Design of a 17-million Gate Network Processor using a Design Factory

Gilles-Eric Descamps, Satish Bagalkotkar, Subramanian Ganesan, Satish Iyengar, Alain Pirson
Silicon Access Networks Inc., 211 River Oaks Parkway, San Jose, CA, 95134
{Firstname.Lastname}@SiliconAccess.com
www.siliconaccess.com/idf/dac03

Abstract:

*Silicon Access Networks taped out in one year four high performance SoC products: a high-end Network Processor and three associated Co-processors, providing the industry with the highest performance OC-192 Data Plane Processing solution. The four chips are shipping for revenue and went into production from first silicon with no mask change. They were designed using state-of-the-art 0.13*µm *technology and collectively represent about 750-million transistors, implementing a variety of analog, digital, high-speed memory and functional blocks.*

This contribution describes the design of the Packet Processor and some of the key aspects of Silicon Access Networks' design methodology that enabled to accomplish repeatable "first pass silicon" successes, despite system complexity challenges. The 175-million transistor iPP was simultaneously designed in three locations (San Jose/CA, Raleigh/NC, Ottawa/Canada). Bring-up and pre-production showed that first silicon met all its targets: power, speed, yield and complete functionality.

Categories and Subject Descriptors: K.6.1 [Management of Computing and Information Systems]: Project and People Management -- Management techniques ; J.6 [Computer-Aided Engineering]: Computer-Aided Design ; B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids ; B.7 [Integrated Circuits] : Types and Design Styles

General terms: design, management

1. INTRODUCTION

Silicon Access Networks introduced in Q1'02 its breakthrough iFlow Data Plane Processing Platform, a family of semiconductor and software products that provides a complete solution for 20Gbps line rate packet switching and routing. The heart of the platform is the iFlow Packet Processor (iPP), the industry's first single-chip 20Gbps Network Processor.

This highly complex 175M transistor, 17M gate processor is a System-On-Chip design as defined by Keating and Bricaud [3]. It consists of:

- 4 multi-processors (called *Atoms*) and their memory subsystem. Each processor is made of 8 multi-threaded Network Processing Units (NPU). Together, the 32 NPUs support a total of 256 concurrent threads of execution with zero latency context switch. The memory sub-system includes register files, on-chip SRAM and CAM and an optional off-chip memory.
- A datapath that reassembles and dispatches 30M packets per second through a pair of SPI4.2 interfaces, each capable of sustaining 12.8 Gbps.
- Blocks that perform flexible packet editing allowing for adding, replacing, inserting, or deleting fields.
- Other I/O interfaces including a PCI 2.2 control plane interface, 5 LVDS high-speed coprocessor channels and a QDR SRAM interface.

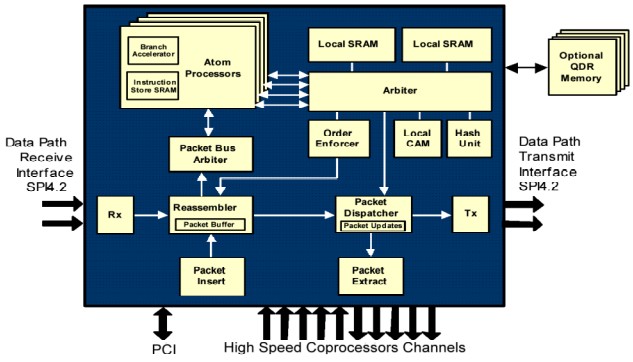

Figure 1: iPP functional

This paper presents the design methods we adopted to successfully develop the Packet Processor. We will address the methodology and its implementation covering logic, custom and physical design and verification.

The concept of *design factory* is to produce tapeout data in a consistent and timely fashion. It relies on a unique standard design process applicable to all designs with any available resources. This was critical as we had to simultaneously build four chips with aggressive time-to-market constraints using dynamic resource allocation.

