Development of ASIC Chip-set for High-end Network Processing Application – a Case Study

Sanjeev Patel
Wipro Technologies
72, Electronics City
Bangalore 561229 India
sanjeev.patel@wipro.com

Abstract

Choosing the right methodology is a significant step towards successful VLSI designs. Traditional methodologies and tools are no longer adequate to handle large and complex designs. This paper presents a novel design methodology for complex deep-submicron designs, using a case study of the development of a high-end network processing ASIC chip-set. The paper focuses on the synergetic use of the "dual design verification approach", along with static verification methods in achieving defect free silicon. It also discusses the techniques employed for achieving faster and less-iterative timing closure.

1. Introduction

With shrinking geometries, more functionality can be integrated on-chip. While higher integration enhances performance and helps reduce system costs, it also presents higher challenges. As design size grows, verification complexity grows in non-linear proportions. Achieving timing closure is a significant challenge at deep-submicron geometries due to the dominance of routing delays over gate delays and increased on-chip coupling effects. The design methodology must be robust enough to ensure defect-free functionality without compromising time-to-market.

Traditional VLSI design methodologies do not address these issues well. Verification environment and test cases use hardware description language, which lacks verification-friendly constructs, leading to longer development time for large designs. Gate level simulations using back-annotated timing, used for checking functional and timing integrity, are too time-consuming. Conventional EDA tools lack the sophistication necessary to model deep-submicron effects, leading to multiple iterations for timing closure.

Newer approaches are being adopted to deal with these shortcomings. Higher levels of abstraction are used to speed up test development time. Static methods like static timing analysis and formal verification are employed for timing and functional integrity checks. Current generation physical design methods use early floor-planning, physical synthesis and timing driven placement and routing for faster timing convergence.

Our methodology combines a new functional verification approach with the static verification methods to achieve faster functional and timing validation. The new approach, called *dual design verification*, uses two representations of a design throughout functional verification. One is a behavioral model in 'C', developed during high-level design phase. The other is an RTL model. The 'C' model is used as a reference to check the RTL representation. It is used not only in unit level, ASIC level, and chip-set level simulations, but also in software testing. This reuse saves considerable time. Once the design is functionally verified, static methods i.e. formal verification and static timing analysis, provide functional and timing consistency checks throughout the physical design cycle. Unique approaches adopted in physical design help achieve faster timing closure.

Design goals are described in section 2, followed by discussion on issues and challenges in realizing these goals in section 3. An overview of our design methodology is provided in section 4. Key features of our methodology, namely *dual design verification* approach, static methods and timing closure techniques are explained in section 5, 6 and 7 respectively. Section 8 concludes the paper with key findings.

2. Design Goals

The chip-set implements network processing functions upto OC-192 speeds for high-end edge routing

applications. Flexibility in supporting newer protocols is achieved through programmable data inspection and formatting engines. The chip-set supports UTOPIA, POS and CSIX interface standards to enable seamless connection to switch fabric and a variety of physical interface devices. Efficient buffer management and connection management functions support millions of simultaneous connections.

These functional requirements drive the ASIC design goals. Three ASICs are needed to implement the functionality in a practical, scalable way. The ASIC core and inter-ASIC interfaces operate at 200 MHz while the selective interfaces operate at 100 MHz. PLLs (Phase Locked Loop) are used to generate in-phase 200 MHz and 100 MHz clocks. We targeted the design to IBM 0.16 u process technology. Area-array based IO implementation and flip-chip CCGA (Ceramic Column Grid Array) packaging [2][5] is chosen to accommodate the 700+ user I/Os. Design for test methodology is LSSD (Level Sensitive Scan Design) scan for logic core, MBIST (Memory Built-in Self Test) for internal RAMs and IEEE 1149.1 JTAG boundary scan for the I/Os.

3. Key Design Challenges

Early identification of key issues is an important step in the design methodology since it allows incorporating checkpoints and strategies for the further work.

The variety of configuration options needed in network processing calls for a sophisticated high-speed on-chip system bus for which, issues with heavy loading and fanout of common signals need to be addressed. Debug features must be provided on-chip to help debug such highly configurable designs. Wide, on-chip buses operating at high frequencies and large number of on-chip as well as off-chip memories place a significant constraint on design partitioning and make timing closure tougher unless design guidelines are well established upfront.

