post-layout sign-off before the chip enters the fabrication process. If the customer is satisfied

[3] David Koller, "Initial Design Review," *ASIC Application Note* (IBM internal document), IBM Microelectronics Division, Essex Junction, VT, 1995.

[4] Ann Rincon, "Release to Layout Sign-off," *ASIC Application Note* (IBM internal document), IBM Microelectronics Division, Essex Junction, VT, 1995.

with the function and performance of the chip's physical design, post-layout sign-off can be completed[5].

- *Tape-out to manufacturing*

After post-layout sign-off, the IBM design center takes the customer's design into final-shapes generation (creation of mask data), and logical-to-physical and ground-rule checking. At this time the design is run through ATPG to create the patterns that will be used at the manufacturing tester. The mask data and test patterns are delivered to the manufacturing control center for fabrication.

Automatic model generation and verification

One of the challenges in supporting EDA systems from multiple vendors is maintaining a consistent set of model libraries. These libraries must be identical in their modeling of a logic cell's functional behavior, timing, and electrical parameters, yet provide the syntax required by a diverse set of tools. To answer this need, an automatic model generation and verification system known as LibTools™ was developed. This system makes use of a common technology database for consistency and a common set of code for cross-verification.

The common technology database contains most of the information pertinent to synthesis, simulation, and test-generation models. Timing data are retrieved from the early timing estimator (ETE) and delay calculator language (DCL) timing models [5]. The information contained in the database includes cell names, pin names, power levels, cell area, cell functions, and pin functions. Additional model-specific data are stored as required [12–14].

Since timing information is not stored in the database itself, a common set of access routines is used to obtain the data in the timing models. This takes the form of an ETE parser, replaced recently by direct access to the EinsTimer timing analysis tool, which obtains timing data from the models.

Numerous model types are generated from the LibTools database, including

- IBM BooleDozer™ models.
- Synopsys Design Compiler™ and Test Compiler™ models.
- Cadence Verilog-XL™ and Veritime™ models.
- VITAL (VHDL Initiative Towards ASIC Libraries) models.
- Cadence Composer™ symbols.
- IBM TestBench™ models.
- Mentor QuickSim II™ models.
- Compass PMD format, which is used by the Compass Mercury™ Library Development Tools to provide models for vendor tools such as Viewlogic VIEWSIM®.

A typical model-generation function accesses a requested cell in the LibTools database, retrieving both generic and model-specific data for construction of the target model type. The common database preserves data consistency, whereas multiple model-generation functions provide the diverse set of models required in supporting a diverse set of customers.

Verification of the generated models is done by applying a simulation pattern set, generated from a transistor-level representation of each library cell that has been qualified against manufactured hardware. Additionally, the various model types are cross-verified for consistency and additional coverage. This is important for patterns not easily realized in actual hardware, yet realizable in simulators, such as conflicting data on two ports of a latch, and other patterns designed to cause failures or to identify opportunities for pessimism reduction.

The verification program contains a pattern generator, which accesses the common database to determine a suitable pattern set for the function of the target cell. For combinational circuits, an exhaustive pattern set is used, whereas for bidirectional (BIDI) elements, the high-impedance Z state is added to the existing 0, 1, and unknown (X) value set. The BIDI element is tested exhaustively both as a driver and as a receiver; then an X state is applied to the driver-enable circuit for testing in an unknown mode. Latches, multiple-bit cells, and memories have unique pattern generators which fully exercise the cell.

A common data structure is used to define the test patterns. Common parsing routines are used to create the pattern formats unique to each target tool type. Patterns are submitted to the known good reference, or "golden," simulator to generate the golden pattern set.

The IBM ASIC design methodology resolves the key issue of accounting for subtle differences in behavior among the various tools. Whereas differences between tool behavior and the golden pattern set would normally signify a model or tool error, some differences are found to be justified, and a correction action must occur. Either a global correction file or a model-specific correction file can be employed in this situation.

An example of a global correction is that made for a circuit that produces a pessimistic result. A multiplexor would produce an X output given an X on the select pin in the golden pattern set, which was generated from the transistor-level schematic. However, in the case of equal data inputs, the output is, in fact, known despite the X value of the select pin. This known value is incorporated

[5] Ann Rincon, "Release to Manufacturing Sign-off," *ASIC Application Note* (IBM internal document), IBM Microelectronics Division, Essex Junction, VT, 1995.

in the global correction file, allowing simulation to be less pessimistic.

A model-specific correction is applied, for example, when a target tool cannot fully implement a certain function. A common example among simulators is the approximation of the behavior of a bidirectional pin.

An automatic correction file generator has been developed that produces a file of recognized differences between the golden pattern set and the patterns of the models and tools being verified. Each correction must be manually signed off, however, before it can be added to the actual correction files to be used for subsequent model verification.

An output of verification is a list of successful and unsuccessful pattern comparisons. This list is used to determine when a model must be reverified, which occurs when the prior comparison was unsuccessful, or when the model has changed since a prior successful comparison.

Some complex cells, such as phase-locked loops [12–14], require custom input patterns and manual examination of output behavior. For such cells, manual sign-off is required, and the justification is stored with the successful comparison record in the database.

For logic synthesis models, a simulation model is program-generated using the synthesis model behavior, and the pattern-compare strategy is employed.

Consistency of timing behavior between the timing models and the target tool is verified by creating an SDF for each cell using EinsTimer, and loading the SDF into each target tool. The ETE or DCL models, which contain timing delays for input-to-output combinations as well as specific checking scenarios, are compared against the timing behavior of the target tool and model, and the database is updated with the appropriate record of successful or unsuccessful comparison.

Additional verification is provided by the generation of an all-instance netlist containing one of each type of library cell. Each cell's input and output is a netlist I/O, and the above verification steps are applied.

The above capabilities, depicted in **Figure 2**, provide a powerful system for the automatic generation and verification of a diverse set of technology models, using a common database that ensures consistency among the models and provides a strict audit trail of each model's quality.

Static timing analysis

The IBM ASIC design methodology incorporates a timing verification methodology based on static timing analysis, the product of almost twenty years of experience in system development projects within IBM. This timing methodology uses the IBM EinsTimer [5] tool as the central timing analysis and delay calculation tool.

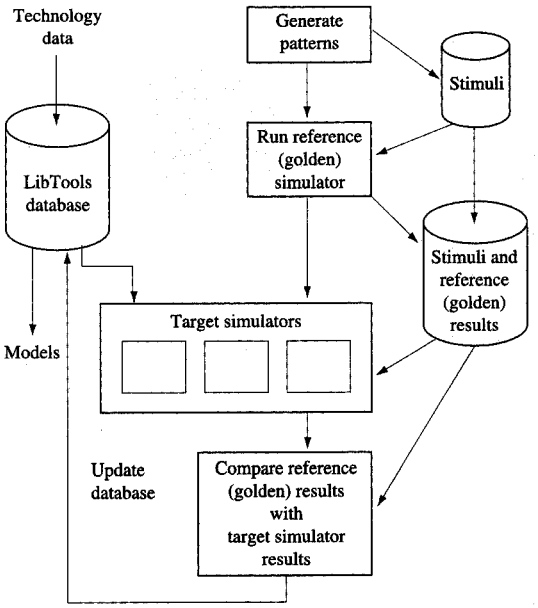

Figure 2

IBM ASIC model-generation process.

With the application of static timing analysis to ASIC design, IBM is unique and a leader in an industry that has traditionally used delay simulation for timing verification and sign-off [3, 4, 15]. The effectiveness of delay simulation is largely determined by how well the simulation test-set covers all logic paths in the design. Additionally, the timing granularity of the simulator is a further limit on the effectiveness of the analysis. It has been found that the EinsTimer static timing analysis tool provides timing verification of greater accuracy as well as greater speed. All paths are timed, as compared to the limited and unknown path coverage of delay simulation, which is the case regardless of the speed of the simulator, and regardless of whether the simulator is a software tool or a hardware accelerator.

For delay calculation, EinsTimer uses multiple wire-load models for a design, rather than single capacitance and *RC* estimates. These wire-load models contain capacitance and *RC* data for a logical group of cells, either based on the number of cells in the group, or as determined in an analytic fashion by a floorplanning tool or by the results of physical design.

For a flat, unfloorplanned chip, a single wire-load model that applies to the given die size is used. This can include flat (logically flattened) designs or designs that contain a logical hierarchy.

A chip can be floorplanned either with a single level of physical hierarchy or as multiple floorplanned groups. The parasitic (wire-load) data determined by the floorplanner are used for the entire chip in the case of a single hierarchical level. For multiple levels of physical hierarchy, the parasitic information for each floorplanned group can be used together with the chip-level parasitics for the connections between groups. EinsTimer can also use wire-load models derived by the customer using various tools, allowing flexibility in the early development process.

EinsTimer has traditionally used, for gate-level timing models, early timing estimator (ETE) models. ETE models are a human-readable representation of input-to-output timing paths, with pin capacitance and coefficients for delay and transition, as well as tests (such as setup-and-hold) to be performed between input pins. When using ETE models, the delay calculator tool (EinsTimer) contains the delay equations that make use of these delays and parameters.

