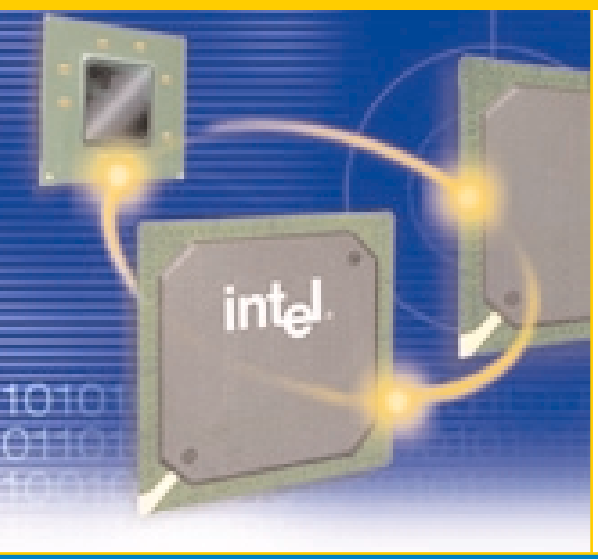
Volume 06 Issue 03 Published, August 15, 2002 ISSN 1535766X



Intel® Technology Journal

Network Processors

Challenges and Methodologies for Implementing High-Performance Network Processors

A compiled version of all papers from this issue of the Intel Technology Journal can be found at:
http://developer.intel.com/technology/itj/index.htm

Challenges and Methodologies for Implementing High-Performance Network Processors

Ram Bhamidipati, Ahmad Zaidi, Siva Makineni, Kah K. Low,
Robert Chen, Kin-Yip Liu, Jack Dahlgren
Intel Communications Group, Intel Corporation

Index words: network processors, reuse, verification, clock architecture, hierarchical flow, transactor, simulator

ABSTRACT

Moore's law has been the guiding principle for performance and transistor density improvements over the years. While this is true, in the context of network processor development, the challenge is multi-faceted to keep the silicon development on the curve.

This paper describes the challenges for a network processor implementation in each facet of design. The network processor designs adopted the following implementation techniques to manage the design challenges and the Time-to-Market (TTM) schedule:

Reuse of Intellectual Property (IP).

Extensive functional validation.

High-performance clock architecture and design.

Streamlined hierarchical physical design flow.

Efficient and cycle-accurate c-model for performance simulation.

A case study of implementation on the IXP2400 design is presented with the above strategies in detail.

The silicon results show that the IXP2400 is a successful design following the stated methods.

INTRODUCTION

Network processors are the emerging class of chips that offer Original Equipment Manufacturers' (OEM) flexibility in creating a wide range of applications. They are targeted to replace expensive and inflexible fixed-function silicon Application-Specific Integrated Circuits (ASIC). The implementation of network processors has to form-fit to the schedule needs of a telecommunication industry moving at the Internet speed. At Intel, we chose the architectural approach of providing a highly integrated and highly programmable solution to customers. This means a lot of functionality is packed into the silicon, thereby increasing its complexity for implementation. In addition to the functionality, the network processor has to operate at the targeted line rates, often running multiple tasks as demanded by the end-user applications. The applications may range from basic L3 forwarding to sophisticated algorithms that are more compute-intensive, as in the case of creating firewalls and intrusion detection services.

Network processors interface to a host of devices to perform their functions: media/switch fabric, PCI for control interface, DRAM for packet storage, Quad Data Rate (QDR) as a fast memory for queues and table lookup, and miscellaneous device support including Universal Asynchronous Receiver/Transmitter (UART) and General Purpose Input/Output (GPIO). A number of parallel microengines work on the data packets executing a specific microcode sequence downloaded into their memories by the control processor. The microengines, the control units, and I/O interfaces are connected via an internal chassis bus, and data movement happens in an efficient manner through arbitration schemes. This is akin to a system-on-chip design.

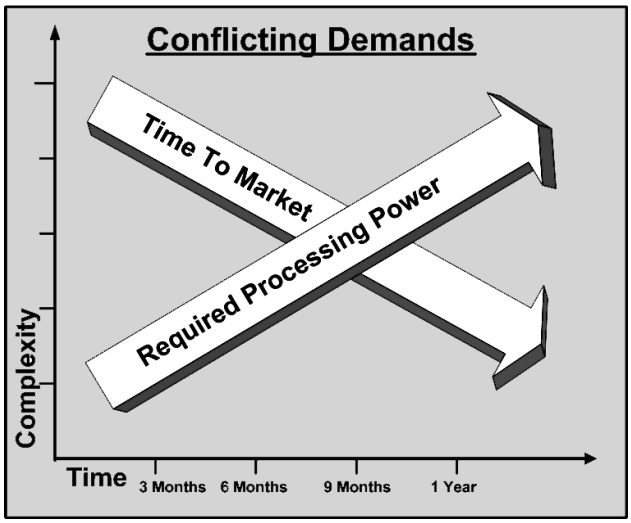


Figure 1: Constantly increasing processing power is required even as the market requires shorter design cycles and time-to-market

A number of challenges for implementation are already evident. As the demand for performance scales up (Figure 1), the number of transistors increases with collateral increases in power and physical design complexity. As an example, the IXP2400 design on p859 packs ~60 million transistors, which is comparable to a high-performance IA32 processor. Further, to meet the performance goals, the critical processing elements such as the microengines and the XScale™ control processor are working at the highest clock rate possible (600MHz in IXP2400). The high-frequency operation requires custom design of Arithmetic and Logic Unit (ALU) and memory elements.