Verification efforts for multimillion gate ASICs consume significant portion of the development cycle. Development of reusable components and use of higher levels of abstraction for test vector generation becomes a key challenge. Achieving high functional coverage on complex logic such as multiple instantiated microcode programmable data inspection and formatting engines would require high-level abstraction, randomization and automation in verification.

Large size, density and high frequency of operation of these deep-submicron designs present enormous challenges to physical design. Skew targets as low as 150 ps, requirement to keep the two on-chip clocks in phase, source synchronous clocking on inter-ASIC interfaces, large on-chip memories occupying majority of the die area, and large number of user I/Os switching at high-speed were the key challenges to clock design, routability and timing closure.

The design methodology discussed in section 4 addresses these challenges without any compromise.

4. Methodology Overview

A design methodology must put together a right mix of tools, techniques and best practices to enable complex deep-submicron designs, while keeping in mind the demanding time frames. Our methodology balances traditional approaches with newer ones, eliminates wasteful processes and implements consistency checks throughout development cycle. This is made possible with unification of design's reference model, *dual design verification* that reuses design abstraction, use of static methods that eliminate costly gate level simulations, and novel techniques in timing closure.

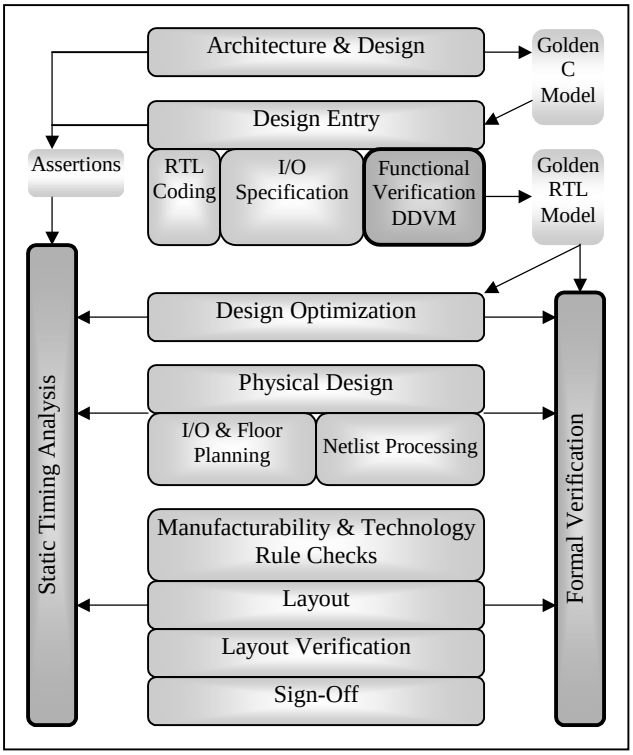

Figure 1 Design Methodology

Figure 1 shows the process used for our designs. The key phases in the process are described in brief below.

Design entry involves RTL coding, simulation and I/O specification. RTL is purified through linting and verified using the *Dual Design Verification (DDVM)* approach. I/O specification is created considering I/O interface requirements and SSO issues. [1]

Design optimization starts with synthesis for performance optimization. Bottom up approach allows easy integration of design changes. Top-level logic, like

PLLs, and boundary scan logic, is instantiated into the core netlist through a step called *ASIC top-level insertion*, after which the netlist is flattened for further processing.

I/O specification of the design is used for *I/O planning*, along with the list of on-chip macros, the footprint image of the package and the die. The macros are floorplanned first. I/O pads are then placed across the die, keeping in mind the vicinity to the package pins and placement restrictions due to macros. I/O Planning is validated for electrical rules. [3]

Floorplanning starts with the I/O plan and the post-synthesis netlist. Logic flow, vicinity to the placed I/O and macros, timing and congestion are considered. A good floorplan is key to successful physical design.

Test synthesis and clock insertion are carried out in *netlist processing*. [4] Scan logic is added to the top-level inserted netlist and scan chains formed. Test structure verification validates the scan insertion. The IEEE 1149.1 JTAG boundary scan logic is also chained together. Clock design is done keeping in mind latency, skew, transition time, clock tree configuration and layout considerations.

