Patrick Gelsinger, Desmond Kirkpatrick, Avinoam Kolodny, and Gadi Singer

Such a CAD!

Coping with the complexity of microprocessor design at Intel.

PICTURED, FROM LEFT, ARE PATRICK GELSINGER, GENE HILL, AND ALBERTO SANGIOVANNI-VINCENTELLI. PHOTO COURTESY OF INTEL

During the 1980s, Intel Corporation transformed itself from a semiconductor company producing memory chips into a computer company [1]. Intel's transformation was actually part of a revolution in the entire electronics industry: In the beginning of the decade, microprocessors were considered to be toys, and the computer industry was dominated by mainframes and minicomputers made by vertically integrated companies. By the end of that decade, microprocessors became the standard engines for computing platforms, and the whole industry was restructured. Many more vendors entered the industry, each specializing in different areas.

These changes were fueled by the continuous scaling of MOS technology, following Moore's law. Interestingly, in his original 1965 paper (reprinted in 1998 [2]), Gordon Moore expressed a concern that the growth rate he predicted would not be sustainable because the process of defining and designing products in the context of such rapidly growing complexity would not be able to keep up with his predicted growth rate. The highly competitive business environment, however, drove companies to fully exploit technology scaling. The number of available transistors doubled with each new generation of process technology, which came on line roughly every two years. As shown in Table 1, major architecture changes in microprocessors were brought about by an approximately fourfold increase in available transistor counts every second generation. Intel's microprocessor design teams had to come up with ways to keep pace with the size and scope of every new project.

This incredible growth rate could not be achieved by hiring an exponentially growing number of design engineers. It was fulfilled by adopting new design methodologies and by introducing innovative design automation software at every processor generation. These methodologies and tools always applied principles that raised design abstraction, becoming increasingly precise in terms of circuit and parasitic modeling while simultaneously using ever increasing levels of hierarchy, regularity, and automatic synthesis. As a rule, whenever a task became too painful to perform using the old methods, a new method and associated tool were conceived for solving the problem. This way, tools and design practices were constantly evolving, always addressing the most labor-intensive task at hand. Naturally, the evolution of tools occurred bottom-up, from layout tools to

Digital Object Identifier 10.1109/MSSC.2010.937691

circuit, logic, and architecture. At each abstraction level, the verification problem was typically the most painful; hence it was addressed first. The synthesis problem at that level was addressed much later.

This article is the story of the coevolution of design methodologies, practices, and CAD tools in Intel's design environment as it coped with increasing complexity in the turbulent 1980s and up through recent years. It is interesting to note that at the beginning of this process the engineering culture was advocating a tall, thin design. Nowadays, very large scale integration (VLSI) engineers are highly specialized in different areas of the design discipline, where specialized tools are used in each area. This is analogous to the restructuring of the whole computer industry from vertical to horizontal.

In the 1980s, the CAD industry itself was nascent at best. While some areas like schematic or layout entry had solid commercial offerings, the rapidly evolving complexity of this young industry meant there could be little hope from commercial tool offerings. Therefore most tools emerged from internal development, external university research, or often a coevolving blend of internal work with external tools and research. While there were a number of corporate-university relationships at that time, none was as prolific as that of Intel with the University of California, Berkeley. In particular, Alberto Sangiovanni-Vincentelli and his collaborative research team, which consisted of Robert Brayton, Richard Newton, and many graduate students, had developed a strong partnership with Intel and its microprocessor teams. This long partnership with Intel stands as one of the most fruitful relationships in EDA, bringing fundamental breakthroughs in multiple elements of microprocessor logic, synthesis, and layout. Many of these early successes resulted in enormous benefits to Intel and eventually made their way into the EDA industry as key enablers of many EDA tools and today's fabless ASIC/SOC semiconductor industry.

TABLE 1. INTEL PROCESSORS, 1971–1993.

PROCESSOR	INTRO DATE	PROCESS	TRANSISTORS	FREQUENCY
4004	1971	10 μm	2,300	108 KHz
8080	1974	6 μm	6,000	2 MHz
8086	1978	3 μm	29,000	10 MHz
80286	1982	1.5 μm	134,000	12 MHz
80386	1985	1.5 μm	275,000	16 MHz
Intel 486 DX	1989	1 μm	1.2 M	33 MHz
Pentium	1993	0.8 μm	3.1 M	60 MHz

Design Environment for the Early X86 Processors

Inherited Tools from Memory Chips

Intel's initial design environment was formed to serve the needs of memory chips. During the 1970s, the primary CAD tools were layout capture and verification tools, used by draftsmen to generate and check mask layouts. These tools were put in place because the layouts were already too complicated to develop and maintain solely on paper or Mylar. Polygon-based layout representations therefore had to be stored and handled by computerized tools, initially on dedicated systems such as the Calma or Applicon.

Engineers were doing circuit and logic designs at the transistor level, usually by hand, producing hand-drawn schematics at the transistor level for the layout designers. The engineers did most of their design work using pencil and paper, but they also had circuit simulation tools derived from the industry-standard SPICE [3] program. SPICE originated in Don Pederson's group at Berkeley and later on was refined by Richard Newton, Alberto, and their students (Intel's version was known as ISPEC). It was possible to simulate and check logic behavior and timing waveforms for small circuits that incorporated up to a few hundred transistors.

As Intel started designing logic products, including the first microprocessors (the Intel 4004, 8008, and 8080), design engineers inherited all of those tools and methods, which had initially been conceived for memory chip design. Some engineers preferred to perform logic design using gate-level schematics, but this generated some resistance from the layout designers. They were familiar with transistor representations, which directly matched the layout. Translating logic gate symbols into transistor structures was not a trivial task because the early microprocessors and numeric coprocessors (8087, 80387) were designed in NMOS technology. Circuit operation relied on device strength ratios, so each gate symbol had to be accompanied by specific transistor sizes. In addition, the prevailing design style supported many complex gate pull-down devices,

This article is the story of the coevolution of design methodologies, practices, and CAD tools in Intel's design environment as it coped with increasing complexity in the turbulent 1980s and up through recent years.

pass transistors for clocking structures, dynamic circuits, and numerous other clever and often "tricky" structures that could not be cleanly represented by a simple logic gate abstraction. Consequently, even logic design was actually performed by engineers at the transistor level (also called the switch level), such that even well-known techniques such as logic minimization by Karnaugh maps, which were taught at engineering schools, were not widely used by VLSI engineers in those NMOS days. The clever NMOS design tricks typically resulted in superior densities, albeit with commensurate complexities.