Given the varied usage models in the field, extended temperature range (-40deg.C to 85deg.C) is a Plan Of Record (POR) for network processor implementation at Intel. The power requirements are also stringent, with applications ranging from fully enclosed boxes, as in cellular base stations, to heat-sink solutions on high-performance blades (line cards) in a rack system. These requirements pose a significant challenge for reliability and robustness of the design.

Since network processors have to interface to a lot of I/Os, the result is complexity of package, I/O design, and board design. The requirements on design may be more stringent here in network processors than on a CPU, due to the proprietary nature of designs from OEMs.

Functional verification of a network processor is also a very challenging task due to the fact that it has several interfaces (PCI, DRAM, QDR, slowport, media, switch fabric, etc.) and supports several network protocols. Several on-chip clock domains, both synchronous and asynchronous, make the task even more complicated.

One of the essential deliverables of a network processor design project is a simulator that models the functionality of the network processor with execution-cycle-level accuracy. To external customers and internal software teams, this simulator enables application software development and performance optimization long before the network processor product becomes available in silicon. To the internal design team, this simulator facilitates conducting performance analysis and architecture/microarchitecture studies. The unique nature of network processors poses significant challenges and imposes special requirements on the development of such a simulator. The requirements for the simulator are best illustrated by reviewing the architecture of network processors and the complexity and paradigm of the application development. The simulator must minimize the application development complexity. Furthermore, due to lack of network processor performance benchmarks, the simulator and reference applications must become available at least three quarters before silicon sample date. This schedule enables the potential customers to evaluate the capability of the network processor effectively, and facilitates the committed customers to gain time-to-market advantage by starting application development early. It helps Intel to engage the customers with an architecture before the silicon is available.

In a competitive environment of network processor silicon solutions, Time-to-Market (TTM) becomes a compelling factor for Original Equipment Manufacturers (OEMs) in picking an architecture for product design. For silicon providers, as the performance demand is increasing and the transistor count is thereby increasing, the RTL to GDS2 design cycle times are staying flat. One way to form-fit the complexity of design into the same schedule is by increasing the size of design teams. Studies show that this leads to inefficiencies and increased cost of product development beyond a point.

In response to these challenges, the network processor design teams at Intel have focused on increasing productivity and efficiency in design through reuse, co-development, innovative methodologies, and streamlined tool flows.

The following sections describe IXP2400 as a case study, going into the details of each phase of the network processor design from RTL to GDS2.

IXP2400 DESIGN: A CASE STUDY

Reuse

The strategy for front-end RTL development was co-development with IXP2800 and reuse of design modules from an Intellectual Property (IP) repository. The design modules for RTL coding were partitioned between IXP2400 and IXP2800 at the beginning of the project execution. The originating project that developed a functional block assigned primary logic owners who are completely responsible for functional correctness. The receiving project assigned secondary owners who are responsible for physical implementation at their end. This co-development was managed well through good interaction at engineering-peer-to-engineering-peer level and between management at the schedule level. Project-specific sub-modules were clearly identified (e.g., reset module). For these sub-modules, a common interface was worked out ahead of time to make it an easily swappable block of code. The least common denomination of memory elements in size and usage was also worked out in this manner. High-performance custom datapath blocks in the microengine were isolated from synthesizable blocks with clear interface partitioning to allow parallel development. It is to be noted here that the IXP2400 and IXP2800 have different process and performance goals. Therefore, the sharing is limited to RTL code.

A number of other functional blocks and sub-blocks have been reused from an IP repository. These include the PCI core, Universal Asynchronous Receiver/Transmitter (UART), XScale™ core (Elkhart) and Double Data Rate (DDR) I/O as shown in Figure 2.

Several IP were harvested for the physical implementation for reuse. For the I/O design, good inventory of IP was available from chipset groups for DDR I/O and a basic I/O buffer design for all the others: Media Switch Fabric (MSF), PCI, and miscellaneous I/O. Quad Data Rate (QDR) I/O was generated by modifying the DDR I/O design. Much of the I/O effort then was focused on integrating the I/O blocks for the chip floorplan and for doing signal integrity checks for the board reference designs.

The analog high-frequency Phase Locked Loop (PLL) design was imported from the chipset group and tuned extensively to IXP2400 requirements.