In advancing the effectiveness of delay calculation, IBM has recently incorporated new delay rules (NDRs). These new gate-level timing models replace the ETE models, and are written in the delay calculation language (DCL), which was developed by IBM but is becoming an industry standard in the future evolution of EDA tools [16].

Through the use of NDRs, ownership of the delay calculation equation is moved from the timing tool to the ASIC foundry, by incorporating the delay equation in DCL data provided by the foundry and compiled together with the NDRs. The result is better correlation of timing results across tools, such as static timing analysis and synthesis optimization, which incorporate delay calculation into their algorithms.

NDRs provide better protection of the intellectual assets contained within the data, parameters, and equations, as compared to ETE models, because the data delivered by the ASIC foundry are compiled rather than human-readable. NDRs require an application program interface (API) to obtain the internal data.

For IBM ASIC timing sign-off, EinsTimer is run in its linear-combination delay mode (LCD mode). A combination of scaling factors and best-case/worst-case delays is used to accurately account for process variation in manufacturing. Specifically, path scaling factors are used for the clocks to account for process-induced clock skew:

- A scaling factor is applied to the path from the oscillator input, through the clock-powering tree, through the slave latch, and through combinatorial logic to the master-latch data input.
- Another scaling factor is applied to the path from the oscillator input, through the clock-powering tree to the master-latch clock input.

These factors account for variations across the chip in channel length and width, threshold voltage, and wiring density, which in turn have a number of causes. These include nonplanarity of the wafer or mask; mask defects; horizontal, vertical, rotational, or Z-axis skew between mask and wafer; variation of feature density across the chip; vibration; light intensity variation; implant variation; and crystal defects.

LCD coefficients for a target technology are determined through a combination of steps using EinsTimer and ASX (a circuit-level simulator developed by IBM):

1. Using ASX, 3-sigma path delay variation for a number of clock distribution paths is estimated, taking into account variations in the target technology for the physical factors described above. This estimation using ASX simulation produces 3-sigma path delay variation, a capacitance file, and an RC delay file.
2. LCD coefficients are determined iteratively, through comparison of EinsTimer results (using estimated LCD coefficients) with the ASX results. The LCD coefficients are iteratively re-estimated until the EinsTimer and ASX results match.
3. The above steps are executed for late-mode worst-case, early-mode best-case, and late-mode nominal conditions.

Correlation of timing characteristics

Because the IBM ASIC design methodology requires sign-off using static timing analysis before design layout and again before release to manufacturing, a high correlation between the timing data driving synthesis and the data driving the static timing analysis tool must exist. Conversely, the degree of difference negates the efforts of the designer to achieve performance targets through timing optimization within the logic synthesis tool. For customers using the IBM synthesis tool, this is not a problem, since the EinsTimer delay calculator, which is also used for static timing analysis, is used by synthesis for timing optimization. For customers who are using the Synopsys Design Compiler [17, 18], however, a different form of timing model and a different delay calculator are used. A method was developed to ensure that the Synopsys timing models are generated to provide the highest degree of correlation between Design Compiler and EinsTimer timing results for IBM ASIC designs. For worst-case process, temperature, and voltage, a difference in the delay, computed on a path through a cell, of less than 1% for the entire library is achieved by this method.

Synopsys nonlinear timing models consist of a two-dimensional array of delay values indexed by input transition time and output capacitance, allowing the delay equation to be modeled as a set of values that best match the timing behavior of a cell. The EinsTimer delay

calculator is used to determine the delay values placed in the nonlinear tables on the basis of the ETE or NDR rules [5].

Once the models have been generated, the timing correlation between Synopsys and EinsTimer is verified using an "all-instances test case," which is a test circuit containing one of each type of cell in the technology library. The delay of each input-to-output path of each cell is measured by the EinsTimer delay calculator and compared against the corresponding path delay calculated by the Synopsys Design Compiler. Any delay difference greater than 1% is analyzed. The Synopsys timing tables are then further refined until the correlation target of less than 1% is achieved.

Correlation on wire-load models is 100%, since Synopsys and EinsTimer use the same method for calculating wire delays. The Synopsys wire-load models are derived directly from the EinsTimer models.

Since Synopsys currently has no way to account for the degradation of rising and falling transitions due to the effects of *RC*, any high-fan-out nets will cause timing problems to be seen in EinsTimer that were not seen by Design Compiler. This can be minimized by limiting fan-out to less than fifteen. Testing has shown that once *RC* transition degradation is taken into account, the correlation between EinsTimer and Synopsys will be close to 100% for worst-case assumptions.

Floorplanning

Wire delay for deep-submicron designs can be the predominant part of total path delay. For this delay, a significant advantage can be achieved with floorplanning, which provides for early placement of logical functions, macros, and inputs/outputs (I/Os) on the chip to meet various goals.

Floorplanning increases in importance with greater chip densities and higher performance requirements, and generally becomes a necessary step in the design process for chips with more than one million gates.

Without floorplanning, the prediction of parasitics is based solely upon technology-level data or area-based wire-load models, resulting in significant differences from the actual net parasitics found in the physical design. This limits the ability of logic synthesis to optimize for performance because the synthesis tool is unable to determine path delays accurately.

A logic design environment in which logic synthesis and floorplanning are closely linked is key to shortening design cycles and meeting performance objectives [6]. For one-million-cell designs, a typical iteration through synthesis, floorplanning, and back-annotation of less than one week is typical. This is far less than the traditional iteration through physical design (placement and wire routing).

The advantages gained by customers using floorplanning [19–21] on their designs include the following:

- The effects of a suboptimal I/O assignment are easily seen, giving the customer an earlier opportunity for correction.
- The customer can optimize earlier in the design cycle for die size and metal layers.
- The customer can identify and insert all repowering buffers necessitated by high-fan-out nets.
- Timing closure using accurate parasitics occurs earlier in the design cycle and in the customer's hands, where design changes are less costly.
- The pre-layout static timing analysis sign-off is based upon more accurate parasitics.
- Time in physical design is shorter because of a more predictable design and more accurate pre-layout timing sign-off.

Improved physical design results are achieved through early communication of the logic flow between the foundry and the ASIC customer, enabling an optimized preplacement of RAM, ROM, and growable register array (GRA) macros [12–14]; cell grouping; and development of region placement data for the physical design process. This kind of floorplanning benefits the physical designer. However, there is a far greater advantage if floorplanning is done by the customer in the logic design process. With the use of state-of-the-art floorplanning tools, the logic designer can obtain parasitics and back-annotate these into the synthesis tool, where reoptimization can improve timing and reduce wire congestion. Results from these operations are fed forward to the floorplanning tool. Iteration continues until convergence is achieved.

A highly integrated floorplanning and synthesis design loop is achieved using the "links-to-layout" [6] methodology, which couples the floorplanning tool with the synthesis tool. This is supported in the IBM ASIC methodology using the Synopsys Design Compiler and Floorplan Manager™ tools. An EDIF or Verilog netlist is extracted from a synthesized design and fed into the floorplanning tool, where partitioning and block placement operations take place. The recommended maximum block size is around 2 mm × 2 mm, with an aspect ratio of 0.5 to 2.0. Typically, customers begin with a physical partition that matches the logical hierarchy. If analysis of the floorplan indicates excessive congestion or timing problems, it is possible to repartition the logic into a hierarchy that is different from the logical partition. The Synopsys Floorplan Manager can resolve these differences. Normal floorplanning operations such as analysis of congestion, connectivity between blocks, and wiring track allocation are applied until a promising floorplan results. At this time, the locations of RAMs, ROMs, GRAs, bit

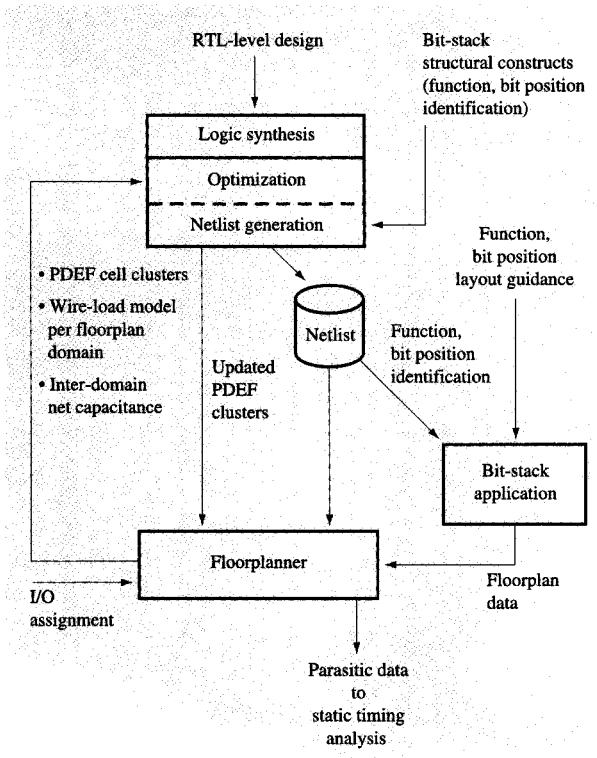

Figure 3

Flowchart of floorplanning and bit-stacking methodology.

stacks, and any other large macros are fixed. The next step is a quick but detailed cell placement, yielding a very close approximation of the final placement. Parasitics and wire-load models from the floorplan are then extracted and back-annotated to the logical design. There are various strategies for back-annotation involving combinations of exact parasitics and wire-load models. Normally, exact parasitics are used for interdomain nets (nets connecting to receiving and driving cells of floorplanning blocks), and wire-load models are used for the remaining nets.