2. PLANNING

Time-to-market without compromising the performance was the critical objective. The implementation of the chip was broken into five different phases, each focused on a particular design goal. The deliverables of each phase were well documented and enforced throughout the entire design process. The refinement nature of the process ensured continuous convergence[9].

DAC 2003, June 2-6, 2003, Anaheim, California, USA.

2.1 Base 0: Budgeting/Floorplanning phase

Budgeting is one of the key phases of chip development as it establishes the foundation of the design process. Any miscalculation, mistake or poor estimation during this phase can have a costly effect in a late phase of the design cycle. Emphasis during this phase is on the definition of all the block inputs and outputs, estimation of block sizes and aspect ratios, generation of a chip level floorplan, power estimation, establishment of a repeater insertion strategy, design and simulation of clock distribution, generation of correct timing budgets at all block boundaries, definition of multi-cycle and false paths, first cut synthesis and basic testing to validate the netlist.

In order to accurately estimate the size of the blocks, several test cases from previous and present designs were selected to address the spectrum of the design. These test cases were divided into several categories based on blocks (small, medium, large) and memories (sizes and number of instances). These test cases were then taken through the entire design cycle until trial tapeout. This was useful for testing and debugging the fully automated synthesis and backend flow as well as to establish metrics on how the design evolves as it goes through the process. The results of this exercise were then formulated into guidelines and used for size estimations and timing budgets at each phase of the design cycle.

To simplify the design, most of the inputs and outputs of blocks were flopped. In addition, fanout of all inter-block nets was limited to one wherever possible. Rule based buffer insertion was used at the top level. This was all done with the intent of pushing the complexity of the design from the top level into the blocks. This ensured single iteration timing closure with no setup or hold violation at chip level. Based on the characteristics of all the cells in the library, guidelines for levels of logic between flops in all intra-block paths were set and monitored. Detailed spice analysis was carried out using available repeaters to arrive at a default flight time (picoseconds per mm). This number was uniformly used to budget inter-block timing.

All the block timing budgets were defined based on three variables: departure time (time needed to generate the signal in the source blocks), flight time (time to get from source blocks to destination blocks), and required arrival time (time required after signal arrives at block boundary until it is flopped). These numbers were automatically converted into design constraints, and were verified by static timing analysis. As design evolved, the timing of each block was compared with these budgets. Nightly regressions were run to ensure that the design did not diverge.

An additional goal of this phase is to identify the critical paths such that most architectural and micro-architectural decisions could be made. The deliverables of this phase is to have 100% of the paths meet timing based on budget and 70% of the paths meet timing after preliminary synthesis.

2.2 Base 1: Prototype/Synthesis phase

The prototyping phase requires 75% of functionality coded, 90% of those paths meet post-synthesis timing and 33% of the functionality verified.

This phase is primarily focused on the front-end aspects of the design: rtl coding, synthesis, block floorplanning, placement, static timing analysis, writing and running test benches and zero delay gate simulations. After the blocks are synthesized, static timing analysis is performed to ensure that the paths meet the timing budgets defined in the first phase of design. The preliminary netlist is taken through the back-end flow, only until the placement stage, to get to within 10% of budgeted timing target. These placements are reviewed by a team of experts to ensure no physical design issues are ignored. This helps establish detail block level floorplan and freeze macro placement and pin location. Sub-partitions that need to be grouped together for placement, are identified. Most of the blocks have to go through several placement, rtl modification and synthesis iterations to get the blocks to within 10% of timing closure. Constraints, multi-cycle and false-paths issues are resolved. Power rails and clock distribution are implemented and analyzed. Zero delay gate level simulations are started to ensure the stability of the design. We used for the Packet Processor, an FPGA-based commercial hardware accelerator.

2.3 Base 2: Implementation/Physical design phase

Entry into this phase requires a netlist with 95% functionality coded, 90% functionality verified and 100% of those paths meet post-synthesis timing to guarantee the maturity and stability of the design.