The basic cells for the SRAM memory elements and the SRAM architecture were reused from a CPU group. This cut down the development time to two quarters.

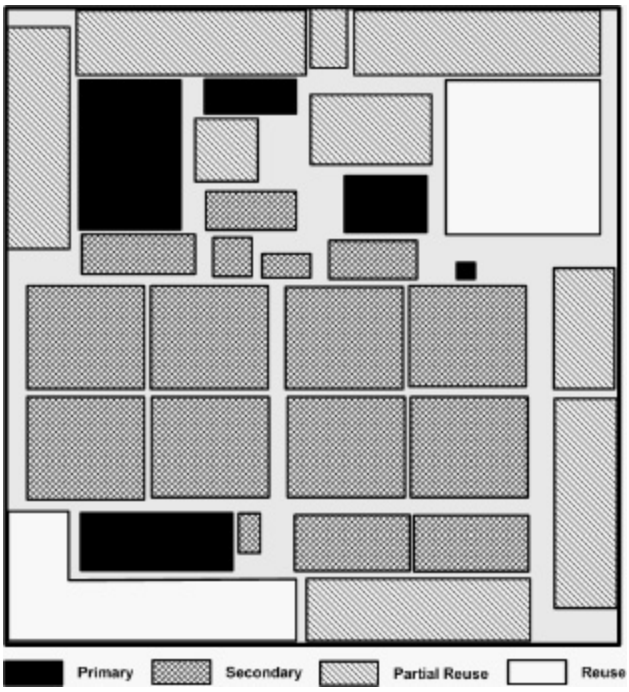


Figure 2: The IXP2400 design is a mix of reuse and co-development

Functional Verification

This section describes some of the methodologies the team adopted to successfully complete verification of the IXP2400 network processor.

Verification of the IXP2400 started with choosing the right tools and verification platform, and defining the verification methodology. The team evaluated several alternatives and chose Cadence's* NCSIM for logic simulator, Verisity's* SPECMAN for test bench automation, X86 Linux platforms for computing servers, Debussy* waveform viewer for debug, and Denali* memory models. A clear methodology was defined and documented with two main goals:

1) The primary goal was to make sure that A0 silicon had adequate functionality that enabled the team to build a system-level environment and run software.

2) The secondary goal was to enable customer sampling on the A-dash stepping.

An extensive upfront methodology was documented with various milestones and exit criteria for each of these milestones. This methodology document defined the rules and guidelines to be followed while developing the verification components to make them extendable, expandable, and readily usable as an IP by another project. IXP2400 and IXP2800 projects used this methodology and

seamlessly shared the verification components. Figure 3 shows the validation view of the Sausalito architecture.

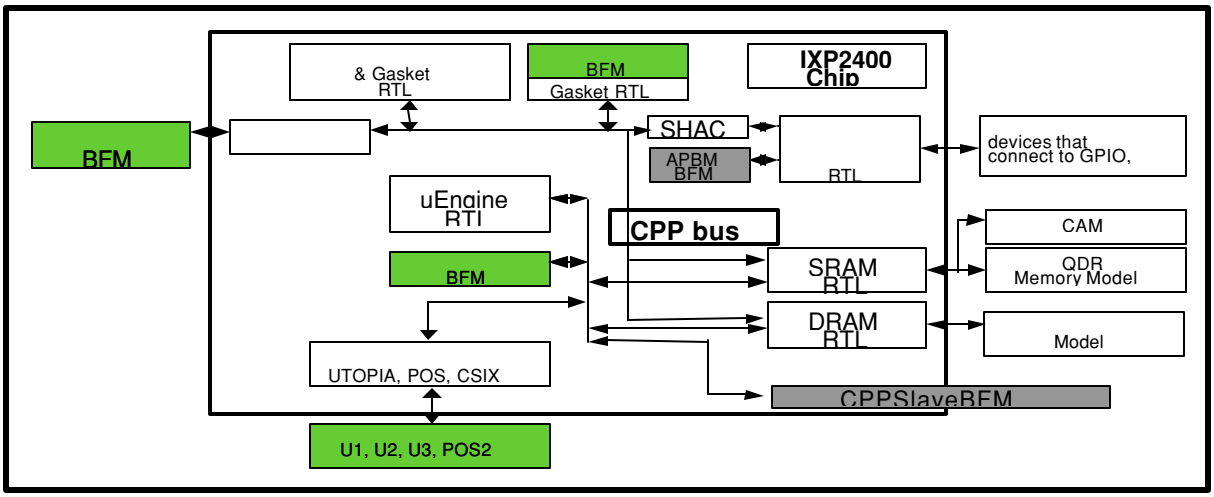


Figure 3: Validation view of IXP2400 architecture

Following are some of the salient features of the IXP2400 verification methodology.