Manufacturability Checks are pre-layout checks to verify test logic, macro and boundary structures to qualify for manufacturability test suitability. *Technology checks* validate the netlist for the technology rules.

Logic and timing consistency checks are performed after each netlist modification. Static timing analysis is used for timing consistency checking while logic equivalence checking is used for functional consistency checking. They eliminate the use of gate level simulation for consistency checking and thus are among the key features of our methodology.

Layout involves two stages. In *Preliminary Layout*, the physical design center performs detailed placement and wiring, optimizes them, analyzes wireability, validates floorplan, analyzes clocks. *Production layout* is performed next. Here, clock trees are balanced for skew minimization, and wire lengths as well as capacitive loading on latches is minimized along with final routing.

Layout verification on the production layout checks if the performance targets and manufacturability requirements are met. It includes technology checks, static timing analysis with process variation modeling, test structure verification and check for minimum 99% test coverage. *Manufacturing test data (ATPG)* is generated.

Sign-off is completed after layout verification and ATPG. A comprehensive checklist is used here to establish readiness for sign-off.

5. Functional Verification

Significant portion of development time is spent in functional verification of complex designs. For early product deployment, it becomes imperative to reduce the verification time without compromising on the quality. Newer and innovative techniques are required to verify multi-million-gate designs. Bottom-up approach is needed to verify a design at unit level, chip level and system level, to promote early detection of defects in the system verification cycle. These challenges necessitate reuse of verification components to speed up test environment setup for different levels of verification and development of test environment at higher levels of abstraction.

Our designs consist of multiple programmable data inspection and formatting engines and buffering logic. To verify the buffering, congestion and flow control functions, a probabilistically driven random traffic source model is required to generate realistic network traffic patterns. Directed testing alone is not sufficient, considering the multitudinous combinations of configuration and traffic arrival patterns over multiple coexisting connections/flows. To cover hard-to-think possibilities, constraint-driven random testing is required.

The objective of verification is to validate the design functionally against the specification. The interpretation of specification must be uniform across all levels of verification. A key challenge is to represent the specification in an executable form that can be directly used in different verification environments. We took a "C" modeling approach to get the executable specification architecturally validated and to use it as the golden reference in subsequent stages of design verification.

Catering to these challenges and features, our verification approach, referred as *Dual Design Verification Methodology (DDVM)*, uses the golden model, mentioned earlier, as the reference design against which the RTL is validated. It supports validation of RTL behavior in random testing and makes the verification setup regressionable. This approach not only reduces the functional verification time but also speeds up system testing. *DDVM* is explained in the next section.

Dual Design Verification Methodology

Our verification approach is illustrated in Figure 2. As evident from the figure, it provides a self-checking verification environment. Here, two representative designs of the same specification are employed. The Design in "C" is the behavioral model, which is used in architectural validation, thus can be viewed as an alter ego of the specification. As discussed earlier, thorough verification at unit, chip and chipset level is required to weed off defects and to have abstraction in the level of testing. *DDVM*, when employed in unit level testing of design components, uses RTL representation of the design unit, which is validated against the "C" representation. In chip level verification, the "C" models of all units are integrated to form a behavioral representation of the chip. Similarly, chipset level simulation setup is realized. In this approach, the other components in the verification

environment are used in all the levels of verification. This reuse cuts short the environment development cycle time significantly. As is evident, our approach easily paves the way for a regressionable verification setup because of highly automated on-the-fly validation of response.