Changes to the logical design, based on the back-annotation of parasitic data, can take several forms. The easiest change is the modification of a cell's power level. With this, an engineering change order (ECO) file can be used to communicate changes to the floorplanning tool, and the existing floorplan can be reused, since no new cells have been added. However, resynthesis of critical paths results in a new netlist. To avoid a modification to the floorplan, a physical definition (PDEF) file containing cell cluster information is produced by the floorplan tool and passed back to synthesis, which produces a new PDEF file with cluster information that includes the new cells in the design, preserving the existing floorplan. Logical restructuring or rearchitecting logic, however, is a more extensive netlist change that requires a new floorplan.

Upon completion of the iterative synthesis/floorplanning process, the designer provides region constraints and preplacement data for I/Os, macros, bit stacks, and interdomain cells to the physical design process. Parasitics for all nets are extracted by the floorplanning tool and fed to the IBM ASIC design center for static timing analysis.

Early experience in processing chips which have been floorplanned by ASIC designers has shown many of the anticipated benefits. Areas for continued improvement in floorplanning include better support of hierarchical design, timing-driven floorplanning and apportionment of timing paths across synthesized entities, optimization of wire codes, porosity modeling, and reduction in time spent in the synthesis/floorplanning cycle.

The floorplanning methodology flow, and its interactions with the bit-stacking methodology (described in the next section), are shown in **Figure 3**.

Bit stacking

The IBM ASIC bit-stacking methodology addresses the need for greater chip density and performance. When bit stacking is applied to designs with a high dataflow content, such as processors using large arithmetic logic units (ALUs), registers, multiplexors, and shifters, the utilization of available gates (or density) is improved from around 60% to well over 70%. This is achieved through utilization by the dataflow logic of nearly 90% of the total utilized area.

The design of a bit stack uses the following general methodology. First, dataflow logic is logically separated from control logic. Then, each dataflow bit position, typically spanning numerous dataflow functions, is isolated, and its circuits are placed along the same circuit row, minimizing the amount of wire required for the buses. Finally, control circuits that communicate with functions common to all bits are placed across circuit rows. Dataflow wires are placed horizontally on metal levels 1, 3, and 5. Control wires are placed vertically on metal levels 2 and 4.

In addition to reduced chip area, the above method often results in smaller control and dataflow circuits owing to a lower drive strength requirement, which in turn provides improvements in performance of typically 10% or more.

By comparison, current logic synthesis tools have difficulty in distinguishing dataflow from control circuits. Control circuitry is created without regard to dataflow placement. A gating signal for a wide data bus, for example, may contain synthesized buffers that fan out to widely separated bit positions, creating unnecessary additional wiring. Synthesized implementations of

different dataflow bit positions and their corresponding control logic often differ, sometimes with varying fan-in of control signals.

Bit stacking is a partially manual process which involves the following steps:

1. Datapath logic is partitioned separately from control logic, and encoded in the HDL source language [3] using structural constructs.
2. A bit stack may contain a single entity or a hierarchy of entities. A netlist representing the bit stack is processed to produce a floorplanning file that contains a fixed placement location for each circuit relative to its floorplanning group. The fixed-placement data are processed in a floorplanning tool, resulting in highly accurate parasitic data that are fed back to logic synthesis for further optimization of those logic paths that interact with the bit stack.
3. Iterative improvements can be made to the HDL of the data path, taking advantage of open space revealed by the prior steps.
4. During physical design of the chip, the bit stacks are given fixed placements relative to the chip. Other floorplanned regions are placed according to their normal floorplanning constraints. Placement of the remaining logic is determined solely by place-and-route optimization. Since the bit stacks are already resolved, run times for place-and-route are reduced.

An off-the-shelf library of bit-stack modules is under development for the design of bit stacks in IBM ASIC technologies; it includes

- Modules compatible with Synopsys DesignWare®, with an emphasis on arithmetic functions.
- *n*-bit modules of functions typically found in dataflows, including buffering trees, register slices, and multiplexors.
- "Gate wrappers" that facilitate portability among technologies by encapsulating technology-dependent components within a technology-independent shell.

LSSD insertion using an external vendor synthesis tool

Level-sensitive scan design (LSSD) is a requirement of the design-for-test methodology for all IBM ASIC products. One of the challenges faced in servicing the original equipment manufacturer (OEM) market has been to provide a means for customers to easily insert LSSD elements into their designs.

The typical design methodology for circuits of more than 50000 gates uses logic synthesis, in which the designer begins with a register-transfer-level (RTL) description of the design, and uses a logic synthesis tool to map the design into a gate-level netlist.

The most common design style in use today, because of its ease in developing RTL descriptions and its support by logic synthesis tools, is *edge-triggered clocking*, where flip-flops are used as the sequential logic elements. Flip-flops are not suitable for LSSD-based testing, however, because they are not scannable (unless a multiplexor is added to the flip-flop's data port to provide scan access, adversely affecting the performance and chip area of the design), and they do not provide the race-free clock control that LSSD master–slave clocking provides for the test environment [7].

Because of the prevalence of edge-triggered designs in the industry, and until recently the absence of LSSD in the external market, a solution was needed that would allow a customer to design using edge-triggered flip-flops, yet produce an LSSD-compatible netlist.

The strategy to provide OEM customers the means to map edge-triggered designs into LSSD applies the Design Compiler and Test Compiler products of Synopsys [23, 24]. The IBM internal logic synthesis tool, BooleDozer [10], provides equivalent capabilities.

When the RTL design is initially mapped into gates by the Synopsys Design Compiler, the resulting netlist contains edge-triggered flip-flops (plus transparent latches, if any are used in the design). IBM ASICs provides *pseudo-cells* in its design libraries that support these functions. These pseudo-cells, however, are not part of the final netlist and are not manufacturable.

Pseudo-cells are characterized by *prefix characters* in the library cell name, which precede the name of the actual LSSD cell that will be eventually mapped in place of the pseudo-cell. The following is a list of the prefix types and their functions:

D_ A D flip-flop that maps to an edge-triggered LSSD shift register latch (also known as a D-mimic SRL).

D_F_ A D flip-flop that maps to an F_ pseudo-cell (a D-mimic temporary cell that is later remapped to the final LSSD implementation).

F_ A D-mimic pseudo-cell that is mapped to a combined master–slave LSSD SRL and a *clock splitter* that converts the edge-triggered clock into master and slave clocks.

L_ A transparent latch that is mapped to a master–slave SRL whose output is taken from the master latch instead of the slave latch.

L2_ A transparent latch that is mapped to a master–slave SRL whose output is taken from the slave latch, which is operated functionally in flush mode.