This phase is about execution and is the most time consuming. Indeed, 90% of paths have to meet post-route static timing. The netlist is taken through the entire backend flow which includes placement, routing, post-route static timing analysis, clock analysis, power analysis and formal verification. Any problem related to congestion or routing is identified and fixed. Most of the blocks are taken through several iterations of backend as logic bug frequency peaks during this time. Critical blocks are identified. Emphasis is placed on the stabilization of these blocks so they don't stay in the critical path for tapeout. Additional test cases are written for these blocks to further reduce the likelihood of finding new bugs during subsequent phases.

2.4 Base 3: Bug Fixing and Timing Closure phase

Entry into this phase requires that 100% of functionality is coded and tested, and 100% coverage is met. This is the last phase where synthesis is permitted for bug fixes. All the inputs - tools, IP, custom memories, technology files, libraries, etc – are frozen.

This is the closure phase where the final 10% of post-route timing issues are addressed while shifting the focus to physical verification. Most of the blocks go through several iterations of DRC, ERC, and LVS to ensure the blocks are clean. Fixes are implemented and most of the blocks will be shelved for tapeout. Chip level physical verification is started to identify problems that might surface at the block boundaries. Post-route block and chip timing are closed.

2.5 Home Run: Tapeout phase

As the name indicates this is the final phase of design, focused on chip level DRC, ERC, LVS. Any functional bug found at this stage is fixed as an ECO. The final timing, power and clock analyses are performed. This phase is mostly compute intensive with very little human involvement. The chip level GDS is assembled and signoff sheets are completed. The final zero delay, SDF gate and SPF based simulations are run for 2 weeks before the GDS is signed off to the foundry.

3. EXECUTION

The Design Factory is a complex web of handshakes between several supply chains as illustrated in Figure 2.

Some portions of the total design are complex enough to be handled as autonomous entities with their own resources, schedule & deliverables (e.g. the Atom processor). The high-performance requirements of Atom and its tight area constraints led us to develop a channel-less flow, with some similarities to a virtually flat flow[13].

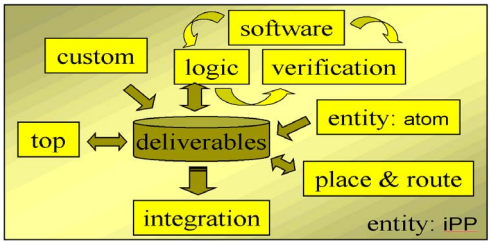

Figure 2: the "supply chains" of the Design Factory

By defining a formal exchange procedure between nodes in the Design Factory "supply chain" (or team of expertise), we are able to track all the deliverables in a multi-site environment. QA processes independent from the design qualify each delivery in a simple color-coded overview.

3.1 Custom macros

The first step of the design process was a secure delivery of memory models to the architects and logic designers. A tracking system was needed to manage hundreds of hard macros (memories, I/Os, data paths, analog blocks) used in various chips, and being built by several teams.

Vendor supplied IP is usually easier to integrate as it already comes packaged. However we encountered that these external products lacked in quality and completeness at times. On the other hand, internally developed cores need product engineering as they were produced by several specialized designers (schematic, layout, logic, characterization). The tracking system has to be a knowledge management infrastructure[4], allowing each designer to learn how his/her work integrates with the work of others. We built an in-house web tool implementing an IP core catalog offering easy access, secure & organized tracking of data, qualification and generation of dependent views, sharing knowledge of design conventions, methods & status. This web-only catalog supposed no previous knowledge from the designers. It allowed them to easily figure out what kind of file is expected from them, which guidelines they should adhere to and who is interested in their deliverables. As the catalog was linked to the intranet roster, a single click enabled any designer to reach (mail, phone, cell, location) the producer – or customer – of any view. Standard bulk data vaults were used to store the design data, while an SQL database was used to capture the intent & actions. As soon as a new version of a view was uploaded, a qualification script was run, enforcing the current company procedures, and informing the designer of the problems encountered in the deliverable.