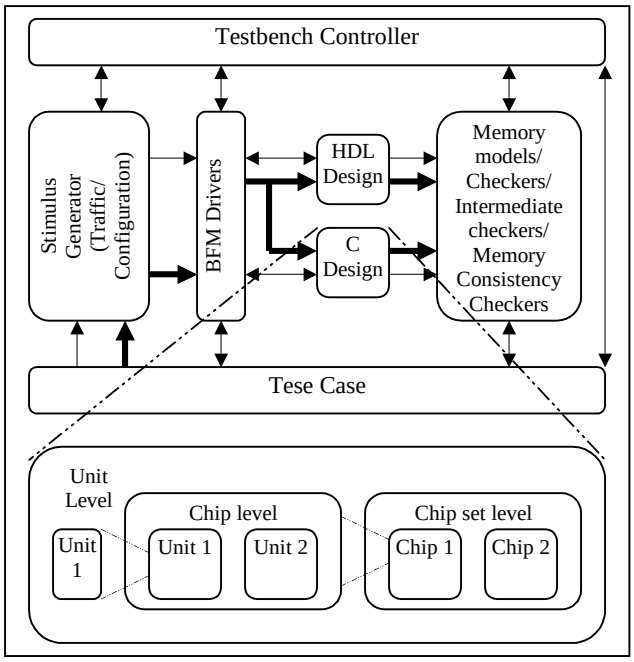

Figure 2 Dual Design Verification

Custom-built constructs and data structures in C, combined with the verification friendly constructs of Vera, shorten the test case and environment development time. It also makes it easier to maintain, debug and reuse.

Random testing can easily be done in *DDVM*. Randomization of input test stimuli can be statistically controlled using dynamic feedback from the test environment. Automated, on the fly self-checkers, help validate test responses. We recommend random testing in verification of networking designs as we uncovered number of critical defects, which were either hard-to-detect or went undetected in directed testing.

In typical system verification scenario, design not only the needs to be verified at unit, chip and chipset level simulations but also in the integrated system environment where hardware and software components coexist. Traditional co-simulation approach is costly and time consuming. In our approach, the RTL design and the behavioral C model are functionally coherent after the thorough chip level functional verification. Hence the C model can be used as another representation of the RTL design in software integrated system simulation testing as well as in standalone software testing. This helps in faster prototyping than conventional methodologies allow.

We employed code coverage and bug arrival rate as key metrics for establishing verification adequacy before important releases of the design. Bug arrival rate, which measures the number of bugs detected with respect to the increase in number of executed test cases and the increase in functional coverage achieved. At least 99% code coverage, flat bug arrival rate and 100% functional coverage is required in order to enable tape-out decision.

6. Static Verification Methods

A VLSI design undergoes multiple changes after the RTL representation - in synthesis, due to test logic insertion, clock tree insertion, etc. Functional and timing consistency must be maintained after each change. Verifying consistency without overhead of time calls for efficient verification methods.

Both dynamic and static methods are used in design consistency verification. Dynamic methods such as simulation stimulate the design by applying external test vectors over a period of time. The more the vectors used, the higher the functional coverage. Our designs had, for example, more than a thousand directed tests for chip level verification. While simulation is best for functional verification of RTL level design, the same doesn't hold good at gate level. Gate level simulations run much slower than RTL simulations and become a bottleneck in achieving rapid regression cycles for every change in design. Static methods, in contrast, are vector less, exhaustive and much faster than dynamic methods of verification. We discuss here two static methods employed in our methodology for consistency checking.

Static Timing Analysis

Static timing analysis examines each timing path in the design and checks the path delay against the assertions. Checking doesn't need functional vectors; thus is faster than dynamic simulation. It is exhaustive since it examines all possible timing paths in the design. Hence, coverage of static timing analysis does not depend on the quality and quantity of test vectors.

A complete run of static timing analysis on our multi-million gate designs takes only a few hours, a significant improvement over gate level simulation that would take hours even for a single test. Our methodology uses static timing analysis for timing verification after synthesis, floorplanning, scan insertion, clock tree insertion, clock routing and in layout.

Static timing analysis tools can also profile the variation of timing into a histogram (*histogram analysis*), and point to instances that contribute to maximum number of timing violations (*bottleneck analysis*). This is very helpful in finding timing problems, especially with large designs having hundreds of thousands of timing paths.

Timing assertions are derived from the ASIC goals. They are created right at the beginning and used

consistently there on. For gate delay computation, DCL (Delay Calculation Language) models were consistently used. Maintaining consistency of assertions and timing models is a key to consistency in timing checks.