Evolution of Intel's Logic Design and RTL Modeling

As it became too error-prone to debug logic behavior of processor circuits by hand and too time-consuming to verify the logic behavior by circuit simulation using continuous waveforms, people at Intel were looking for an executable functional model. At that time, the mainframe computer industry was already using gate-level logic simulators, which used variable-delay models for TTL gates (made with bipolar junction transistors). An attempt to adopt logic simulation at Intel resulted in failure: Intel had developed a gate-level logic simulator called LOCIS in the mid-1970s, and the 8086 design engineers converted their transistor-level schematics into an equivalent logic model using LOCIS gate models. The generic gate models of the simulator did not match the tricky MOS logic structures of the 8086 schematics, however, and its gate-delay models burdened users with too many irrelevant timing-related messages and glitch warnings.

Intel's transformation was actually part of a revolution in the entire electronics industry.

After this experience, engineers turned to building functional models with general-purpose programming languages. One of the first register-transfer level (RTL) models at Intel was developed for the 8087 numeric coprocessor in 1978. It was a FORTRAN program that described the logic behavior of circuits, as extracted by human interpretation of the transistor-level schematics. It was used for verifying and debugging the microcode programs stored on the chip.

In the design of the 80286 processor, the starting point was already a functional RTL model. This model was manually translated into schematics in a top-down fashion, rather than the other way around. The model was written in MAINSAIL [4], an Algol-like general-purpose programming language derived from the Stanford Artificial Intelligence Language (SAIL). RTL modeling and simulation by a compiled program in a standard language (where logic propagation between gates is actually assumed to occur without any delay) was made possible because of a strictly synchronous design methodology, with two nonoverlapping clock signals *Phi1*, *Phi2*. During each phase of the clocks, new signal values can propagate in the logic network, and the logic designer only cared about the final, steady-state values that were latched at the end of the clock phase. As a separate task, someone (a circuit designer) had to ensure that the cycle time was long enough for the circuit to reach a steady state in each phase. The RTL program simulated the circuit behavior at a cycle-by-cycle timing resolution by invoking code for each clock phase in turn. This approach was inspired by Mead and Conway's famous book [5]. Today, this approach seems trivially obvious. In that era, however, logic design was typically done in the context of detailed timing-dependent behavior, where both timing and logical function were verified simultaneously. With the synchronous methodology, separation of the functional simulation from the timing issues enabled successful large-scale design and created two kinds of engineers, who could worry about two separate problems: the logic designers, focused on the functional correctness problem, and the circuit designers, focused on transistor sizes, voltage levels, parasitic capacitances, and gate delays. The separation of concerns like this continues to be a powerful mechanism in design automation.

Taking advantage of MAINSAIL's support for the dynamic linking of separately compiled modules, RTL models of large circuit blocks were coded as program modules, and a simulator, μSIM [6], was developed to control and monitor their execution. The first Intel design to use such a scheme was the iAPX 432 chip set, developed in Oregon and released in 1981.

In the 80286 design, the blocks of RTL were manually translated into the schematic design of gates and transistors, which were manually entered in the schematic capture system, which generated netlists of the design. The schematics would be simulated via the switch-level simulator MOSSIM [7] and compared with the RTL design on a per clock, per signal basis. This was a laborious procedure but verified the logical integrity of the RTL with that of the entered schematic design. Design changes were always challenging, as they required the synchronization of the changes into RTL and schematic databases.

There was a separate path for the handful of programmable logic arrays (PLAs). In this case, the PLA functions were optimized using the internal LOGMIN tool, which automated the logic minimization process. The same resulting PLA codes were loaded into the RTL as a macro function and into the schematic system; they were then used to program the PLA arrays into the layout. Much of the early automation in PLA synthesis at Intel was enabled by Alberto's Berkeley research in two-level logic minimization by Espresso [8] and physical automation (e.g., PLA folding [9]) to make large control circuit synthesis using PLAs practical.

Performance Verification

The RTL-based functional design methodology separated the issue of timing from the issue of functional correctness, assuming that synchronous methodology was enforced and that the clock was slow enough for all logic paths to settle to a steady logic state within each clock phase. During this time, however, critical paths were only modestly considered during the design phase. This was largely due to a lack of tools and the engineer's limited knowledge of the design and the likely critical areas. Most of the critical paths were not resolved until they were discovered on silicon. The clock could be slowed down until no critical path failure existed. Then the clock frequency was sped up, but specific clock pulses were extended to help isolate the failing circuit. For example, the 49th clock pulse during the test program could be made longer to allow completion of a slow logic operation somewhere in the chip. This was done using special "clock stretcher" debugging equipment. However, the 286 design had many second sources, and those manufacturers very quickly found clever ways to speed up their designs to rival Intel's. This led to a minor crisis within Intel: the industry was rapidly exerting pressure on the company with respect to the very architecture and design it had created, and the tools to dig into the problem were both weak and laborious.

This crisis triggered the introduction of static timing analysis into Intel and development of the Coarse-Level Circuit Debugger (CLCD) tool [10]. It was a schematic-level analysis tool for electrical rule checking and critical path finding that could discover circuit-level bugs and resolve device-sizing issues. It could also extract the logic functionality of transistor-level circuit structures and represent them using logical expressions. The new capabilities were applied in the next-generation microprocessor, the 386, which was no longer built in NMOS but rather in CMOS.

The 386 Design Environment

In moving to the 386 during 1982, the design team quickly ported the 286 design modules to the 386 design environment as a starting point. In particular, for the complex memory protection model of the 286, some of these blocks would make it to the final 386 with minimal changes. Most of the remainder of the design, however, went through radical changes with the move to a 32-bit data path width and the introduction of the flat paging model. The design work iterated rapidly, with the RTL being the center of the logic design team's efforts. RTL simulation for the first time dominated the overall computing load of the design team as logical correctness became the focus of the team's activity.