Figure 3: IP core catalog: list of views

For example, one of the quality check was to make sure that all views were pin-compatible. Generation of dependent views like a *cdump* from a *lef*, or a *db* from a *lib*, were also automatically triggered by the upload. As our toolset was rich, we had to support several vendor-dependent views that could be automatically generated from a standard format. The system was even able to build complex dependent views like the whole Avanti Apollo database. Each view was clearly qualified by color-coding: green for pass, orange for warnings, and red for errors.

This web cataloging system managed the delivery of over 500,000 files with a 24/7 availability.

3.2 Logic Design

Micro-architecture definition and logic design, based strictly on the architectural specification and performance requirements of a chip is insufficient. Physical limitations of the process and the methodology need to be understood to achieve the stated goals of the design. "Correct by Construction"[1] techniques and reduced dependencies allow distributed teams to manage the complexity problem at various stages in the process, leading to chips that work correctly the first time.

The iPP was divided into numerous blocks and sections on clear functional boundaries. The micro-architecture and front end design of these blocks were done keeping in mind the timing and area goals for each of the blocks. The interface between blocks was strictly synchronous to ensure that no path in the design gets limited by the delay introduced by long wires.

Margins allow the logic designer or micro-architect to identify critical paths early in the design and make appropriate changes or trade-offs. To ensure speedy convergence in fewest possible iterations, the following margins were used at the beginning of the design of the iPP:

- Clock Skew: Typically 5% of the clock period.
- Jitter: Reasonable margin for clock jitter.
- Synthesis: Margin for addition of buffers between synthesis and P&R. Typically 5% of the clock period.
- Block size: Margin for the case where the cells in a particular path end up being placed far apart. A variable based on the geometry of the block and the unit wire delay of the technology.
- Noise: coupling margins. Typically 5% of the clock period.
- Signoff: Margin to account for devices arriving at the low end of the speed curve. Typically 5%.

Therefore, $t_{period} >= t_{clk->q} + t_{pd} + t_{setup} + t_{margin}$, where $t_{margin} = t_{skew} + t_{jitter} + t_{syn} + t_{blk\_size} + t_{noise} + t_{signoff}$

Most margins are consumed during the process. Only jitter and signoff margins remain at the post-route timing analysis. Another aspect of a "Correct by Construction" design is that the various teams – verification, back-end, semi-custom, custom are working on the same design at the same time, even if they are in different geographical locations. Following Software Engineering QA methods [8], every night, an automated process would tag changes, build the design from scratch and qualify it using established procedures: compile to verify the syntax / completeness, lint to help enhance the quality, run block and chip liveness – a time-limited regression suite of functional tests that ensure that the design is fundamentally sound – and gather metrics on the current synthesis.

Every morning, each owner was informed of problems found. Anybody could browse a fresh set of web pages providing a color-

coded overview of the current state of the design, which could be drilled-down to get extensive reports. This also simplified the roll-out of official releases.

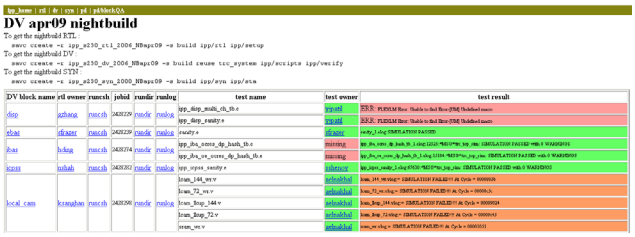

Figure 4: DV night build web page

3.3 Functional Verification

Like any SoC design, the Packet Processor presented a real verification challenge. Its elaborate functionality (about 1500 pages of specification) and its heterogeneous design required a multi-disciplinary approach based on complementary test strategies to address the problem from various angles.