1. Testing design at multiple levels of integration, namely, block level, full-chip, and system levels: System level simulation puts together an IXP2400 full-chip RTL model with the ecosystem surrounding the chip in some real-life applications. The intent of system-level simulation is to make sure that the chip is compatible with ECO system components such as framers, and compliant with industry standard protocols such as UTOPIA and POS-PHY.

2. Monitor-Based Testing (MBT): The real power of SPECMAN lies in its built-in random generator, and this power is used during test plan implementation. Tests are developed in a three-phase approach. In phase 1, simple, directed tests are written to cover the breadth of design. In the second phase, the random power of SPECMAN is unleashed to generate interesting test cases. For each of the test cases in the test plan, monitors are written to make sure that the test case is covered. Hence, all the test cases whose monitors got triggered during this random run are checked off. The coverage report is analyzed to identify the test cases that are not covered. These uncovered test cases are the focus of the third phase, in which directed tests are written to cover them.

3. Random testing: To increase confidence in the model, SPECMAN's random generator was put to use. A concurrent random test environment was built to generate random transactions on a bus with multiple masters and slaves. This environment is highly configurable to choose specific master(s), slave(s), and type(s) of transactions.

4. Gate-level verification: This was used to weed out initialization deficiencies and synthesis bugs.

5. Error checking that uses three methods: 1) extensive score-boarding techniques were used in packet generators; shadow memory techniques were used for memory data checks during and at the end of each test, 2) for microengine verification, a reference model was written in C and was used to validate the RTL model, and 3) protocol checkers were used to verify the behavior of the design for compliance with certain protocols.

6. Structural and functional coverage monitoring techniques.

7. Automation scripts and web-based regression methodology: these used netbatch tools to balance and distribute jobs across multiple servers.

8. Complete debug: the verification team took a goal to debug the RTL failures in order to determine the root cause. In several cases, the team not only root-caused the failure but also identified the fix. This enabled the team to gain extensive knowledge of the design, which helped during later stages of per-silicon debug and also helping post-silicon debug.

9. Rigorous quality and progress measurement indicators: various indicators were developed to measure the quality and progress. The two kinds of indicators used were 1) trend and 2) snapshot. Trends were useful for determining progress against the plan over several weeks. Snapshots were used to get the status at a single point in time and were used to point out problems in a specific block or area of testing.

IXP2400 CLOCK ARCHITECTURE OVERVIEW

To support operations of various memory interfaces, communication interfaces, and internal computing hardware, IXP2400 has 16 clock domains, not to include various test clocks. The highest clock rate is used by a microengine and the on-die XScale™ core, up to 720MHz, to generate high-computing performance for packet processing. The global communication buses for major internal hardware units operate up to 360MHz. To assist sending and receiving data to external memory devices, including both DRAMs and SRAMs, 1X, 2X and 4X clocks are generated for the memory device controller and IO devices. IXP2400 also has four independent media interface clock regions, operating from 25MHz to 125MHz. The clock rates for communication interfaces and memory interfaces are all programmed through control registers. For boot up and host processor communications, IXP2400 has PCI and slow port interfaces running up to 66MHz and 60MHz, respectively.

To support all these clock domains, IXP2400 has a total of 5 Phase Locked Loops (PLLs). Four of them are for generating independent asynchronous clocks for the four media communication interfaces, and the remaining one is for generating all internal clocks for packet processing and all memory interfaces (Figure 4). With various clock domains, data crossing is done through extensive use of a stepping stone control scheme to ensure safe data crossing, in the presence of higher clock skew between different clock domains. Stepping stone control is also enforced in test mode between clock domains that are normally asynchronous in nature.

The clocking in IXP2400 has numerous features to support testing and debug. A debug counter that counts up to 67 million cycles is incorporated to support a count-down and clock-stopping function, so that the device can stop at a particular cycle and SCAN out of internal states can begin. The clock distribution system supports bypassing of external SYS_CLK, SCAN clocks, and JTAG clock.

IXP2400 Clock Design

The IXP2400 clock design can be grouped into two parts: the clock generation and the clock distribution.

The clock generation consists of a PLL and a clock divider. The PLL is a leveraged IP that we adapted to fit into the IXP2400 area constraint. The divider was custom built by the IXP2400 team to meet the more stringent requirement of low latency by the IXP2400 chip. Lower latency means lower full-chip clock skew. The reduction of the full-chip clock skew from the divider is estimated at 60ps. To design a fast divider, non-critical paths were carefully designed to still meet their timing yet, more importantly, have minimum impact on or even help speed up critical paths. Logics were combined innovatively and carefully optimized using custom techniques. As a result, the divider clocks are generated after just one latch delay from receiving the source clock.