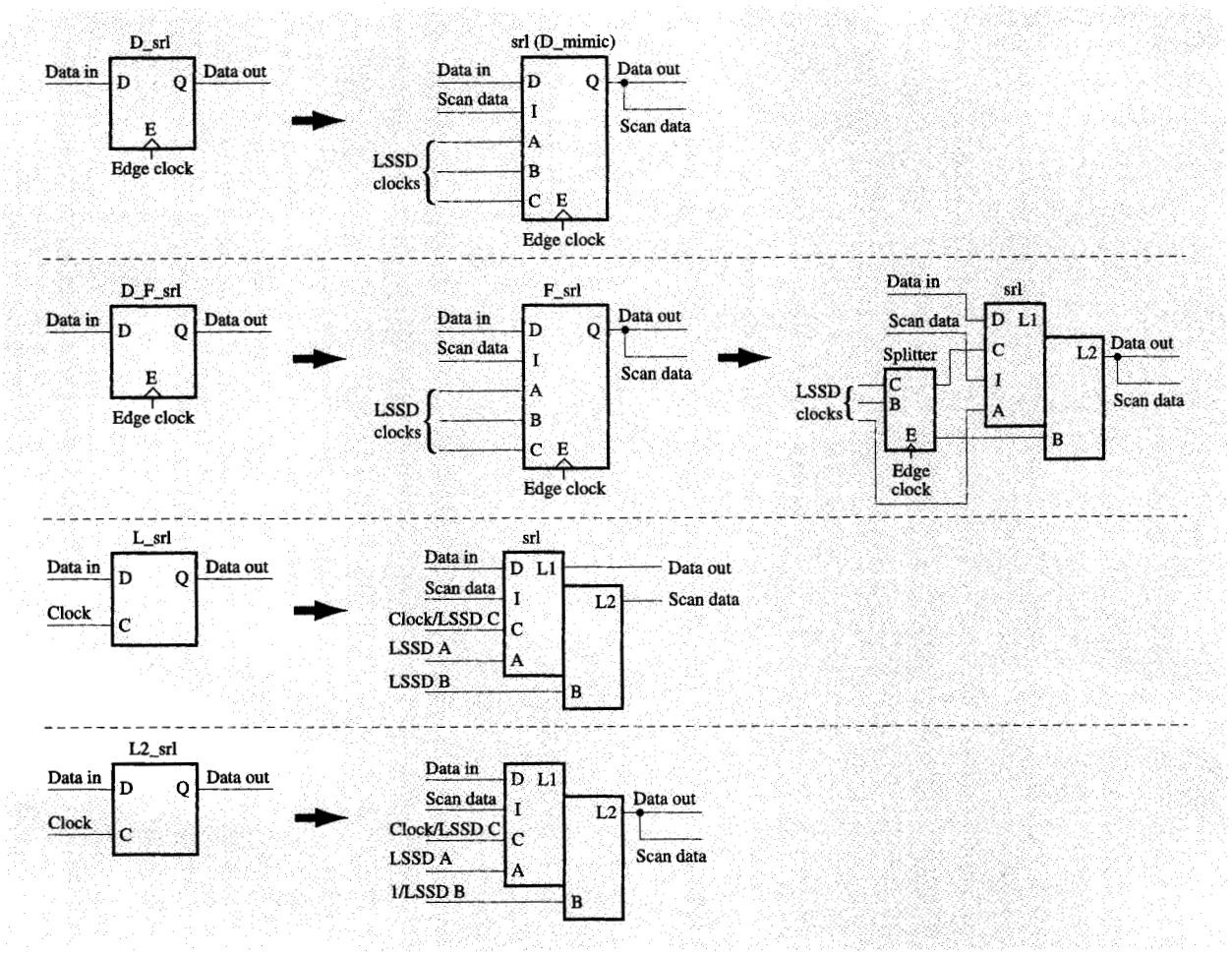

Figure 4

Mapping of flip-flops and latches to LSSD.

These pseudo-cells are mapped into LSSD SRLs as depicted in **Figure 4** and described below.

The methodology that maps from D_ and D_F_ pseudo-cells, using either the Synopsys Test Compiler or IBM BooleDozer, scans the netlist for these cells and replaces them with their target cells (either a true D-mimic SRL in the case of the D_ cell, or an F_ pseudo-cell in the case of a D_F_). This process additionally adds the LSSD clock (A, B, C) inputs, as well as the scan data (I) input. The new LSSD clock inputs for all SRLs are driven in parallel by new A, B, C inputs according to the customer's design. The scan data path becomes a serial connection: The output of each SRL is connected to the scan input (I) of the next SRL, creating a scan chain that connects to a scan-in input port and a scan-out output port of the customer's design.

The mapping of D_F_ pseudo-flip-flops to F_ pseudo-D-mimics often provides advantages in chip area over the D_ mapping [25]. The target cell of D_ mapping is an edge-triggered LSSD SRL, which contains an internal clock splitter that generates the master and slave clocks from the edge-triggered clock. In contrast, the subsequent mapping of the F_ cell creates a master–slave LSSD SRL, plus a clock splitter that can be *shared* among several SRLs, thus reducing the chip area overhead of LSSD. One clock splitter is typically shared by ten to twenty SRLs. The mapping from the F_ pseudo-cells is performed by an IBM BooleDozer operation; this operation is not provided by the Synopsys Test Compiler.

The algorithm that maps L_ and L2_ pseudo-cells is similar to D_ mapping. The difference is that the system clock is connected directly to the master clock input of the

SRL. For the L_ mapping, since the data output is taken from the master latch, transparent latch functionality is preserved; the scan output is taken from the slave latch. For the L2_ mapping, the B clock must be held high during functional operation, providing flush operation through the slave latch for system use, whereas the B clock is used for shift operation during test.

Using a combination of logic and test synthesis products, a customer can easily achieve LSSD compliance in an edge-triggered design paradigm. Development continues to achieve a more tightly integrated solution, providing the customer still greater ease in achieving LSSD compliance.

Integration of LSSD and industry-standard design for test

IBM has developed design-for-test (DFT) methodologies that integrate LSSD design structures with those design structures standardized by the IEEE Joint Test Action Group (JTAG), also known as the IEEE 1149.1 standard [26]. This methodology integrates the master–slave design practice, which results in a reliable, race-free test capability [7], with the edge-triggered design practices commonly used by OEM customers. This integration is achieved both for the customers' system design styles and for their use of the IEEE 1149.1 test standard for component, board, product, and field test. This integration is referred to here as a *co-compliant* design structure.

This test methodology combines LSSD and IEEE 1149.1 test requirements in each of the following areas:

- Test access to the component internals.
- Boundary scan capability for both component and intercomponent test.
- Internal scan capability.
- Logic built-in self-test (LBIST) [28].
- Array built-in self-test (ABIST) [12–14].

Co-compliant IBM ASICs provide the interfaces required of LSSD test as well as those required for access to the IEEE 1149.1 test access port controller (TAP controller). Test of a co-compliant component is viewed first as an LSSD-compliant component. Such a design, including the IEEE 1149.1 compliance logic, must be fully LSSD-compatible. The JTAG logic is fully tested, along with the customer's functional design, in IBM ASIC component manufacturing. However, when the LSSD test interface on the component is set for normal functional operation, the TAP controller is functional, allowing test in the customer's environment. LSSD inputs required for normal functional and IEEE 1149.1 operation are used as IEEE 1149.1 compliance-enable inputs, which are defined in the JTAG standard [26].

Boundary scan for intercomponent test is a primary objective of the IEEE 1149.1 standard. The TAP controller implements a set of instructions which provide the serial loading, unloading, and clocking of latches which control and observe each functional input, output, and bidirectional component pin. The logic at each pin that provides this function is known as an IEEE 1149.1 boundary cell. Boundary cells have been implemented that provide IEEE 1149.1 capability and support IBM ASIC manufacturing requirements [12–14] for

- Reduced-pin component test (scan access to all I/Os for full internal scan test using 64 or fewer test-access I/Os).
- Package test (component I/Os to package pins, or to scannable I/Os of other components in the case of multiple-chip modules).
- I/O wrap test (access to and from chip pins from boundary cell latches).

Internal scan capability is supported but not required by the IEEE 1149.1 standard. Only a subset of IBM ASIC customers who use the IEEE 1149.1 capability also choose to implement internal scan. However, customers have successfully reused the inherent scan capabilities of the internal LSSD-compatible latches, accessing these internal scan chains from the IEEE 1149.1 TAP controller. The TAP's edge-triggered test clock, under the control of the TAP's instructions, is converted into nonoverlapping master–slave clocks which shift the internal chain. The internal chains are reconfigured for component access at the IEEE 1149.1 test access port.

The use of built-in self-test (BIST) structures is a growing requirement, both for manufacturing component test and for customers' in-product use. In conjunction with co-compliant internal scan structures, a design structure was developed that incorporates self-test using multiple-input-signature registers and a shift register sequence generator [28], also known as the STUMPS architecture, for OEM customer designs. This implementation of STUMPS provides

- Customer control of STUMPS using the TAP controller for test of the component within the product, board, and component test environments.
- Optional use of STUMPS in the IBM ASIC manufacturing environment.
- Static and at-speed logic test via internal control of the master and slave clocks, derived from the product edge clock.

IBM ASIC RAMs contain an internal array BIST (ABIST) controller for manufacturing test (including at-speed) of the RAM memory cells. The above internal scan and BIST structures can additionally reuse the RAM's

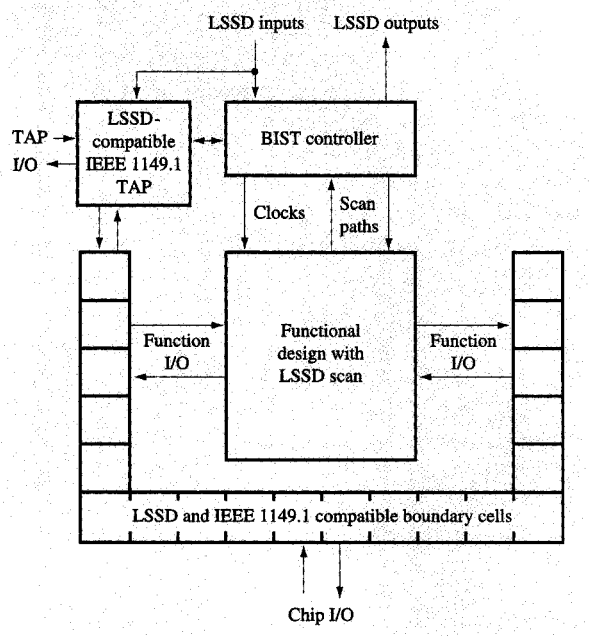

Figure 5

Co-compliant test structure.

internal ABIST capability to provide customer control of ABIST from the TAP controller for RAM test in the customer's environment.