The first step of the functional verification was the creation of an executable specification using System-C. This bit-true and quasi cycle accurate model provided a flexible software development platform for test and application microcode. It was actually integrated into the iFlow Programming Environment (iPE) and used to develop the iFlow Reference Software (iRS), a typical switch/router application. Both iPE and iRS were key in verifying the architecture specification and the design of the Packet Processor.

The C-model was also turned into a powerful plug-and-play host platform for RTL modules. This strategy allowed early introduction of hardware modules in a system-level environment, hence early discovery of bugs usually showing up after RTL integration. This was especially beneficiary for the Atom processor that could rapidly run application code.

At RTL level, the verification proceeded with a divide-and-conquer bottom-up approach as recommended by Bening and Foster [2]. Three levels of verification were adopted: block-level, transaction-based chip-level and system-level.

3.3.1 Block-level verification

All the major components were fully tested at block-level. Due to its nature (a multithreaded multi-NPU processor core), the Atom was singled out as a standalone project with its own verification team.

Two functional test strategies were adopted for the processor. The first one implemented directed tests to cover most of the functional behavior. These tests were executed in multi-threaded mode (64 instances of the test running concurrently on the Atom) to validate the resource sharing mechanisms. They contained some self-checking microcode allowing each thread to detect faulty behavior. This technique removed the dependency from the C-model that had a hard time predicting the load balancing between the 8 NPUs. Test development was automated and led to a suite of 7000 directed tests.

The second strategy took advantage of random tests to stress the design. Many aspects were randomized: microcode, packet data, transaction response delays… The test cases were created using Maurer's Data Generation Language (DGL)[6]. Based on the concept of context-free grammars, this original language is used to code test templates, elegantly capturing the elements that need to be randomized. The templates are then compiled into data generators that in turn produce the test cases. This powerful approach helped creating microcode for the 450 million cycles of random tests that were simulated (1.5 seconds of processor time at the target frequency of 300 MHz).

An ingenious strategy was devised to cause the random tests to be self-checking. It consisted of having each thread compute a CRC value of its register file at the end of each packet processing and compare this value against a pre-calculated one embedded in the input stream itself. The C-model was used to calculate the reference CRC values. Self-checking random tests proved their efficiency on the hardware accelerator, turning a 2-day simulation into a 3-minute run.

3.3.2 Transaction-based chip-level verification

Moving up the verification effort to chip-level, an innovative transaction-based strategy was put into place. It relies on a homegrown protocol of verification packets where each packet carries in its header the series of transactions or data manipulations it expects from the Packet Processor. They include packet editing and dispatch, memory and Cam access, semaphore and timer operations… The limited number of transactions makes it easy to compute the expected output packets without the need of the reference model.

A generator produces a bit-stream from a sequence of verification packets described with a high-level language (Specman E). At simulation time, the Atoms pull out the verification packets from the bit stream, decode and execute the requested transactions. The output stream is collected and matched against expected data, checking bit accuracy as well as packet order.

This methodology proved very effective. Using the Atom as a virtual machine, it eliminated the need for test-based microcode. Test cases were fully characterized by the input stream. As soon as this approach was in place, the verification productivity soared. Writing a new test simply consisted of filling the fields of an E structure defining each verification packet.

3.3.3 System-level verification

System-level verification consisted of simulating the RTL model running the iFlow Reference Software, testing all available Ethernet and Point-to-Point (PPP) configurations on large numbers of packets. This effort started with the plug-and-play approach, inserting major RTL modules in the C-model. It culminated with the entire RTL design processing packets.

It was important to inject large numbers of packets to fully stress the multiprocessing capabilities of the Packet Processor. We used the hardware accelerator that allowed the 17M gate design to reach speeds of 12,000 packets per minute. 20M packets were simulated.