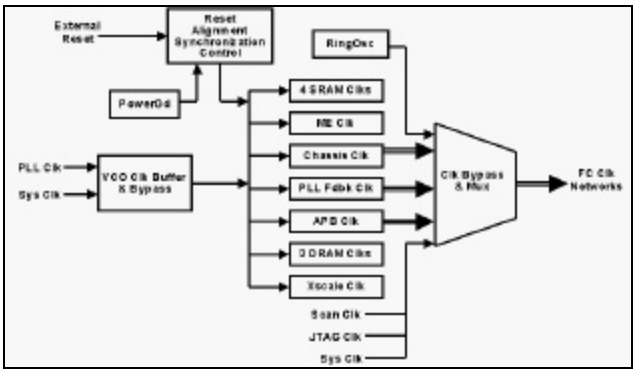


Figure 4: The IXP 2400 clocking scheme

Clock balancing within the divider was made more challenging; given the quest for low latency, more design efforts on balancing were spent after low latency was achieved. Layout was also carefully scrutinized for balancing. A local pre-divide grid was used to reduce RC. Delay elements were added to allow further fine-tuning of the clocks, if necessary. The frequency range of the clocks on IXP2400 is wide; thus a high-ratio divider was designed. An important part of the divider is the error-

correction circuit, which combats noise and ensures clock alignment. The clock dividers are also programmable.

The clock network design challenges were evident from the beginning. The number of clocks in a network processor chip is high compared with general-purpose microprocessors. In addition, the IXP2400 has a large die size; so, inherently, the clock skew would be large if not carefully designed. Low power was another consideration. Tight schedule was another challenge, and quick turn-around time was another goal. These were some of the challenges in designing the IXP2400 full-chip clock network.

For low-power consideration, a balance tree clock network style was selected. It uses a fixed route to stabilize RC and reduce iteration impact to full-chip layout. The large number of clock drivers is grouped into clock station macrocells. Layouts were done with easy programming in mind. Most of the clock stations were designed to drive a fixed load to ease clock tuning. At the chip level, all clocks were routed with shielding. The full-chip clock network RC was extracted and simulated in SPICE. Scripts and automation were developed to quickly tune the clock networks once the block-level clock data are in. Eventually, towards tape-out, the full-chip clock network tuning turn-around time is just one day. At the block level, a pre-grid clock scheme was used to reduce clock skew. RC extraction data were fed back from the block to the full chip for top-level clock tuning. Block-level-clock tuning was automated for smaller blocks and carried out by hand for large blocks and I/Os. SPICE was the primary simulation engine for accuracy of the results. Place and route blocks were tuned by a clock tree synthesis tool for the last two stages of clock networks just before reaching the flops. Clocks lines were plotted and reviewed by the clock owner and the individual block designers.

IXP2400 HIERARCHICAL DESIGN METHODOLOGY AND FLOW

The primary requirement on the design flow is to enable short Time-to-Market (TTM). Therefore, it is imperative to employ a high level of design automation to increase productivity. We employed the following basic strategies to help achieve the TTM goal:

Top-down-driven hierarchical design flow.
Cell-based methodology.
Streamlined custom circuit-design flow.

We began by performing careful floorplanning at the chip level to obtain an accurate wire model of major/critical signals/busses, block sizes/placements, and location of the block-level pins. A full-chip timing budget was then done to allocate timing constraints to the blocks. The block-level constraints were then passed on to the block designers, who then performed an initial design and provided the feedback to the full-chip designer. By performing upfront planning and getting early bottom-up feedback, we reduced the number of iterations needed to converge on the final design goals.

Secondly, through the use of cell-based methodology with only static CMOS logic, we were able to take advantage of the industry standard Application-Specific Integrated Circuits (ASIC) design tools, namely, logic synthesis and automatic place and route tools, which offer a relatively fast design cycle. These tools were used on most of the blocks, with the exception of the timing critical datapath blocks of the microengines, memory arrays, and IO pads. In addition, we were careful to ensure that flip-flops were used at all block boundaries to minimize inter-block interactions.

The key enabler for achieving high productivity in our custom design flow is a tool suite from MicroMagic (now part of Juniper* Networks). Coupled with the cell-based design methodology, the tool allowed us to specify the relative placements of the cells while composing the schematics of the datapath blocks. The tool automatically generates the block layouts with placed cells. Timing analysis is then done with global routings to obtain the performance of the physical design. The placements of the cells can then be fine-tuned to improve timing where necessary. Once the timing goal is met, an automatic detailed router is used to complete the layout. In this manner, the datapath block layouts were completed with less than one-third the amount of effort compared to full custom-design methodology. Likewise, the memory arrays were constructed through the use of MicroMagic tools, which automatically assemble the array layouts based on a set of high-level commands. Furthermore, the command scripts were parameterized so that arrays of different sizes (within a pre-defined range) could be compiled automatically with minimal efforts.

Last, but not least, the design project was managed using a structured design flow with a series of discrete milestones. The flow diagram in Figure 5 depicts the high-level view of the design flow.

*Other brands and names are the property of their respective owners.