With the team focused on RTL design and the substantial complexity increase from the 286, the question was how to more effectively provide the translation to schematics and the logical representation of the chip. In particular, we were looking to accelerate the design process, minimize manual translation errors, and handle the rapidly increasing design complexity. Given these goals, the relationship with Berkeley—and especially with Alberto—quickly became central to our efforts. Albert Yu (manager of the Microprocessor Division) and Pat Gelsinger (then in charge of new design methods in the corporate CAD group) visited Berkeley to explore some of Alberto's research work and its potential application to our problems as well as the possibility of partnering on these challenges.

The meeting focused on topics such as the regularization of layout, the potential use of YACR (yet another channel router) [11], TimberWolf [12], logic synthesis, and the potential for multilevel logic synthesis, where the path between input and output could propagate through several logic gates rather than only two as in a PLA. Albert Yu's idea was that Intel needed to keep a two-year beat to develop a new microprocessor, and he thought that the only way to keep to this timetable was to introduce new tools and methods. Albert and Pat fully appreciated the potential of multilevel logic synthesis and of regular layout. Albert proposed supporting the research at Berkeley, introducing the use of multilevel logic synthesis and automatic layout for the control logic of the 386, and setting up an internal group to implement the plan, though Alberto pointed out that multilevel synthesis had not been released even internally to other research groups

It is important to note that in the 1990s, a very significant productivity gain was achieved by increasing the computational power available to design teams.

at Berkeley. The design manager of the project, Gene Hill, put Alberto on a consulting contract: Alberto would facilitate progress on these issues as well as review the 386's floor plan to better understand the broader applicability of advanced CAD methods to the design.

It is important to note that with the 386, the era of CMOS began at Intel. While we were far from the power wall of the early part of the decade from 2000–2009, NMOS power was increasing at a nearly exponential rate. CMOS brought with it a reasonable P device and a strong bias toward complementary logic structures to eliminate steady-state power dissipation, achieve symmetry between rise and fall times, and get full-swing logic voltage levels regardless of transistor sizes and transition speeds. With NMOS, there was much to be gained from a cleverly ratioed design. While there were still arguments for complex domino-type design approaches, the inherent nature of CMOS design created a strong move toward using a standard set of gates from a cell library rather than the individually sized and customized gate structures that were common in the days of NMOS.

Working with a cell library, we were able to employ Berkeley tools like Espresso for logic minimization and TimberWolf for simulated annealing of cell placement. We were quickly demonstrating large regular blocks of reasonably well optimized logic designs. While the idea of simulated annealing seemed rather chaotic at best, the results were quite good. An oft-repeated lesson in science and engineering is to apply proven techniques from other fields to similar problems in your field. In this case, simulated annealing proved to be the perfect answer. Of course, with ample computing cycles made available on the IBM 3081, one could play with the parameters offered at length to find ever better layout results. After global placement by TimberWolf, specific cell placement occurred in standardized rows of standard cells and routing channels using a tool called P3APR, developed by Manfred Wiesel, who came to Intel from the BellMac project at AT&T.

In fact, the results were good enough that the design team eliminated all the small PLAs from the 286 and simply converted them to interconnected logic gates (i.e., random logic). This made the logic blocks larger, with greater potential for further logic-design optimization. Only the I/O ring, the data and address path, the microcode array, and three large PLAs were not taken through the synthesis tool chain on the 386. While there were many early skeptics, the results spoke for themselves.

A key challenge of applying logic synthesis to our industrial design was the clocking style.

With layout of standard cell blocks automatically generated, the layout and circuit designers could myopically focus on the highly optimized blocks like the data path and I/O ring, where their creativity could have much greater impact. Further, these few large blocks greatly simplified the overall global chip floorplanning effort, allowing for a much more rapid final chip assembly with far fewer errors. An in-house program called CVS, written by Todd Wagner [13], performed verification of final connectivity. While today the 386's 275,000 transistors seem trivial, at the time it was a monumental feat, breaking ground in performance, ISA compatibility, and design methodology.

The 486 Design Environment

The Challenge of Logic Design in the 486

While the 386 design heavily leveraged the logic design of the 286, the 486 was a more radical departure. It involved the move to a fully pipelined design, the integration of a large floating-point unit, and the introduction of the first on-chip cache: a whopping 8-kB write-through cache used for both code and data. Given that substantially less of the design was leveraged from prior designs and the fourfold increase in transistor counts, there was enormous pressure for yet another leap in design productivity. While we could have pursued simple increases in manpower, there were questions regarding the company's ability to afford the employees, find them, train them, and effectively manage a team that would have needed to be much larger than the 100 people who eventually made up the 486 design team.

With this challenge in front of us then, several aggressive goals were proposed to enable our relatively small team to tackle the 486 design:

- a fully automated translation from RTL to layout (we called it RLS, for *RT-to-layout synthesis*)
- no manual schematic design (direct synthesis of gate-level netlists from RTL, without graphical schematics of the circuits)
- multilevel logic synthesis for the control functions
- automated gate sizing and optimization
- inclusion of parasitic elements estimation
- full chip-layout and floor-planning tools.

To execute this visionary design flow, we needed to put together a CAD system that did not yet exist. Once again, we traveled to our (now) good friend Alberto at Berkeley to extend our previous collaboration with new tool development. A liaison from Intel (Gary Gannot) was stationed in Berkeley for two years as a participant in Alberto's research team.

While we were working on the 386, academic CAD research was going through a major renaissance at Berkeley. The original research in CAD there was being combined into the Berkeley Synthesis Project, with a focus on merging the efforts in logic synthesis and layout generation. After collaborating with Alberto at IBM from 1980 to 1982 and a Berkeley sabbatical in 1985, Robert Brayton joined the Berkeley faculty full-time in 1987. The three main CAD professors, Alberto, Brayton, and Richard Newton, then joined forces to begin what became a highly prolific period in CAD. In a tour de force 2003 Design Automation Conference (DAC) keynote speech, Alberto termed this era "the age of heroes," a "vibrant era of creativity and expansion." In hindsight, Alberto and his colleagues fostered strong industrial collaboration with their decision to make the results of the Berkeley research (including software systems) freely available to everyone. Through this arrangement, the close technical collaboration between Intel and the Berkeley CAD group was able to benefit both academia and industry, which in turn fueled further research advances.