Formal Verification

There are many formal verification methods – model checking, equivalence check, theorem proving, etc. Equivalence checking is basically a comparison of two designs for functionality. One of the designs is assumed to be golden and the other is checked against it using a tool. Equivalence checking tools are algorithm based and works on logical representation (truth table) of the designs. Equivalence check can be run between any two representations of a design. Unlike simulation, equivalence checking does not need test vectors.

We used equivalence checking for RTL to gate netlist checking, gate netlist to gate netlist checking and for ECO changes. The RTL representation of the designs, verified through functional simulations, is treated as the "golden" model for equivalence checking. Further representation of the design, for example, the netlist after synthesis, netlist after scan / clock insertion or a layout netlist, is checked against the golden RTL model. Like static timing analysis, equivalence checking is also exhaustive, and provides complete coverage. Thus, there is no need to run gate level simulations for logic consistency checking, which saves a considerable amount of time.

Being functional vectorless, equivalence checking runs very fast. A million gate design, for RTL to gate equivalence check takes only ten hours. Gate to gate checking runs even faster. Further speed-up can be achieved using hierarchical approach.

We discovered a couple of synthesis related problems and a few scan insertion related problems through equivalence check. However, it should be noted that equivalence checking is not a substitute for functional simulation. If functional simulation was inadequately done, equivalence checking cannot find problems in implementation. Also, it does not completely eliminate the need for gate level simulations. In our case, equivalence checking cannot find problems with phase relationship of the two clock outputs from the PLL, for which gate level simulation is required.

Static verification approach saved us enormous time, allowing us to move from step to step faster and execute some of them in parallel. Static timing analysis and logic equivalence checking stand out as two solid pillars of our methodology that easily support large complex designs.

7. Timing Closure Techniques

Timely time closure is a significant challenge in high-performance deep-submicron designs. Traditional "over-the-wall" approach, characterized by little interaction among logic and physical designers, no longer works. Closer interaction with physical designers and awareness of physical design issues among logic designers, are the needs of the day. Iterations between physical design and logic design are no more exceptions. Successful timing closure methodology must employ techniques, both in logic and physical design, to minimize iterations.

Each of the ASIC in our chip-set is unique - One is large, IO limited and dense; the other is IO limited, has large on-chip macros but limited logic, the third is large, not dense or IO limited but contains many on-chip macros. Our methodology addresses the needs by beginning physical design process early and by employing several physical design techniques.

Early Beginning of physical design

Selection of die size, package and I/Os, framing of design guidelines, I/O planning and floorplanning are crucial in making design decisions based on potential timing / physical design problems. The earlier these are performed, the lesser the risk of starting all over again and wasting time and resources.

Early design partitioning and finalization of the number of signal I/Os helps deciding the die size and package, significant especially for I/O limited designs. [6] Large macros result in placement and routing restrictions. They also obstruct power routes. The required logical size of an *on-chip macro* could be realized through multiple configurations of single/multiple macros. Trade-offs must be made while choosing the on-chip macros, keeping in mind the die size, potential restrictions and timings.

Design guidelines are driven by the required performance, selection of ASIC library, die size, I/O methodology and package selection. Guidelines that we followed, like point-to-point global nets, fanout restrictions, for the high-speed nets crossing unit boundaries, for those connecting on-chip macros and I/Os of the chips, and maximum logic depth between two flip-flops, were instrumental in minimizing iterations between physical design and RTL.

In designs employing area-array I/Os, *early selection of I/Os, die size and package* helps in proceeding with I/O planning. Spice simulations are carried out to validate the selection of I/O pads, the timing on the I/Os. [6] *I/O planning* is carried out to satisfy the board layout requirements, the placement restriction caused by on-chip macros, grouping of pads so as to limit the SSO and to minimize IO wiring. A good I/O plan thus goes a long way in achieving timing closure on the I/O pins and on related logic, significant especially for high-speed I/Os.

Floorplanning is carried out when the first core netlist is available. It starts with the I/O plan and distributes the logic across the die in physical groups,

while keeping in mind the restrictions caused by on-chip macros, logic flow, future introduction of clock tree, prevention of highly dense "hot-spot" areas, the vicinity to the on-chip macros and I/O pads. Early floorplanning helps identifying potential problems like long global routes, high fanouts, and routability problems.