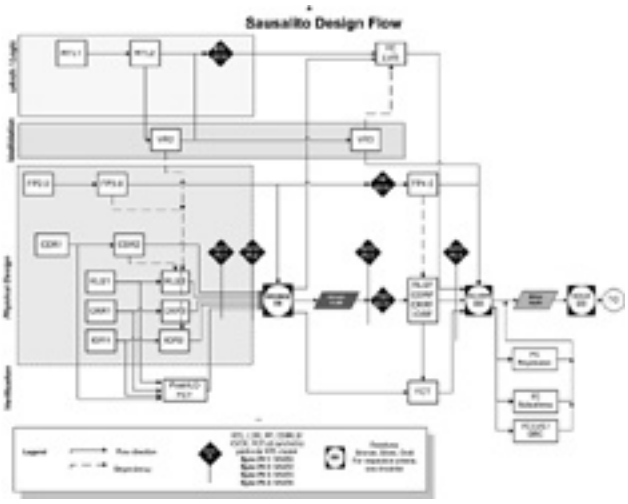


Figure 5: The IXP2400 team used this flow to rapidly converge on the final implementation while maintaining control of the data

IXP2400 Design-for-Test Methodology

IXP2400 is a complex System on a Chip (SOC) design with more than 300 embedded arrays and 88 scan chains encompassing 120K flip-flops spanning across 14 different clocks running and configurable from 33 to 600MHz. The complexity of the system requires a carefully planned Design For Test (DFT) methodology to enable manufacturing and silicon debug. To achieve high test coverage, full-scan design methodology is used throughout the entire chip. Embedded memory arrays are tested using either memory built-in self-tests (MemBIST) or scan-collars. In scan-collared arrays, scan flops are placed on both the input and output stages and are used to control and monitor the arrays. All of the scan-collared arrays are part of the 88 scan chains. Also, boundary scan is implemented on the IO pads to facilitate system-level testing.

To assist in silicon debug, we included a novel scan-debug feature on the chip. This feature allows the chip to run at full speed from reset and stop at a user-defined cycle. The internal state of the machine, i.e., the content of each scanned register, can then be shifted out through the scan chains.

Network Processor Cycle-Accurate Simulator

This section describes the challenges, the requirements, and the successful development strategy and tools that the IXP2400 development team employed.

Architecturally, a network processor consists of clusters of packet processors, co-processors with specific functions, on-chip memory, various kinds of memory and bus interface controllers, and many mechanisms that provide fast communication and signaling among these hardware elements, and a general-purpose CPU, all on a single chip. In the case of IXP2400, the packet processors are called microengines. A microengine is a multi-threaded processor that excels in processing packets at line rate. Each IXP2400 microengine supports up to eight threads of execution. Thread switching is controlled by software and poses zero cycle penalty. IXP2400 contains integrated SRAM and DRAM controllers. Moreover, IXP2400 offers hardware acceleration for managing queues and First In First Out (FIFO) rings, and supports atomic operations for the SRAM and on-chip memory address spaces. Furthermore, IXP2400 provides highly flexible network media and switch fabric interfaces for receiving and transmitting packets.

From the user's perspective, application development for network processors is an exercise of real-time, multi-threaded, and multi-processor programming at the same time. The performance of the application must ensure that the throughput of packet processing exceeds the desired line rate so that packets do not get dropped. In order to create optimized and efficient applications, developers must account for the latency and sequence of all the transactions, as well as the interactions among the various execution threads and hardware units. As a result, the simulator must offer both functional and cycle accuracy. Furthermore, the simulator must monitor a rich list of performance statistics and all the transactions every clock cycle, and must enable the developers to visualize them through an effective Graphical User Interface (GUI).

For the Intel IXP family of network processor products, the simulator is called Transactor, and the GUI tool is called Workbench. Workbench offers the single GUI for code development using assembly or microengine C language, for running simulations to debug and performance-tune applications, and for debugging with the real network processor hardware.

The complexity of network processors, the requirement of 100% cycle accuracy, and the fact that external Transactor releases begin during the early phase of the project all pose significant challenges to the Transactor development team. In addition, the team must achieve excellent development efficiency and quality.

The development strategy that the IXP2400 project employed is based on an internal tool called VMOD. Conceptually, VMOD accepts a logic design at the RTL level and generates the corresponding cycle-accurate C++ model, i.e., Transactor. Moreover, this C++ model supports an API for interfacing Transactor with the workbench. This API relays user commands to Transactor and facilitates communication of model states,

performance statistics, and status of transactions under simulation between Transactor and Workbench.

RTL code presents the functional and cycle-count behavior of a logic design to VMOD. However, RTL code describes only the low-level hardware and does not convey model states at the architecture level. For instance, Transactor users operate with architectural registers, but a register in RTL may be a group of flip-flops that are individually addressed through signal names with long hierarchies. In addition, the RTL code of a logic design does not include performance statistics and does not monitor transactions that execute on top of the hardware that the very RTL code models.