As the 486 project was starting in 1986, Gene Hill (the director of microprocessor development) was deliberating whether to take the full risk of devising a new CAD system or to work on a more conservative plan in parallel as well. Gary Gannot recalls: "He asked me if I felt comfortable that the code written by the students at Berkeley would be reliable enough in a production-worthy environment. Since I was proud to be part of the MIS team, I immediately responded that I felt very comfortable." Finally, Hill decided to go for it: he transferred "open requisitions" to hire 15 engineers from his budget for the corporate CAD group. There was agreement among Gene Hill, Albert Yu, and Mike Aymar (who headed the corporate CAD group) that a central methodology development group under Rafi Nave should be formed, with Pat Gelsinger and Jim Nadir at the center of the group. Jim Nadir's primary focus was on library and physical design; Pat Gelsinger was in charge of the methodology and the tools, working closely with the CAD teams in the United States and Israel and with Berkeley and Alberto. He did not expect this at the time, but his next assignment would be managing the 486 design: he was shortly to become the customer for the very tool chain he was driving.

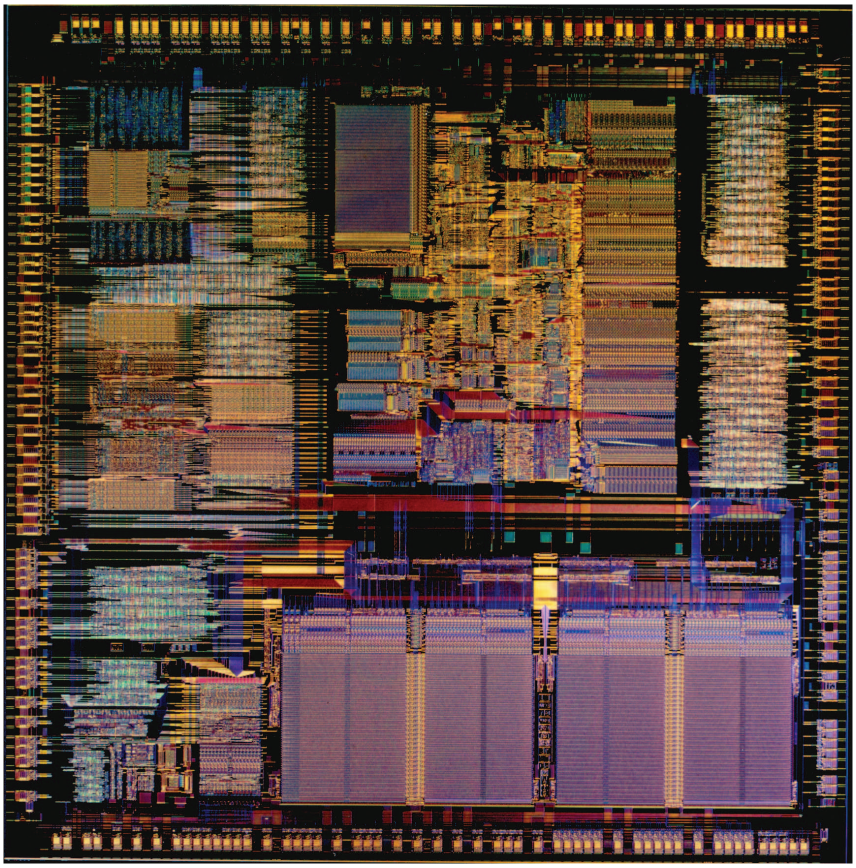
FIGURE 1: The Intel 80386 processor. Taking a clockwise path around the chip: The upper right was bus interface and instruction decode, the lower right was test and control logic and the large microcode ROM, and the lower left was the data path for primary instruction executive. Moving up the data stack on the left of the chip was the segment and virtual address generation. Finally, paging and final physical address generation was in the top left. Synthesized random logic blocks stand out clearly in the middle given their row of cells and routing channel characteristics. (Photo courtesy of Intel Corporation.)

Intel's Hardware Description Language

A major technical challenge we had to overcome for enabling a direct link from RTL to logic synthesis was the input language for RTL modeling. Languages like MAINSAIL and C didn't have the formalism required to describe synthesizable hardware. Languages like VHDL were in the process of being invented at the time but were considered hopelessly complex, given the broad industry process being used to define them.

Thus, we launched the iHDL effort to devise a language defined specifically by the formalism required for synthesis, with clear semantics for items like buses, native algebraic and Boolean logic functions, and the basic control flow mechanisms a logic design requires. The iHDL language defined by Tzvi Ben Tzur, Randy Steck, Gadi Singer, and Pat Gelsinger filled the bill. In a series of summits among the Israel, Oregon, and Santa Clara teams in 1985 and 1986, we converged on a language definition. Meanwhile, the CAD team in Israel was developing the language compiler. The result was a formal language description for RTL development and logic/layout synthesis from that description. Berkeley's creation of a standard intermediate format for logic representation (BLIF) was a key enabler for Intel (and others) to develop higher-level description languages. Amazingly, Intel didn't replace iHDL with Verilog until 2005, simply because of its expressive completeness and effectiveness for synthesis. It had a 20-year life.

Collaboration among Intel, Alberto Sangiovanni-Vincentelli, and the University of California, Berkeley, continues to this day in a broad range of areas of computer architecture, particularly platform-based design.

Intel's First Standard Cell Library

The vision of automatic conversion of RTL to layout hinged also on the existence of a standard cell library. The library cells had to fit multiple tools: they had to have a standard "height" and ports to enable automatic placement and routing. Their delay characteristics had to be modeled for static timing analysis, and the whole library had to serve as input to the logic synthesis tools. Beyond this, a decision was made to develop a single library for use by multiple design teams across Intel and to increase productivity via large-scale reuse and modularity. Given the long history of individual transistor optimization at Intel, obtaining agreement on standard cells was no small task.

Jim Nadir in corporate CAD was given the assignment to create the common cell library, working closely with people at Intel's technology development group in Oregon. This turned out to be one of the more difficult and politically sensitive assignments anywhere in the company at the time, as each project group in the company wanted to have some unique cells. The resistance to a standard cell library sounds absurd today, when libraries are offered to design houses as the basic access interface to semiconductor manufacturers.