3.3.4 Coverage

The purpose of coverage metrics is to quantify the level of confidence in the verification effort, providing a means of efficiently managing it throughout the project. Bening and Foster [2] present a taxonomy of various metrics used in today's hardware design practices. We applied code, state machine and functional coverage, an ad-hoc metric: the bug detection frequency, and we introduced the concept of percentage of

functionality tested. This percentage is calculated by adding for each block the ratio of its number of passing tests over its number of expected tests, weighted by the relative complexity of the block (estimated gate counts). This criterion was highly regarded to gate the RTL releases according to the phases described previously.

Functional coverage was introduced to guarantee that all the functionality in the specification was present in the design. Using monitors, it ensured that specific instruction sequences were exercised, provided information on table and FIFO access and address ranges. It was vital to confirm that all the arbiters were fully exercised. This was achieved by tracking transaction request buffers.

3.4 Physical design flow

The physical design process for such a sophisticated chip is very extensive due to the high number of complex tools (~15) and steps (~60). By using a fully automated repeatable physical design flow, the back-end teams were able to support front-end design with a predictable turn-around time. That flow delivered "tapeout quality" GDS from netlist, in 80% of blocks just by pressing a button. The remaining 20% of blocks were 98% tapeout ready, requiring only minor corrections. We define "tapeout quality" as achieving: timing closure, signal integrity closure, power analysis, DRC / ERC / LVS, Antenna, DFM, formal equivalency and even 3D spice analysis of clock and critical paths. Breaking the design into manageable block size ranging from 1K to 150K instances ensured a turn-around time of 24-48hr from netlist to tapeout quality.

3.4.1 Automated flow

Using an advanced 0.13µm technology in its infancy introduced a lot of new challenges[10]. No single EDA vendor was able to provide a complete solution. We evaluated several tools, selected the best-of-breed, and integrated them in a flow. We used a commercial "visual make" to give each tool the same look & feel interface, and to gain visibility in the design process by tracking important files. We wrote gateway scripts for a smooth data exchange between tools. This addressed not only simple syntax conversion, but also undocumented semantics pitfalls. A centralized CAD and Methodology[11] team captured the knowledge of experts into generic command files.

An abstraction layer around each resource helped us to shield from potential resource problems. Instead of rlogin into a specific machine and firing a job which would hold resources while waiting for a license, a flow designer would list requirements – machine, memory, licenses – and let a central queuing system find the best available machine. Multi-site design was addressed by mirroring only necessary data on remote sites and providing each campus with sufficient local compute servers. Efficient use of the compute and license resources was significantly raised by deploying a multi-site queuing system which would arbitrate resources based on complex company-wide policies. For some tools we managed to increase the average usage of licenses over a week from 46% to 81%. The large number of blocks – almost a hundred – reduced the granularity of the jobs and smoothened the workload. Adding more compute servers or more peak licenses would shorten the design cycle, instead of being stuck waiting on a huge process at it is still too often the case in flat EDA software.

3.4.2 Predictability and Visibility

Several "exit" points – floorplan, place, route, drclvs – were specified with predefined quality and completeness criteria. A script would then extract significant metrics from the released data for each block and compile this information into a single web page with color-coding. This gave the whole company an instant overview of the state of the project. Due to the aggressive schedule, and the presence in-house of experts covering all domains, a full-fledged data-mining effort like METRICS[7] was not justified.

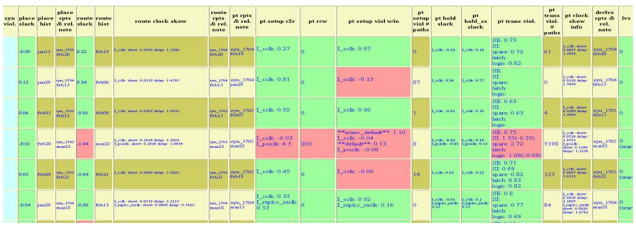

Figure 5: sample project overview

Data and results for all the exit points could be accessed from this table. A single-click could turn a slack number into detailed timing report, graphical slack histogram, or number of errors/warnings.. In the same manner, graphical floorplan, placement, or routing congestion map, or clock skew/slew 3D spice were just one click away. This also helped independent reviews to identify any new potential problem.