The integrated capabilities described above (and depicted in **Figure 5**) are being increasingly automated by the IBM ASIC design center. At present, these capabilities are incorporated through a combination of internal and external logic synthesis tools, with some manual design and connection steps. Work continues to automate these capabilities fully.

Front-end processing

Front-end processing is a step in the IBM ASIC methodology that is used, after logic synthesis, for the design and optimization of technology-dependent clock networks, test structures, and a set of prerequisite foundry-specific circuits [12–14]. Traditional design automation applications have been found to be inadequate for these tasks, and manual entry is prohibitive from a design productivity and verification standpoint.

Front-end processing begins with a design that may contain

- Idealized clocks (i.e., no clock-repowering networks), where a clock simply fans out to each target latch.
- Partial or no design-for-test structures.
- Incomplete incorporation of foundry-support circuits (such as I/O driver and receiver inhibit signals).

At the completion of front-end processing, the design is fully implemented, from a logical standpoint, as a manufacturable component that contains all required functional and test circuits.

Front-end processing has traditionally been run by the IBM ASIC design center, but it has recently been packaged for customer use, enabling the customer to verify that the final logical design meets foundry sign-off criteria. Prior to this, customer pre-layout sign-off had been done on an incomplete, abstracted version of the design, resulting in inaccuracies and thus additional iteration within the IBM ASIC design center.

Front-end processing uses two IBM software applications. ClockPro™ [29] is used to determine the requirements of the clocking network logic. BooleDozer Lite, a subset of the IBM logic synthesis tool [10], provides for user-defined logic transformation programs in a script language. This is used to perform a variety of logic editing and analysis functions, ranging from simple net manipulation to complex network construction. These accomplish specific postsynthesis logic insertion and optimization tasks.

ClockPro is an IBM clock planning tool that uses information about the design's clock connections and performance targets and the customer's clock-tree component preferences to calculate families of alternative clock networks. These are sorted by latency and presented with data describing cell utilization, the number of clock buffer levels, fan-out per level, and estimated capacitance at each level. A choice can then be made on the basis of speed, size, structure, or power. The resulting selection is then used as input to BooleDozer Lite for insertion into the design.

Front-end processing performs several standard tasks for a typical design, including clock planning and construction, test insertion, and design finishing. Other customized tasks are occasionally required. Standard or custom tasks are always reviewed with the customer at a preliminary requirements analysis session, to determine the exact objectives and prerequisites before the design is synthesized. New transformations are developed by the IBM ASIC design center as required.

The supported netlist interfaces into IBM ASIC design processing are EDIF 2.0.0 and VIM. For users of Synopsys synthesis tools, IBM provides a script to write the EDIF in the format compatible with the requirements of front-end processing.

- *Clock planning and construction*

The initial design normally contains one or more system clocks driving a large number of flip-flops and latches.

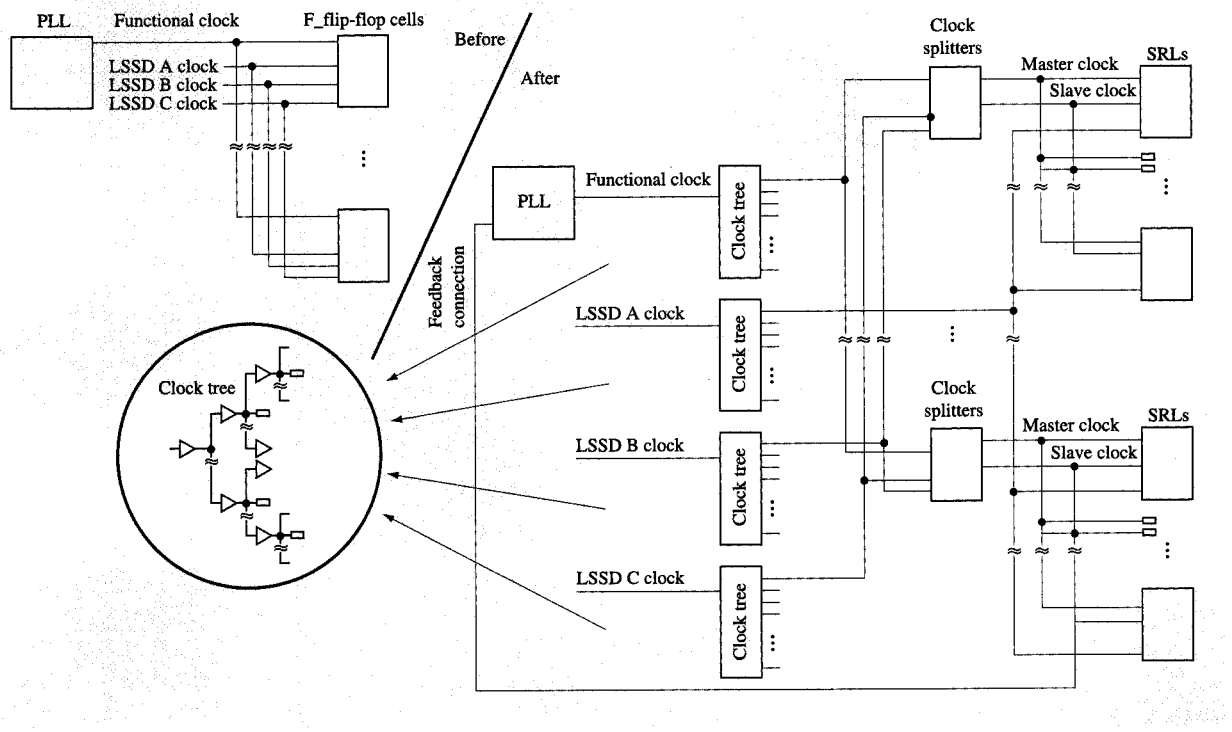

Figure 6

Clock planning and construction.

These are incorporated in the customer's designs as logical pseudo-cells, containing only the data functions and clocks, without yet containing the LSSD functions.

The target implementation is a serially repowered system clock tree [30] using clock driver cells that feed a clock-splitter cell, which in turn drives LSSD shift-register latches. The clock splitter creates two level-sensitive clocks that control the system master and slave clocks of each SRL. The clock splitter is treated as the leaf driver in ClockPro when calculating the clock network. Also, ClockPro inserts terminator cells (cells that capacitively load a net) on branches of the tree to guarantee logical balance and to provide a means of tuning the load, as needed, on any branch.

Repowering networks can also be inserted for nonsystem clocks. Normally, since performance is not as critical for these clocks, ClockPro can be given relaxed performance parameters, and the insertion of terminators can be nullified, thus saving on chip area and wiring congestion.

Figure 6 depicts how an idealized clock is transformed into the required clock tree and clock-splitter logic.

- *Test insertion*

If test logic has not been inserted by the customer, front-end processing can be used to accomplish this requirement. Latch pseudo-cells are replaced by LSSD SRLs, the test clocks are connected to the SRLs, and the SRLs are serially connected to form one or more scan data paths [7]. These scan paths are balanced by length to minimize the data-load time for the manufacturing tester [12–14].

With front-end processing, multiple LSSD clocks can be accommodated. This is sometimes needed to address LBIST [28] configurations and to achieve race-free LSSD operation. Gated clocks can also be used with front-end processing, and are often required when test clocks are used functionally, such as for internal system scan or for low-power applications.

- *RI/DI chaining*

Receiver-inhibit and driver-inhibit (RI/DI) chaining involves the daisy-chain connection of up to four driver-inhibit signals and one receiver-inhibit signal using specific ports on the I/O buffer cells, resulting in the insertion of

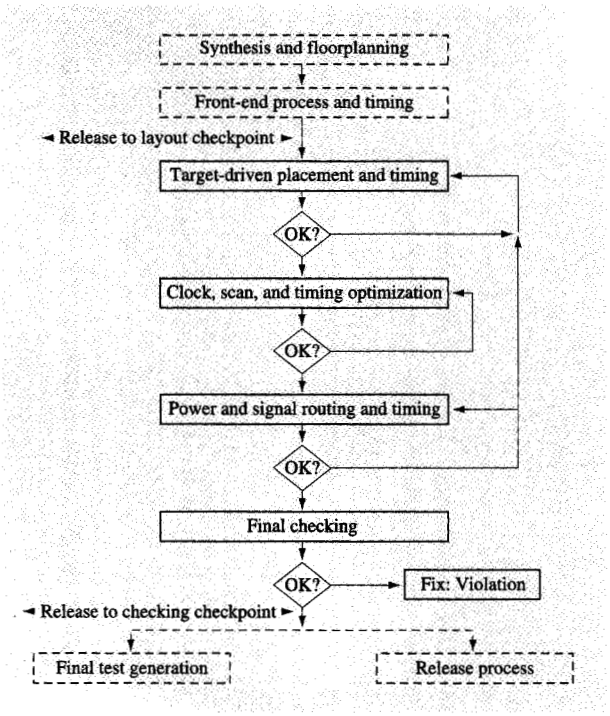

Figure 7

Flowchart of physical design methodology.

up to five inhibit chains [12–14]. A typical design has the RI/DI inputs of the I/O cells tied to a noncontrolling value (for functional simulation) upon entry of the design into front-end processing. RI/DI chaining removes any existing connections to the RI/DI ports and connects them into the appropriate chains.