Intel's Adaptation of Berkeley Logic Synthesis

We decided that our RLS system would be based on a tool called Multi-Level Logic Interactive Synthesis System (MIS) [14], which was actually an experimental workbench being developed by the graduate students at Berkeley for executing various restructuring operations on combinational logic blocks. Gary Gannot, our liaison, was regularly sending software releases of MIS from Berkeley to the Intel CAD team in Haifa, Israel. The team in Haifa wrote software programs to perform a series of tasks: compile iHDL models into intermediate data structures; decompose the compiled blocks into separate combinational blocks and sequential elements (latches or flip-flops); feed each combinational block into MIS for logic minimization (the output was a network of generic NAND gates); convert the generic form into a combination of actual gates from the library; and combine all the results, along with the sequential elements, into a final netlist that could be handed over to the layout synthesis tools. Berkeley developed the library-mapping step at our request [15], as it was essential for our program.

A key challenge of applying logic synthesis to our industrial design was the clocking style: Intel designs commonly used transparent latches to allow more flexibility in the number of logic levels between state elements. Furthermore, skew penalties apply only once to a loop of transparent latches, rather than to every sampling element (as with flip-flops, where the "hold time" is wasted in each flop). Finally, latches were smaller. Yet this introduced complexity for synthesis in coping with a two-phase clocking system, both at the logic level and during place-and-route. Since design debates raged then (as they do now) about whether to flop or to latch in any given design, our CAD programs had to cope with both. We also needed to address timing, parasitic estimation, and the automated sizing of gates. To do this, we used the internally generated CLCD tool [10]. In addition to CLCD, we also developed a central timing tool called TISS, which managed the global timing signal requirements. Some of these would be generated automatically from synthesis, some would be globally determined by external requirements, and some were highly optimized design signals, such as critical data path signals that were tuned through manual circuit design and layout approaches.

Much of the RLS integration/development effort was done in Israel due to the central role that the CLCD tool played and the relative stability of the other tools in the flow. Pat Gelsinger recalls some of the sensitivity associated with working across a geographical and cultural barrier. He says, "I demanded the CLCD team work directly for me, as I knew how central it was to the overall flow. Mike Aymar, who ran corporate CAD at the time, refused, claiming I needed to learn how to manage indirectly and through influence. I wanted to kill him at the time, knowing the Israelis were tough and remote , and I didn't have time for such nonsense if we were going to pull the overall RLS system off in time for the 486 program to start up on it. It was a valuable learning and development experience for me as a manager, even if I despised Aymar for at least a year for making me live through such a challenging management experience." Aymar put a strong emphasis on continually pulling the teams in Oregon, California, and Israel together. He recalls: "This placed significant demands on people's personal lives, as they had to spend quite a bit of extended time in remote sites from their home site. It worked though, and overall pretty well. In those days I got hooked on e-mail. I remember describing to Andy Grove how amazing it worked in allowing folks to communicate between various sites and time zones. He didn't buy it at the time. This was one of the rare times—maybe the only time—I anticipated the importance of an emerging trend before he did!"

Physical Design Automation in RLS

With the advent of multilayer metal process technologies, layout synthesis became competitive with manual layout artwork. The complexity of creating dense designs made automation more suitable for and acceptable to engineers. At the time, Intel still used manual effort to generate structures that were more regular, such as memories and data paths, but control logic was synthesized both at the logic and layout levels. Place-and-route algorithms came from Alberto's students at Berkeley; TimberWolf used simulated annealing and was directly and heavily used to create optimized placements. While routers were written for industrial use, the algorithms were heavily based on technology from Alberto: the infamous YACR2 algorithm [11] and the Chameleon [16] multilayer approach by Doug Braun (who joined Intel in 1987 and wrote most of the routing compaction algorithms).

The physical design automation software was written in MAINSAIL, as were most CAD tools at Intel at the time, and the team, led by Manfred Wiesel, produced a series of tools. The DAPR standard-cell tool [17], [18] placed and routed blocks of several thousand standard cells in double-back rows (shared power supply) with diffusion sharing, routing over the cell, and double-layer metal technology. For the 486, we had for the first time developed a full-chip floor-planning and assembly tool called ChPPR, which used a 1/2 design rule boundary abstraction to create correct-by-construction abutting block placements, over-the-cell routing, and a hierarchical global abutment and connectivity check that bypassed traditional connectivity verification [13] and that was orders of magnitude faster because it eliminated layout extraction. The ChPPR hierarchical tool was actively used on mainstream microprocessors at Intel for almost 20 years, until 2005.

At the end of the design, Pat, Gene, and Alberto were featured in a video that Intel distributed worldwide to universities.

Complete RLS Flow for Random Logic Synthesis

The combination of all these tools was stitched together into a system called RLS, which was the first RTL-to-layout system ever employed in a major microprocessor development program, although similar synthesis projects were implemented at several other companies in the 1980s. RLS was used only for control logic in the 486 chip, covering the most complex and tedious logic design effort; the highly regular data path was done manually in order to achieve high density and speed.

RLS succeeded because it combined the power of three essential ingredients:

- CMOS (which enabled the use of a cell library)
- a hardware description language, or HDL (which provided a convenient input mechanism for capturing design intent)
- synthesis (which provided the automatic conversion from RTL to gates and layout).

This was the "magic and powerful triumvirate." Each one of these elements alone could not revolutionize design productivity. A combination of all three was necessary. These three elements were later standardized and integrated by the EDA industry. This kind of system became the basis for the entire ASIC industry and the common interface for the fabless semiconductor industry.

At the end of the design, Pat, Gene, and Alberto were featured in a video that Intel distributed worldwide to universities [19]. The video described how microprocessor design was done at Intel and how we had revolutionized CAD by working with Alberto's team to bring in new technology and deliver stunning acceleration in the 486 program. This touting of our academic collaboration was widely recognized in the industry as extremely effective. As part of that video, Pat joked about that "small school in the bay." For a Stanford graduate, a partnership with Berkeley might be a bit unexpected. But with our collaborations in Israel and with Berkeley, we had created an extraordinary sense of teamwork, crossing numerous unwritten barriers to diversity and creativity.