In order to enable Transactor to present architectural states and performance statistics to the users, and to track all the simulated transactions, VMOD accepts C++ code in addition to RTL. This C++ code can read and write individual RTL signals and runs in lock-step with the simulation of the RTL. During simulation, this C++ code collects performance statistics and tracks all the transactions by accessing the relevant RTL signals. Moreover, when the Transactor user wants to access an architectural state, this C++ code translates the mapping of the requested architectural state to the actual collection of RTL signals that make up the architectural state.

Within the IXP2400 design team, the Transactor team owns the development of the Transactor, and the logic design team owns the development of the RTL model. The Transactor team develops the C++ code for all the architectural states, performance statistics, and tracking of transactions. In addition, the Transactor team inputs both the C++ code and the RTL of the logic design into VMOD for Transactor generation. In addition, to ensure excellent quality, the Transactor team builds a thorough regression suite for validating Transactor.

RESULTS

Reuse

Reuse has been a tremendous win overall for the IXP design program. In particular for IXP2400, it cut down the development times for critical elements of design, for example, in I/O and clock design. Co-development of RTL in the front-end has helped IXP2400 and IXP2800 to synergize and develop designs that are completely compatible from the microarchitecture level to the cycle-accurate models on simulators. The sharing of knowledge and resources helped to avoid duplication of effort and kept the design cost low with beneficial affect on the time-to-market schedule.

Pre-Silicon Verification Effort

Sausalito RTL verification was done very efficiently by following a robust methodology. The team completed the pre-silicon verification in approximately nine months after the first RTL model. Sharing verification across IXP2400 and IXP2800 was very beneficial. Quality was never compromised in the verification effort. Results of the work are as follows:

> No functional bug escaped pre-silicon verification after seven weeks of extensive testing on three platforms, namely, the Omaha validation platform, the Angel Island evaluation platform, and the IX/IMS testers.
>
> Though the two projects IXP2400 and IXP2800 used different simulators and validation platforms, the pre-defined methodology and guidelines allowed then to share the verification components seamlessly.
>
> Using Linux machines saved several hundreds of thousands of dollars to the division, and it proved that risk-taking pays off.

Clock Architecture

A low clock skew well-balanced clock architecture was achieved through the methodology at the end of the IXP2400 design. The clock tuning also converged rapidly at the end. The silicon probing confirmed the simulated results.

Hierarchical Flow and Methodology

The high-end ASIC design flow that emerged from the IXP2400 design flow enabled rapid physical design convergence at the end of design. The time from RTL closure to the GDS2 database freeze for tapeout was less than one quarter.

Using the hierarchical design methodology and flow described in this paper, the IXP2400 design team was able to implement and complete the chip design in 12 months. This represents a tremendous achievement, considering the complexity and performance level of the chip. The design flow is further validated by a functional first silicon. This proves that a high level of quality is possible with the TTM design flow.

Transactor

With this effective development strategy and the capability of VMOD, the IXP2400 Transactor project has been on schedule since the first external SDK release, which happened more than three quarters before the IXP2400 sample date. Moreover, architects successfully

completed performance analysis by developing reference applications and validating that IXP2400 meets the performance goals during the chip design phase by using Transactor.

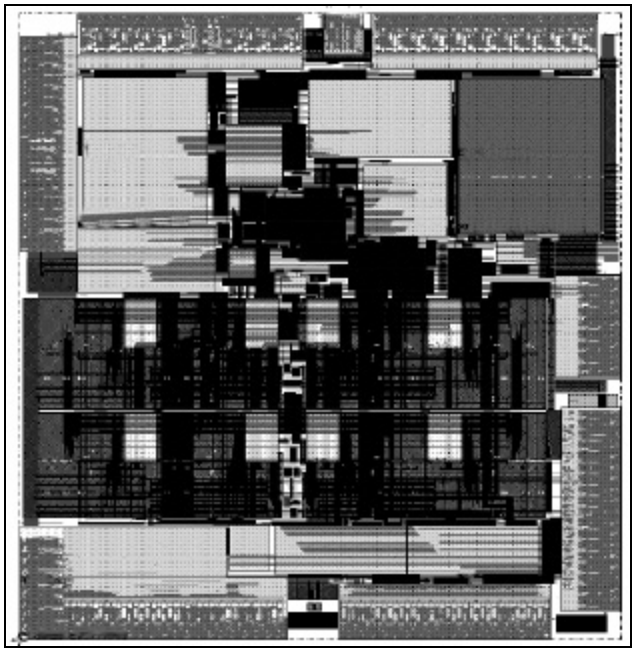


Figure 6: IXP2400 die plot

CONCLUSION

In an emerging and competitive environment of network processor solutions, it is imperative to keep the customers engaged continuously. This interaction starts for the design team with providing an accurate simulator months ahead of time to the actual functional silicon availability. It is also essential to keep the network processor development times on the scale of Moore's law or face extinction.