- *Tie cell repowering*

Often, signals that provide a constant logical one or zero value can present layout problems for technologies with explicit tie-up or tie-down cells, in that multiple usages of a constant value can result in tie cells with high fan-out and the use of a significant number of wiring channels on the chip. Tie-cell repowering addresses this problem by adding a tie cell for each pin fed by the original net. The added cells are then connected one-to-one to those pins. During physical design, the tie cell is placed close to the receiving cell so that the corresponding wire length is minimal.

Physical design

Customers designing in the IBM CMOS 4 technologies must provide a flattened netlist to the design center for layout using the IBM mainframe layout tool. Flattening of the netlist can be done either by the customer or by an IBM design center. Customers designing in the IBM CMOS 5 technologies can use either a hierarchical or a flat physical design process, using the IBM ChipBench™ [11] layout tool. The hierarchical design process is generally good for large designs, whereas small chips are often processed more efficiently using the flat methodology. If it is determined that the physical design of a CMOS 5 ASIC is to be flat, a flattened netlist is provided to the IBM design center.

If the physical design of the chip is to be done hierarchically, the customer provides a two-level hierarchical netlist. This netlist contains a top level that calls out second-level entities or blocks, which in turn call out individual cells or instances of register arrays or RAMs. In the hierarchical methodology, each of the blocks is individually placed and routed. The I/Os for the set of blocks are interconnected and are connected to the chip I/Os in a global wiring pass. The flow of the physical design process is shown in **Figure 7**.

If the physical design of the chip is to be flat, only one set of timing-assertion files [5] for the chip is required. If the physical design is to be hierarchical, and if the customer requires timing on each of the individual blocks, the timing assertions must be provided for each of the block entities as well as for the chip level. The customer provides any additional timing assertions needed for static timing analysis of the post-PD design.

In a hierarchical environment, if timing-driven design is used, each of the blocks is placed and routed separately with respect to the timing assertions provided. A second pass through wiring connects the blocks together with respect to the chip-level timing assertions.

Clock optimization and scan optimization processes are applied to improve timing, wiring, and clock skew. Clock optimization modifies the organization of the clock-tree buffers and the clock splitters according to the layout of the latches and the customer's requirements for slew and latency. Scan optimization reorders the scan chains to minimize wiring congestion.

A high-performance CMOS 5 design may have to iterate within the steps of the place-and-route process and timing analysis in order to achieve the timing targets of the design. Prior to the initial placement of logic cells, EinsTimer is used to produce capacitance targets for nets identified in the worst-case timing paths. The capacitance targets guide ChipBench during initial placement. Prior to clock optimization, static timing analysis is applied again, using idealized clocks, to verify the placement; the IBM ChipEdit [31] tool can be used at this time to manually change the initial placement to improve timing. After running clock optimization, the option exists, again using ChipEdit, to manually prewire worst-case nets prior to automatic wire routing. ChipBench is then used for automatic routing, followed

again by static timing analysis using actual clock-tree characteristics instead of idealized clocks. ChipEdit can be reapplied to fix any remaining timing problems. This process is generally effective at eliminating the need to recycle through the physical design methodology to meet performance objectives.

Embedded "system building blocks" create system-on-a-chip technology

System-on-a-chip, or "system building block," technology makes high-level functions, such as microprocessor cores, signal-processing compression functions, and other system-oriented functions available to ASIC chip designers as part of the ASIC design process. The use of core macros in the past required custom chip design by a dedicated design team at the silicon vendor. The system building block functions, or "cores," are constructed as predesigned building blocks, to be integrated as easily as gates and latches. This idea is not evolutionary, as is going from larger to smaller die sizes or increasing processing speeds; it is completely revolutionizing the way in which we design our chips by bringing core-plus-ASIC design to the designer's desktop [32].

There are two types of core macro functions, soft and hard. Soft cores are delivered to the ASIC designer as a hardware design language (HDL) design. The design data for the soft core can be provided at various levels: premapped to the ASIC vendor's technology as a gate-level netlist, or as synthesizable behavioral or register-transfer-level code.

Once incorporated into the ASIC design through synthesis or netlist "stitching," the soft-core logic cannot be distinguished from the rest of the ASIC design, and therefore presents minimal methodology issues. Gate-level timing, simulation, testability analysis, and floorplanning of the soft core are accomplished using the models for the standard ASIC library elements.

Soft-core macros do not have a predefined layout. Soft-core gates are placed and routed as part of the ASIC design. Because the layout is not predefined, it is important that macro functions implemented as soft cores do not have performance requirements that would drive custom layout or custom library cells.

Hard cores are macro functions that have a predefined layout. The core is modeled as a single library element, or "black box," much the same as a RAM or ROM macro. Macro functions, such as microprocessors, are implemented as hard cores for several reasons, predominantly to preserve a custom layout of the circuit that is required to meet the desired performance, and to accommodate an ASIC vendor who chooses not to reveal the detailed design of the macro.

Because a hard core is modeled as a single black box rather than as the synthesizable code of a soft core, the internal description of the macro function is completely "hidden" from the customer, providing intellectual property protection for the ASIC vendor. Development of these black-box models for a complex macro function, with the accuracy required for use by the ASIC designer, often requires significant effort and invention on the part of the ASIC vendor.

Both hard- and soft-core macros provide valuable predesigned function for use by the core-plus-ASIC designer.

However, this new core-plus-ASIC technology has created a new generation of integration challenges. Integrating cores with various interface requirements often requires significant "glue logic." Because these interfaces are now buried in the ASIC, system verification requires better simulation models than have traditionally been available for standard products. Larger gate counts drive a need for better simulation tools and new approaches to system verification. To support manufacturing test of the ASIC, innovative solutions are required.

To solve these problems, many functions are marketed as synthesizable "soft macros." These macros are easily integrated with the designer's logic and can be customized to meet individual design requirements.

The disadvantage of soft macros is that speed and density are limited by the characteristics of the ASIC standard cell library in which they are implemented. Standard products often use custom circuit design to push the speed and density of the technology beyond the limits of a standard cell implementation. Corresponding core products can take advantage of this by using a "hard macro" approach. Because of the methodology challenges associated with developing and testing hard cores, ASIC designs using hard macros are usually designed by the ASIC vendor to customer requirements. A few ASIC vendors have developed the capability to support the next level of design methodology, in which customers design the ASIC logic around the hard macro, and the ASIC vendor integrates the hard macro with this logic before sign-off. In either case, the involvement of the ASIC vendor in the design process limits availability of this technology to only the highest-volume projects. Today's challenge is to provide tools and methodologies allowing design, integration, and sign-off verification of the core-plus-ASIC chip at the ASIC designer's desktop, making cores as easy to use as the traditional standard cell library.

The current trend for core libraries is to incorporate functions originally designed as standard product chips. Such functions may incorporate additional features not required by the application, or may optimize the interface around package technology limitations that do not apply to a core implementation. ASIC designers often must customize soft macros to achieve an optimal design. This customization can require up to 30% redesign of the

function. The interfacing of multiple cores may also require significant glue logic.

As core libraries evolve and incorporate new functions originating as core products, interfaces will develop that are optimized for performance, routability, and flexibility, unfettered by package pin-count limitations. Core designs will become modular, with plug-in features that can be included or deleted from the design, rather than incorporating a wide range of features into a monolithic design. Industry standards will develop for on-chip bus architectures, allowing the integration of cores with little or no additional glue logic. Development of modular cores and on-chip bus standards makes it possible for tools to emerge that can automatically "build" the netlist for an overall system, using parameters entered by the ASIC designer.

Simulation of an ASIC containing a core macro requires a simulation model of the core to permit the designer to determine whether the ASIC logic that communicates with the core is functioning correctly. To minimize time to market, hardware and software designs for a product invariably proceed in parallel. The types of simulation models available for a processor core can affect the efficiency of this parallel design process. A range of simulation models supporting the entire design process is preferred.

During the synthesis step, the behavioral HDL for the ASIC and any behavioral soft-macro cores are mapped to technology-specific gates. Any logic originating from a soft macro is then indistinguishable from the ASIC designer's logic. A hard-macro core, however, passes through the synthesis step unchanged. It continues to be represented as a single library element and is included as such in the netlist output by the synthesis tool.

Testing of core-plus-ASIC designs presents challenges that are added to those caused by traditional ASIC methods. These include the need for core access and isolation during scan-based testing and, in some cases, the need to apply core functional patterns. Test patterns from different source tools and for different technology libraries must be merged and applied during core-plus-ASIC testing.

After synthesis and DFT checking, the core-plus-ASIC design is functionally verified at the gate level. At this stage, simulation using an HDL full-timing model for the hard-macro cores is appropriate, although the bus-functional model is still an option.