Design Environment of Pentium Processors

The 486 processor was followed by Pentium, Pentium Pro, and various more advanced generations, which integrated numerous architectural extensions and continuously increased complexity. It is interesting to note that the same basic design methodology and design flow remained in effect through all of those generations, while the initial set of tools was replaced by more robust and better-integrated tool sets. As the EDA industry matured, some of the in-house tools were replaced by commercial tools. Starting with the Pentium generation, the two-phase clocking scheme was largely replaced by a single clock and master-slave flip-flops, which were simpler to synthesize and check and are easily supported by commercial tools. RTL remains the primary entry point into the design cycle. No higher-level synthesis has emerged in the design of processors, although higher-level models are used in defining and verifying system architectures.

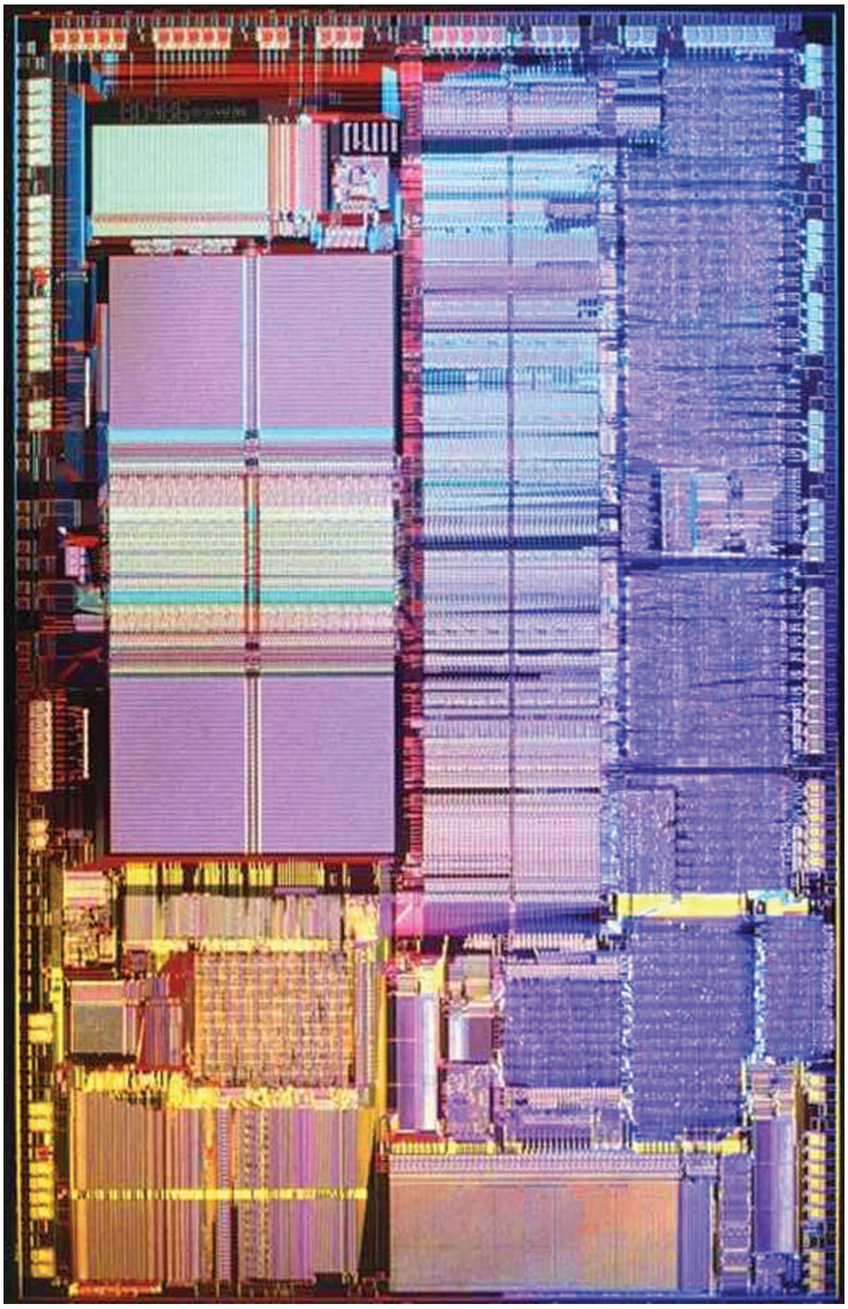
FIGURE 2: The Intel 486 processor. Counter-clockwise from top left: memory interface, 8-kB unified cache, floating-point unit. At the bottom right is the decode logic; microcode ROM is at bottom (mostly handcrafted); going up, there is a split into data path on left (handcrafted) and control on right (all synthesis, with a small handcrafted section in middle of die); crossing control signals are handled by full-chip assembly. (Photo courtesy of Intel Corporation.)

The first Pentium was a superscalar microprocessor design, and the microarchitecture included new features like microcode-based instructions, a 64-bit fast external data bus, and a completely revamped floating-point unit with unprecedented levels of performance (e.g., FMUL operations had 15 times greater throughput than in the 486). At 3.1 million transistors, the Pentium required stronger EDA capabilities. Gadi Singer, designated to be the next Intel liaison to Alberto's group, relocated from Israel to California in the summer of 1990; Avtar Saini, the Pentium design manager, met Gadi at Intel's Santa Clara cafeteria the evening before Gadi drove to Berkeley, and he convinced him to retarget his stay and become the Pentium design automation (DA) manager. That shift did not end up as a total negative for the Intel-Berkeley interaction, as the Pentium DA team continued a very deep and effective interaction with Alberto, Richard Newton, and the rest of the Berkeley team.

Logic and layout synthesis for the control circuits in the Pentium could be performed by the RLS flow and was no longer a problem. The productivity bottleneck for the Pentium design was mainly in the much more complex data path circuits, which were still designed at the schematic level by manual conversion of the RTL model. In particular, the translation of schematics to layout was too slow. The layout designers were using a new symbolic editor, but following well-entrenched practice they continued to lay down wires and gates in a polygon-oriented manner. With a combination of basic training and a set of automation tools to aid symbolic layout, productivity tripled in a matter of weeks. This was an important lesson for the future: the human factor is a major aspect in getting value out of new capabilities.