The network processor design teams have embraced the best-of-class practices to manage the unique design challenges and deliver the products in line with customer expectations. The IXP2400 design (Figure 6) was completed in four quarters from the Implementation Plan Approval (IPA), a goal set at the start of the project. The A0 post-silicon was obtained on schedule. After one quarter of extensive testing on three different platforms, no functional issues have been found. The first customer samples, based on A0 silicon, were shipped out one week ahead of the plan established at IPA.

ACKNOWLEDGMENTS

The authors acknowledge the contributions of Suri Medapati, Tim W. Chan, Jianhui Huang, and Kamal Koshy. The authors also acknowledge the contributions of Bill Wheeler, Chris Clark and Tim Fennell in the deployment of VMOD tool for IXP2400.

REFERENCES

[1] C. Narad and L. Huston, "Introduction to Network Processors," Hotchips-12 Presentation, August 2000.

[2] S. Batzer, et. al, "Modeling the Cost Avoidance Potential of a Structured Approach to IP Reuse at Intel," *DTTC* papers, July 2002.

AUTHORS' BIOGRAPHIES

Ram Bhamidipati joined Intel in 1989, after completing his M.S. in Electrical Engineering from N.C.A.&T. State University. He has worked on processor design groups for the development of i486™, Pentium® II, and Itanium® processors. He holds two US patents in design. Currently, he is managing the back-end design of the IXP2400 network processor. His e-mail address is sriram.bhamidipati@intel.com.

Ahmad Zaidi joined Intel in 1987, after completing his Master of Electrical Engineering degree from Virginia Tech University. Ahmad is currently Director of Silicon Engineering for NPD-San Jose, focusing on architecture, design, program management, and manufacturing of network processors for the Access and Edge market segments. His prior assignments include engineering management positions on the Itanium Processor, and engineering positions in the i386™, i486, and Pentium® microprocessor projects. Ahmad holds nine US patents in microprocessor design and architecture. His e-mail address is ahmad.zaidi@intel.com.

Siva Makineni joined Intel in 1992, after completing his Master of Engineering (E.E.) degree from Worcester Polytechnic Institute in Worcester, Massachusetts. Prior to joining the IXP2400 team as pre-silicon verification manager, Siva held several engineering and management positions in Itanium, Pentium, and 486SL projects. Most recently, he designed floating point arithmetic units on the Itanium processor and managed the SIMD floating point implementation team. Siva holds ten US patents in floating point and integer arithmetic. His technical interests include high-speed floating point architecture and design, computer arithmetic, and developing effective verification strategies. His e-mail address is siva.makineni@intel.com.

Kah K. Low is currently the design center manager of Intel's Malaysia Network Development Center, where he leads the development of next-generation network processors. Previously, he was the global design manager in the IXP2400 design project. He joined Intel in 1995 to work on the Itanium design project, where he managed the circuit design automation group. Prior to Intel, Kah K. was with Motorola, Inc. from 1989-1995, where he worked on statistical design, device modeling/characterization, CAD, digital signal processors, and where he served as a project manager in SEMATECH's phase-shifting mask program. He holds three US patents and received his B.S. degree from the University of Massachusetts, and his M.S. and Ph.D. degrees from Carnegie Mellon University, all in Electrical Engineering. His technical interests include network processor design, VLSI design methodology, CAD tools, and communication networks. His e-mail address is kah.k.low@intel.com

Robert Chen received his Ph.D. degree in Electrical Engineering from the University of Notre Dame in 1993. Since then, he has worked in various areas of IC design: library, SRAM, register files, TLB and CAM, low power, clocks, place and route, and logic design. He received "Top Gun Award" from Sun Microsystems, Inc. in 1995, and "IA-64 Processor Division Award" from Intel in 2000 for his work on McKinley power reduction. Currently, Robert is working on the clock design for the next generation of IXP2400. His e-mail is robert.chen@intel.com.

Kin-Yip Liu joined Intel in 1990, after completing his Master of Engineering (E.E.), Bachelor of Science (E.E.), and Bachelor of Arts (Economics) degrees from Cornell University. Kin-Yip now co-manages the NPD NPBU Architecture team at San Jose, focusing on network processors for the Access and Edge market segments. His prior assignments include engineering and management positions in the Itanium Product Family architecture and firmware teams and in the 386SL and 486SL microprocessor projects. Kin-Yip holds four US patents in microprocessor architecture. His technical interests include network processing, computer architecture, and simulator development. His e-mail address is kin-yip.liu@intel.com

Jack Dahlgren joined Intel in 1997, after ten years in the architecture and construction management industries. He has provided project control services on several projects including development of Itanium and the Mobile Intel® Pentium® III Processor. His educational background includes Master's degrees in Architecture and Civil Engineering from the University of California at Berkeley. His e-mail address is jack.dahlgren@intel.com

For further information visit:

developer.intel.com/technology/itj/index.htm