Intellectual property protection is a concern for the ASIC vendor providing the detailed full-timing simulation model of the core. The model must be completely accurate in terms of the behavior and timing characteristics of the core, but it should not be described, for example, in a way that exposes the actual design of a microprocessor. The preferred method is to use an encrypted model originating from a detailed netlist of the core macro, which has been compiled so that it is "simulator-independent," or able to work in a variety of VHDL, Verilog, and other simulators. The full-timing model for the hard-macro core must also be compatible with the ASIC library in order to support a consistent method of post-layout timing back-annotation. A standardized format, such as standard delay file (SDF), is often used for back-annotation of ASIC post-layout timing.

For a hard-core macro, a static timing analysis model is required that models delays and specifies timing checks at the interface to the core. Internal timing assertions are built into this model. For a soft macro, such information must be provided as timing assertions or constraints to be incorporated by the customer in the chip-level timing analysis. The static timing analysis tool must be able to handle and correctly interpret both the abstracted timing model for the core and the timing models for the base library elements.

Because a large hard-core macro is likely to occupy a significant percentage of the chip's area, floorplanning becomes a requirement in order to achieve overall chip timing and ensure wirability of the final design. Floorplanning models that provide both timing and blockage information for the core are required to help the designer make layout trade-offs.

Core-plus-ASIC technology is being used to drive the next step of evolution for a broad spectrum of electronic products. A paradigm shift in the way designs are conceived and created by a design team has already begun. By the year 2000, ASIC vendors will be selected on the basis of the availability of ASIC cores and of tool support for those cores. The emergence of bus standards optimized for on-chip buses will eliminate performance limitations imposed by package pin counts and will enable automated generation of system designs. New methodologies are emerging that will make possible core-plus-ASIC chip design at the designer's desktop and make this technology as easy to use as a standard cell library. As a result, every ASIC designer will have access to the vast intellectual property assets of the ASIC vendor.

The design methodology for core-plus-ASIC technology is summarized in **Figures 8** and **9**.

Conclusion

As chip technologies continue to provide for increasing circuit densities and performance, the ASIC design methodology is increasingly challenged to provide product designers with the means to exploit these technological advances. The capabilities described in this paper are being used today by IBM ASICs to give designers, both within IBM and throughout the industry at large, the ability to use leading-edge IBM CMOS technologies in the

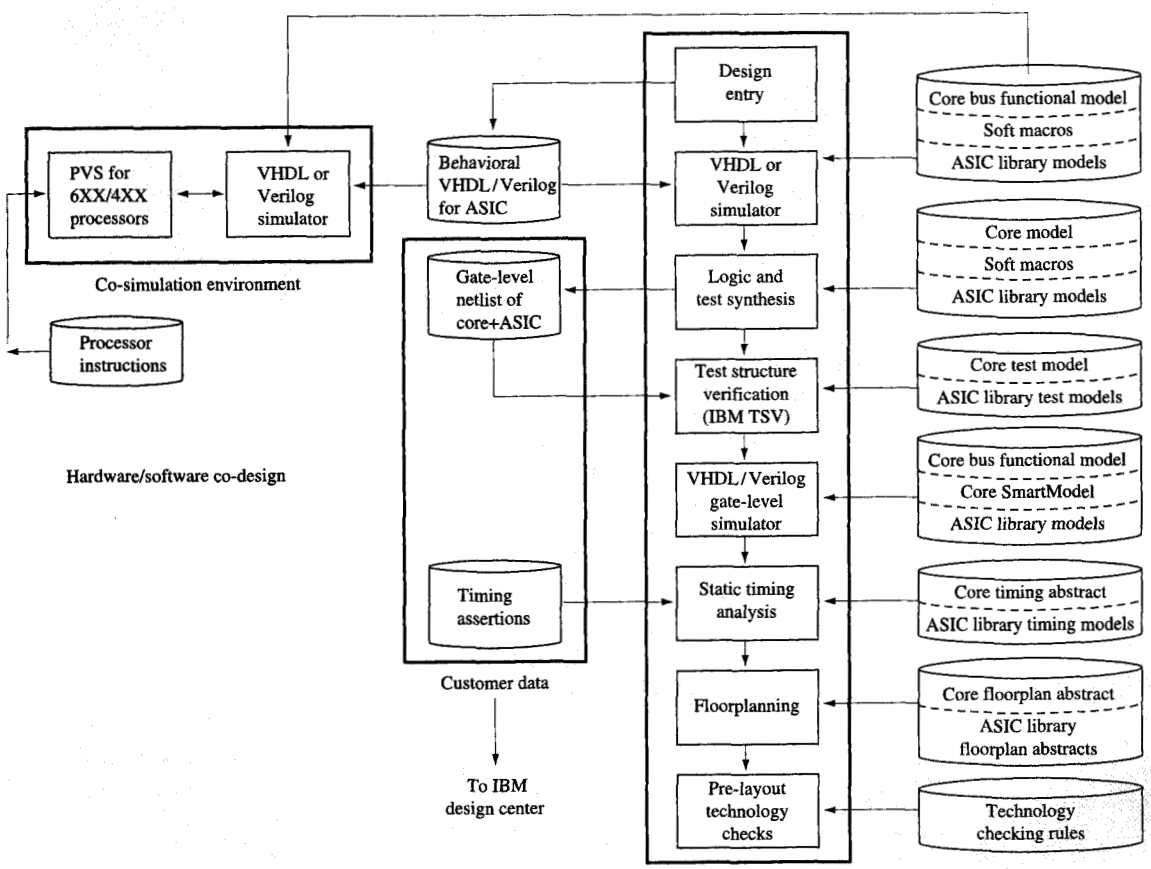

Figure 8

Flowchart of front-end process for embedded-cores methodology.

development of leading-edge products. These capabilities are being extended and refined to meet the challenges posed by further advances in chip technology.

Additionally, new methodologies are required to meet these ever-increasing demands. The capabilities of cycle simulation, hardware accelerators, and emulators must be coupled with advances in formal verification, high-level synthesis, and system-on-a-chip architectures to provide designers with the increased logic design productivity required by greater chip densities and shorter time-to-market requirements. The ability to estimate chip power consumption, more accurately and much earlier in the design cycle, must be provided to ASIC designers. Finally, the overall product design flow must become increasingly integrated, as the electrical and physical factors involved in chip realization affect logic design and system architecture to a much greater extent.

Acknowledgments

The authors wish to acknowledge the contributions of the following individuals to the work described in this paper: Steven F. Oakland, for contributions to the LSSD mapping methodology and design of an LSSD-compatible IEEE 1149.1 boundary-scan structure; James G. Swift, for contributions to the front-end design methodology; Michael T. Trick, for contributions to the physical design methodology; Richard F. Paul, for contributions to the timing-driven design methodologies; Jeannie H. Panner, for contributions to the physical design, CMOS checking, and release methodologies; Julie Druckerman, Ram Kelkar, Ted Lattrell, and Don Pierson, for technical and editorial contributions to the embedded core methodology; and Michael D. O'Neill, for contribution to the overall methodology.

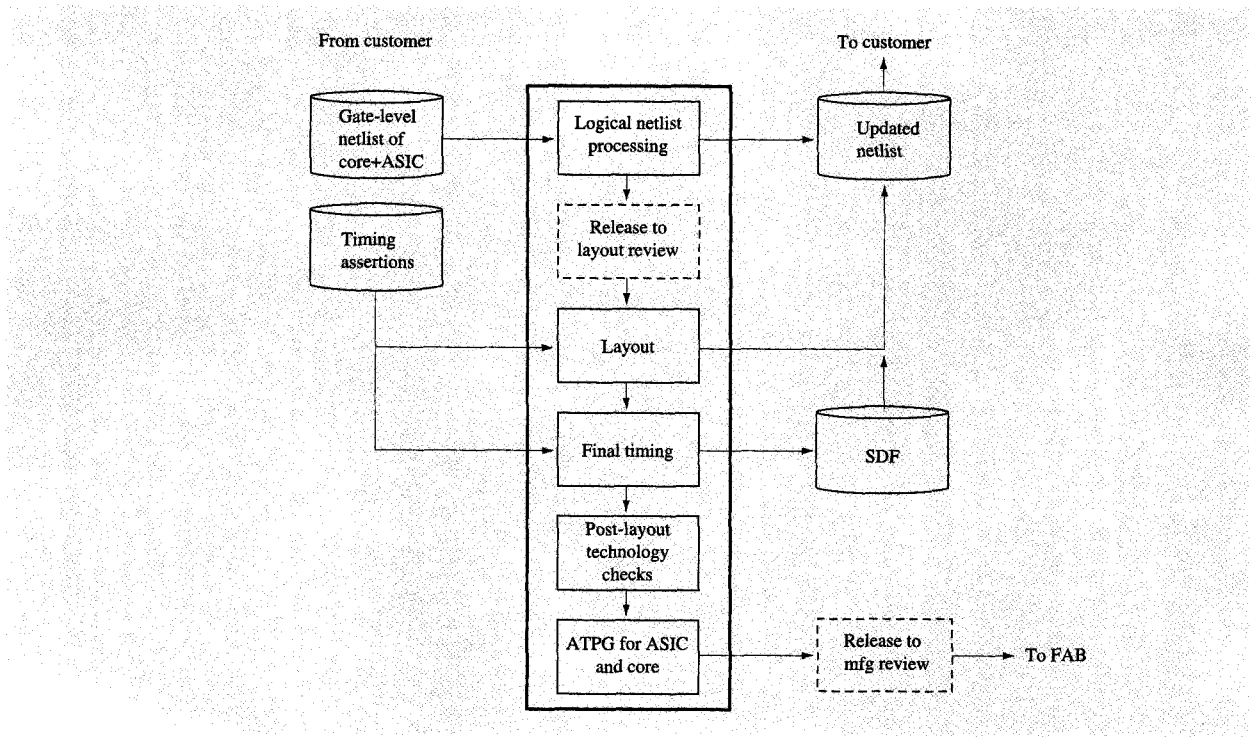

Figure 9

Flowchart of layout process for embedded-cores methodology.