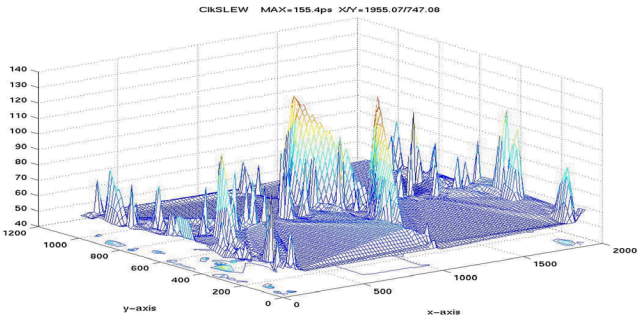

Figure 6: spice clock slew mapped to block layout

Due to the generic commands and scripts across all projects, similar results would be obtained every single time. By deploying a centralized flow, which was tuned for that design spectrum, we eliminated most of the unknown factors from the equation, and kept the netlist as the single variable. By preventing user mistakes and avoiding re-work, the tapeout was on schedule. The procedure was built so that the user could not bypass any steps or work around any error. By capturing all the relevant files and migrating them in a company-standard exchange place, we were able to consistently recreate any data.

Simple metrics - level of logics, net length, fan out, capacitance, miller caps - were gathered for early detection of potential problems. For example, we discovered that static timing analysis would fallback to the default value when the capacitance fell outside of the library range.

3.4.3 Technology

In 0.13µm technology, signal integrity and power-induced changes need to be addressed. Noise was partially handled at the "place" exit by a noise avoidance step. It was finally resolved by a noise correction step before the "route" exit in the form of

upsizing buffers, or wire re-routing depending on which router – Apollo / Nanoroute - has been selected.

At the top level, clock was implemented using a custom H-tree using tunable buffers leading to a 5ps skew. Inside the blocks, standard clock tree synthesis tools were used. Power distribution was done up-front and the power straps were pushed down in the blocks. A full chip IR drop analysis was run and feedback provided to the block owners. Special care was given to Design For Manufacturability (DFM) rules to maximize yield. For example, vias were consistently doubled. In addition, all large memories have built-in redundancy. This effort proved to be efficient since we are getting above-average yield from our foundry. Standard sign-off tools were used to validate all the results, as every tool had differences in their reports. For example, Apollo CTS, Primetime and Spice provide clock skew numbers. Spice was the sign-off tool for clock distribution.

By having such an extensive flow aggregating best of breed tools with foolproof procedures, we built a totally predictable back-end, where the consequences of a new netlist or ECO drop could be prognosticated reliably.

3.5 Chip integration

Physical verification of a large design in its entirety is often viewed as a challenge[5]. The GDS size of the iPP reached 8GB.

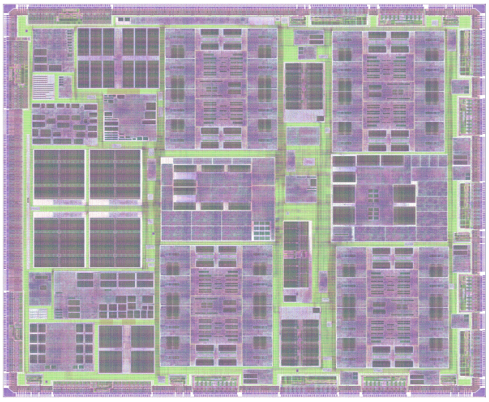

Figure 7: iPP layout

As block/entity and top-level implementation were clearly decoupled and the quality of the constituents was verified prior to delivery, chip integration was straightforward as planned. Chip level DRC/LVS runs were quick due to distribution. Thanks to trusted change control mechanisms, only the recently changed data needed to be re-run. The partitioning of the distributed run was based on the floorplan for easy ownership reasons.