Manually designed data path circuits had to be checked to verify that their behavior was identical to the RTL model. This area required substantial investment in developing test vectors that would be executed on both the schematic and RTL and would result in high coverage of all functionality branches. Simulating the schematics at the switch level was a major sink of computing resources, and incomplete coverage left holes in verification that were manifested as circuit bugs. Gadi developed a new technology to formally and completely validate the correlation between schematics and RTL. It was a combination of two existing capabilities in a brand-new context. First, the data path circuit schematics were automatically analyzed for their logic expressions and translated into RTL representation [10], [20]. Then, the extracted logic models (in iHDL) were fed into the logic synthesis programs codeveloped with Alberto's group, which could take two logic descriptions, turn them into canonical form, and compare them mathematically. By using this new schematic formal verification (SFV) functionality, all circuits that resided between latch and memory elements could be fully verified against their original RTL descriptions without a single simulation cycle. This removed a whole domain of investment during the Pentium project, reducing test development for schematic verification to zero, reducing the run time to a fraction of the previous dynamic verification, and improving the quality level significantly toward zero schematic mismatches.

Still, the size of the functional verification task for the full-chip RTL model has grown nonlinearly with the size of the processor. The importance of verification was exemplified by the infamous "Pentium FDIV bug," where a rare and minute numerical inaccuracy in a mathematical calculation created a business crisis. The technical challenge, then, was to formally verify that floating-point arithmetic logic as well as all associated microcode was functionally correct. This spawned another phase of Intel's close collaboration with academic researchers (though not with an emphasis on Berkeley) [21], which led to the creation of the Intel Strategic CAD Labs.

Formal verification looked like a promising approach. In principle, this is a static method that examines the design without simulating its behavior over time and does not require test inputs. But the promise did not fully materialize. Functional equivalence checking of RTL to gates has been added to the design flow as a static check. Widely used at Intel, SALT and PEPPER are two internally developed tools for combinational and sequential equivalence checking, respectively. Dynamic verification remains the main way to address the functional verification problem, however. Formal techniques were helpful for property checking by simulation instrumentation (tracking violations of formally specified properties). RTL simulations, carefully designed to "cover" the enormous space of processor states and logical conditions, remain the primary verification vehicle, and they still consume more than 80% of the computing resources for such projects.

Yet another area that became critical in the Pentium era was full-chip timing and modeling of interconnects

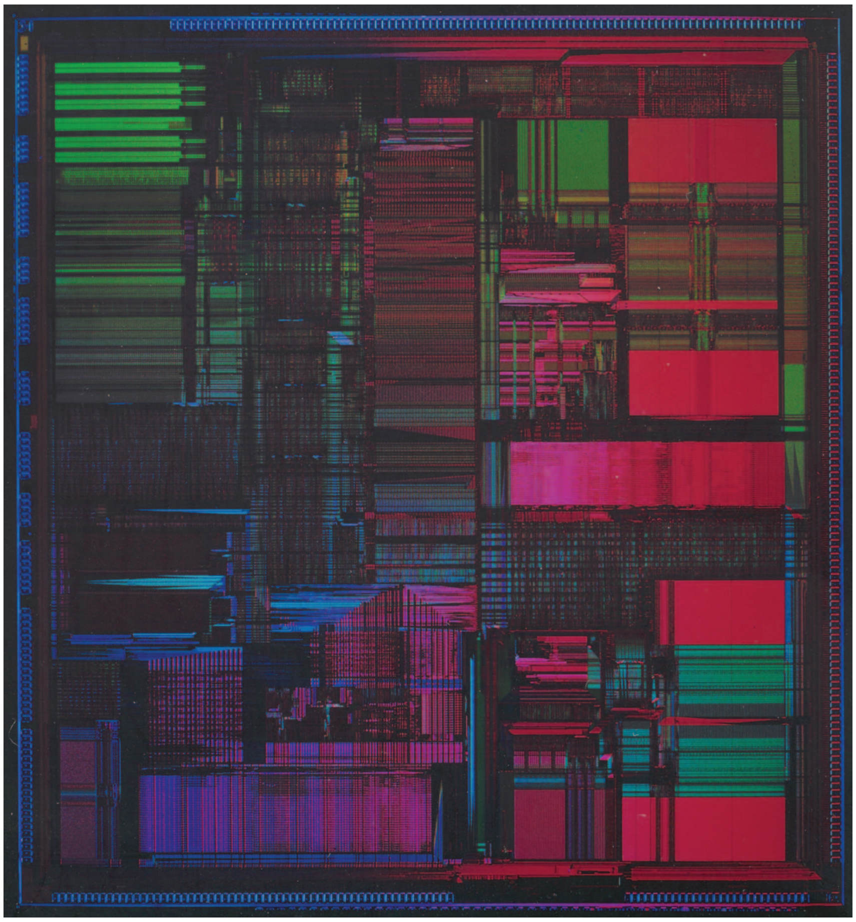

FIGURE 3: The Intel Pentium processor. Counter-clockwise from top left: floating-point unit (handcrafted data path on the left and synthesized controls on the right). The middle of the die consists of the primary data path (handcraft on the right) with a control section on the left (all synthesis) with a channel for chip assembly. The top right consists of an 8-kB data cache. The bus interface logic resides below the data cache. The 8-kB instruction cache occupies the lower right of the die. The instruction fetch and the branch target buffer memory are on the lower left. The microcode ROM and logic were drawn below the floating-point unit. (Photo courtesy of Intel Corporation.)

for static timing analysis. In previous products, smaller circuits were designed using accurate extractions, but large static timing analysis was based on a simplified lumped capacitance extraction model. This was insufficient to support the aggressive timing requirements and the new cross-unit interdependencies that introduced many long-haul signals, however. Distributed *RC* extraction and modeling was introduced for the Pentium along with power grid analysis.