References

1. *Wizard Users' Guide, Version 1.5*, IBM Corporation, Hopewell Junction, NY 12533, 1995.
2. *Design Entry: Composer Reference Manual, Version 4.3.4*, Cadence Design Systems, Inc., San Jose, CA 95134, 1995.
3. *Verilog-XL Reference Manual, Version 2.2*, Cadence Design Systems, Inc., San Jose, CA 95134, 1995.
4. *Standard Delay Format Specification, Version 2.1*, Open Verilog International, Las Gatos, CA 95032, February 1994 (correction, July 1994).
5. *EinsTimer Users' Guide and Language Reference, Version 1.3*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
6. "Linking to Physical Design Tools," *Design Compiler Family Reference*, Synopsys, Inc., Mountain View, CA 94043, 1995.
7. "Level-Sensitive Scan Design: Concepts and Applications," *TestBench Users' Guide, Version 2.1*, IBM Microelectronics Division, Endicott, NY 13760, 1995.
8. *Chrysalis Design VERIFYer Users' Guide, Version 2.06*, Chrysalis Symbolic Design, Inc., Andover, MA 01810, 1995.
9. "Comparing Designs: Boolean Equivalence Checking," *BooleDozer Synthesis Users' Guide, Version 1.4*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
10. *BooleDozer Synthesis Users' Guide, Version 1.4*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
11. *ChipBench 1.2 Users' Guide*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
12. *CMOS 4LP ASIC Products Databook*, IBM Microelectronics Division, Essex Junction, VT 05452, 1993.
13. *CMOS 5L ASIC Products Databook*, IBM Microelectronics Division, Essex Junction, VT 05452, 1995.
14. *CMOS 5S ASIC Products Databook*, IBM Microelectronics Division, Essex Junction, VT 05452, 1995.
15. *VITAL (VHDL Initiative Toward ASIC Libraries), Version 2.2b*, IEEE, Inc., Piscataway, NJ 08855, 1995.

16. *Draft of Procedural Interface and DCL Language*, CAD Framework Initiative, Inc., Austin, TX 78759, 1995.
17. *Design Compiler Reference Manual, Version 3.1a*, Synopsys, Inc., Mountain View, CA 94043, 1994.
18. *Library Compiler Reference Manual, Version 3.1a*, Synopsys, Inc., Mountain View, CA 94043, 1994.
19. *Hierarchical Design Planner Users' Guide, Version 2.1*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
20. *Design Planner Reference, Version 3.1.2*, High Level Design Systems, Inc., Santa Clara, CA 95054, 1995.
21. *Preview Cell3 Ensemble 4.3*, Cadence Design Systems, Inc., San Jose, CA 95134, 1994.
22. *DesignWare Users' Guide, Version 3.1a*, Synopsys, Inc., Mountain View, CA 94043, 1994.
23. *Test Compiler Reference Manual, Version 3.1a*, Synopsys, Inc., Mountain View, CA 94043, 1994.
24. L. Pickup, "Using IBM's LSSD Latches with Synopsys," *ASIC Application Note*, IBM Microelectronics Division, Essex Junction, VT 05452, 1994.
25. D. Lackey, "Benefits of LSSD," *Application Note*, IBM Microelectronics Division, Essex Junction, VT 05452, 1994.
26. "IEEE Standard Test Access Port and Boundary-Scan Architecture," *IEEE Standard 1149.1-1990*, IEEE, Inc., Piscataway, NJ 08855.
27. S. Oakland, *IEEE 1149.1 Boundary-Scan in IBM CMOS 5L ASICs*, IBM Microelectronics Division, Essex Junction, VT 05452, 1995.
28. Paul Bardell and William McAnney, "Self-Testing of Multiple Chip Modules," *Proceedings of the International Test Conference*, 1982, reprinted in *1970–1994 25th Anniversary Compendium of Papers from the International Test Conference*, Washington, DC, 1994, pp. 535–539.
29. *ChipOpt Users' Guide*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
30. Vivek Chickermane, Bernd Koenemann, Thomas Guzowski, T. W. Williams, Andrew Sullivan, and Steven Oakland, "DFT: Test Synthesis and Beyond," *Proceedings of the International Test Conference, Test Synthesis Seminar, TS Paper 3.3*, Washington, DC, 1994, pp. 1–7.
31. *ChipEdit Users' Guide*, IBM Microelectronics Division, Hopewell Junction, NY 12533, 1995.
32. Julie Druckerman, Ram Kelkar, Ted Lattrell, Don Pierson, Ann Rincon, and David Stauffer, "The Evolution of Core Plus ASIC Methodology," *Integrated System Design* **7**, No. 77, 30–41 (November 1995).

Received October 27, 1995; accepted for publication April 5, 1996

James J. Engel *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (JENGEL at BTV; jengel@vnet.ibm.com).* Mr. Engel is a staff engineer in the IBM ASIC Products group, responsible for timing methodologies. He joined IBM in Poughkeepsie, New York, in 1983, and holds a B.S.E.E. degree from the New York Institute of Technology.

Thomas S. Guzowski *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (GUZOWSKI at BTV; tsg@vnet.ibm.com).* Mr. Guzowski is a senior engineer in the IBM ASIC Products group, responsible for front-end design automation and design center methodologies. He joined IBM in Essex Junction in 1978, and worked in IBM Tucson, Arizona, from 1985 to 1992 before returning to Essex Junction. Mr. Guzowski holds a B.S. degree in physics from Indiana University of Pennsylvania.

Anderson Hunt *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (HUNT at BTV; buco_fan@vnet.ibm.com).* Mr. Hunt is an advisory engineer in the IBM ASIC Products group, responsible for datapath methodologies. He received a B.S.C.S. degree from the University of Vermont in 1979, joining IBM in Kingston, New York, in 1984.

David E. Lackey *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (LACKEY at BTVVMOFS; david_lackey@vnet.ibm.com).* Mr. Lackey is a senior engineer in the IBM ASIC Products group, responsible for design for test, formal verification, and overall ASIC methodologies. He joined IBM in Poughkeepsie, New York, in 1978. Mr. Lackey holds a B.S.E.E. degree from Rensselaer Polytechnic Institute and an M.S.C.E. degree from Syracuse University. He is a member of IEEE and Eta Kappa Nu.

Lansing D. Pickup *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (LPICKUP at BTV; lpickup@vnet.ibm.com).* Mr. Pickup is a staff engineer in the IBM ASIC Products group, responsible for simulation and synthesis methodologies. He joined IBM in Essex Junction in 1988, and holds a B.S.E.E. degree from Clarkson University. Mr. Pickup is a member of Tau Beta Pi and Eta Kappa Nu.

Robert A. Proctor *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (PROCTOR at BTVVMOFS; rproctor@vnet.ibm.com).* Mr. Proctor is an advisory engineer in the IBM ASIC Products group, responsible for floorplanning methodologies. He joined IBM in East Fishkill, New York, receiving a bachelor's degree in electrical engineering from Newark College of Engineering in 1971. Mr. Proctor has also worked at the IBM sites in Poughkeepsie and Kingston, New York.

Karla Reynolds *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (KREYNOLD at BTVVMOFS; kreynold@vnet.ibm.com).* Ms. Reynolds is an advisory engineer in the ASIC Products group, responsible for

logic synthesis methodologies. She has nineteen years of EDA tool development experience at IBM Essex Junction, and holds a B.S. degree in mathematics from the University of Maine.

Ann Marie Rincon *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (RINCON at BTVVMOFS; rincon@vnet.ibm.com).* Ms. Rincon is a senior engineer in the IBM ASIC Products group, responsible for core-plus-ASIC and overall ASIC methodologies. She joined IBM in Essex Junction in 1981, and holds a B.S. degree in mathematics from St. Joseph's College, Indiana.

David R. Stauffer *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (STAUFFER at BTVLABVM; dstauffer@vnet.ibm.com).* Mr. Stauffer is an advisory engineer in the IBM ASIC Products group, responsible for core-plus-ASIC design. He joined IBM in 1984. Mr. Stauffer holds a B.S.E.E. degree from Pennsylvania State University and an M.S.E.E. degree from the University of Houston.