4. BRING-UP

Bringing up the Packet Processor was a smooth, fast and successful endeavor. We first ran memory BIST at wafer-level and cut the fuses to repair the memories provided with redundancy. We packaged the clean parts and loaded them on our evaluation system, a board with a PMC (PowerPC Mezzanine Card) running Linux. The next day, 1300 functional tests were passing and the hardware debugger was up and running.

The Packet Processor met all its objectives. Despite its size and complexity, it was fully functional, required no mask change to go into production and yielded better than the foundry estimation. Packaged in a 1036 Ball Grid Array (BGA), it consumes the expected worst-case ~18 Watts (typical ~12 Watts) at 300 MHz, its nominal frequency, and reached the required processing rate of 30M minimal-size packets per second with a comfortable margin.

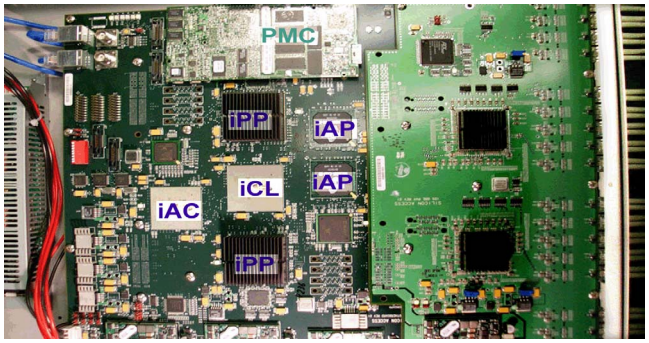

Figure 8: iFlow Development System

5. CONCLUSION

In this contribution, we presented a unique strategy to break a complex chip design into five well-defined phases. Execution of these phases was done through an indigenous fully automated, self-checking Design Factory. This design strategy reaches well beyond tapeout, up to silicon shipment. Its key notion is the continuous convergence[9] of the process, eliminating the usual snowball effect of problems, which jeopardize a project. By using closed-loop practices (plan, do, check, act) in the design flow, we were able to detect problems early, and avoid costly rework. The Design Factory concept provides predictable working silicon with complete visibility in the process. This concept was proven by delivering in one year, not only a high-performance network processor of 175M transistors, but also three additional coprocessors, using the same principles, for a total of 750M transistors.

6. REFERENCES

[1] S. Posluszny & Al., "Timing Closure by Design, A High Frequency Microprocessor Design Methodology", DAC 2000.

[2] Lionel Bening, Harry Foster, "Principles of verifiable RTL Design", Kluwer Academic Publishers, 2001

[3] Michel Keating, Pierre Bricaud, "Reuse Methodology Manual", Kluwer Academic Publishers, 1999

[4] Mentor Graphics, "knowledge management infrastructure"

[5] Oliver Ling, Raymond Lu "Overcoming physical verification challenges in a 100-million+ transistor SoC design"

[6] P. Maurer, "Generating Test Data with Enhanced Context Free Grammars" IEEE Software, Vol. 7 No. 4, July 1990.

[7] S. Fenstermaker, D. George, A. Kahng, S. Mantik and B. Thielges, "METRICS: A System Architecture for Design Process Optimization", *Proc. ACM/IEEE Design Automation Conf.*, 2000.

[8] Mark C. Paulk & al., "The Capability Maturity Model : Guidelines for improving software process", CMU, SEI, 1995

[9] Ping Chao and Lavi Lev, "Down to the wire -- requirements for nanometer design implementation"

[10] Craig Peterson, Tim Elliott, Naveed Sherwani, "Seven Critical Challenges of ASIC Design"

[11] Dan Smith, "Nvidia: scaling methodology", EDP, 2002

[12] Paul Rodman, "Hopper Hierarchical Flow Improves Turnaround in Physical Design of Large IC"

[13] Arun Balakrishnan, Gopal Dandu, Wolfgang Roethig and Benny Winefeld, "Physical Design Flow Taps Partition Layout"