While it was possible to divide the die onto exactly blocks that don't overlap, overlapping is intentionally allowed to allow adjustments in floorplan to fix routability / timing problems.

An important consideration in synthesis is to *prevent use of gates with very low drive and the complex logic gates that don't have scalable drive strength range*. Gates with very low drive can only drive very small local nets. Complex gates typically exhibit larger delays and with only a few drive strength options, they are difficult to size up later in physical design. Early examination of the ASIC library is done to identify such gates and to prevent them from being synthesized.

Modeling of timing assertions close to reality is the key to successful timing closure. High Pessimism is not good as it could mislead the tools and the designers and real problems could remain hidden for long.

Physical Design Techniques for timing closure

While early work in physical design certainly helps, it does not prevent all the problems. In deep-submicron ASICs, significant timing issues are attributed to the vast difference in the estimated and actual route delays. Since route delays account for as much as 70 % of total path delays, the gross inaccuracy leads to large number of timing violations across the chip. Actual routing can provide accurate timing information but is not possible to do early. High speed ASIC such as ours don't have large timing margins available to absorb such inaccuracies and hence physical design must employ alternate techniques to converge on timing in spite of the inaccuracies of estimation. Some of these techniques are described below:

In-place-optimization is a traditional method where weak gates driving large loads are resized while retaining their placement. Resizing can reduce the gate delay considerably and thus improve path timing.

Buffer insertion involves inserting buffers on a net, effectively reducing the length of the resulting net segments and hence in reduction of loading, resulting in lesser delay in driving gates. Buffer insertion is a very effective method of solving timing problems associated with global long routes. It not only solves timing problems but also helps in balancing capacitive loading across nets.

Nets with high fanout could have considerable capacitive loading when the target logic of the various fanouts is located in different physical areas. Overall timing associated with such nets can be improved by *splitting* them into multiple segments using buffers and letting the buffers drive the localized fanouts.

Even when these techniques are employed, a few paths would still not time-close. These are typically long paths, paths belonging to the interfaces to on-chip macros and I/Os, and paths having multiple high-fanout nets. These are analyzed in detail, and *a change in RTL implementation is* called for. RTL changes for timing closure typically involve breaking a path by adding pipeline stages, splitting the critical and non-critical portions of the path, cloning of logic and reducing fanouts.

8. Summary

Our multi-million gate chip-set design was completed in a short span of ten months. The chip-set was a first silicon success and the ASICs are perfectly working to the specifications in the field.

Economy of integration at deep-submicron geometries has taken the complexity of functional verification to astronomical heights, and made timing closure is a long iterative cycle. The methodology presented in this paper smartly combines novel techniques and state of the art tool-set to enable defect free silicon and thus a significant time-to-market advantage.

Acknowledgements

The author wishes to acknowledge the contribution of teammates Prasad Bhatt, Rajagopalan K, Sivakumar B, Shanthamoorthy V, Ramamohan S, Manjunatha B P and Mahesha Puttanna, for their efforts towards this paper.

References

[1] Simultaneous Switching Analysis Overview, IBM Application Note, Revision 2, 01/01/99.

[2] Doug Dreibelbis, Paul Zuchowski, and Patrick Buffet, "Hidden Benefit #4: ASIC's Area-Array I/O Footprint", IBM MicroNews, First Quarter 2001, Vol. 7, No. 1.

[3] Patrick H. Buffet, et al., "Methodology for I/O Cell Placement and Checking in ASIC Designs Using Area-Array Power Grid", IBM MicroNews, Third Quarter 2000, Vol. 6, No. 3.

[4] Vivek Chickermane, David Lackey, Dave Litten, and Lori Smudde, "Automated Chip-Level I/O and Test Insertion Using IBM Design-for-Test Synthesis", First Quarter 2000, Vol. 6, No. 2.

[5] M. Kuzawinski, "The Limits of Wirebond Technology: Is Flip-Chip Attach Now a Prerequisite for Advanced Packaging?," IBM MicroNews, Vol. 6, No. 4.

[6] Keith M. Carrig, "Chip Clocking Effect on Performance for IBM's SA-27E ASIC Technology", IBM MicroNews, Third Quarter 2000, Vol. 6, No. 3