It is important to note that in the 1990s, a very significant productivity gain was achieved by increasing the computational power available to design teams. Interestingly, beginning with the development of the 386, Intel began employing UNIX as its primary engineering development environment, given its more flexible and engineering-oriented nature. In fact, Pat's entry to the design team was because of Intel's zeal for UNIX. The 386 design team was fed up with the DEC 20 and IBM CMS environments and was highly attracted to the flexibility of UNIX. But the only machine then big enough (and available) was the IBM 370/168, later replaced by the 3081. Since Pat was a bit of a UNIX hacker at the time, he set up the entire design team inside his CMS account, which was running the UTS UNIX environment from Amdahl. Thus, he was root on the UTS environment for the entire 386 design team. Everyone was extremely motivated to get to UNIX and thus quickly overlooked Pat's naiveté in logic design as a way to get away from the corporate IT environment. UNIX license plates reading LIVE FREE OR DIE commonly adorned the design team members' offices.

Late in the 486 design process, Intel's whole computing environment was moved to local UNIX workstations. In addition to the interactive performance, the design team was extremely motivated to develop the 486 on 386 machines, the Pentium on 486 machines, and so on. The functional simulations could easily be partitioned into different jobs and run on different workstations. A major invention at Intel was called NetBatch. The idea was to utilize all of the engineering workstations at Intel worldwide as a virtual pool for running verification tasks in parallel, exploiting time-zone differences among sites. This is conceptually similar to grid and cloud computing, which became commercially available several years later.

Discussion and Trends

At each step of the CAD evolution, higher productivity was enabled by increased automation, which leveraged increasing computational capacity, higher abstraction, higher regularity, more usage of hierarchy, and a more disciplined and restrictive methodology.

In the evolution we have described, using RTL instead of schematics was an example of higher abstraction. Using a cell library was an example of higher regularity. Hierarchical decomposition ("divide and conquer") was achieved when complex problems were divided into independent pieces (e.g., the separation of logic verification from timing verification and the separation of logic synthesis from library mapping). This decomposition led to increasing specialization of engineering expertise: for example, due to RTL and synthesis, logic designers have become programmers.

Examples of a restrictive methodology are numerous: the synchronous design paradigm, the specific iHDL language design for synthesis, the cell library, and static CMOS all involve some self-imposed restrictions as part of the engineering discipline. Disciplined restrictions are essential in every methodology. The introduction of new methods and tools did not proceed smoothly but rather encountered skepticism and resistance from designers who did not want to give away their work habits, their control of details, and their creative "rights." They did not want to accept the standards and restrictions of new methodologies (which were chosen in order to minimize verification and allow automation). This kind of conservatism goes together with risk avoidance, as people stick to familiar methods and tools, trying to minimize risks from large-scale engineering programs.

It is also interesting that while manufacturing technology scaling proceeded predictably via coordinated efforts, with Moore's law and Dennard's theory as a top-down road map and a strategic guideline, the evolution of CAD and design methodology occurred from the bottom up and through numerous controversies.

Finally, many of the breakthroughs described in this article were only accomplished by means of significant cross-discipline cooperation. The design teams took significant risks in embracing new methods that had yet to be proven. Design tools were being invented simultaneously with the design team's requirements.

Collaboration among Intel, Alberto Sangiovanni-Vincentelli, and the University of California, Berkeley, continues to this day in a broad range of areas of computer architecture, particularly platform-based design. It is very likely that in order to achieve the next step function in design productivity, people in the electronic design community will have to take such radical codevelopment risks again in large-scale engineering programs where failure is not an alternative. With such risky endeavors, the "era of heroes" may be upon us once more.

Acknowledgments

The authors wish to thank their many collaborators on the design teams at both Intel and Berkeley. Such friends and memories stand as some of the finest of our collective careers. Further, such an overview is prone to errors of memory. While the authors

have attempted to be as accurate as possible, we are certain that errors of recollection exist and there are significant contributions that should be recognized and chronicled to give a fuller history of CAD and microprocessor development.

References

[1] A. S. Grove, *Only the Paranoid Survive: How to Exploit the Crisis Points That Challenge Every Company*. New York: Doubleday, 1996.

[2] G. Moore, "Cramming more components onto integrated circuits," *Proc. IEEE*, vol. 86, no. 1, Jan. 1988, pp. 82–85.

[3] T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE 3BI User's Guide*. Berkeley, CA: Univ. of California Press, Apr. 1987.

[4] C. R. Wilcox, M. L. Dagcforde, and G. A. Jirak, "Mainsail implementation overview," Stanford Comput. Syst. Lab., Rep. No. CSL TR-167, Leland Stanford Junior Univ., Palo Alto, CA, Mar. 1980.

[5] C. Mead and L. Conway. (1980). *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley [Online]. Available: http://ai.eecs.umich.edu/people/conway/VLSI/VLSIText/VLSIText.html

[6] K. Tham, R. Willoner, and D. Wimp, "Functional design verification by multi-level simulation," in *Proc. 21st Design Automation Conf.*, 1984, pp. 473–478.

[7] R. E. Bryant, "MOSSIM: A switch-level simulator for MOS LSI," in *Proc. 18th Design Automation Conf.*, 1981, pp. 786–790.

[8] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis (The Kluwer International Series in Engineering and Computer Science*, vol. 2). Boston, MA: Kluwer, 1984.

[9] G. D. Hachtel, A. L. Sangiovanni-Vincentelli, and A. R. Newton, "Some results in optimal PLA folding (invited paper)," in *Proc. IEEE Int. Conf. Circuits and Computers (ICCC '80)*. New York, NY: IEEE, 1980, vol. 2, pp. 1023–1027.

[10] A. Kolodny, R. Friedman, and T. Ben-Tzur, "Rule-based static debugger and simulation compiler for VLSI schematics," in *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD)*, Santa Clara, CA, Nov. 1985, pp. 150–152.

[11] J. Reed, A. L. Sangiovanni-Vincentelli, and M. Santomauro, "A new symbolic channel router: YACR2," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 4, no. 3, pp. 208–219, 1985.

[12] C. Sechen and A. L. Sangiovanni-Vincentelli. "TimberWolf 3.2: A new standard cell placement and global routing package," in *Proc. 23rd Design Automation Conf.*, 1986, pp. 432–439.

[13] T. J. Wagner, "Hierarchical layout verification," in *Proc. 21st Design Automation Conf.,* 1985, pp. 484–489.

[14] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. CAD-6, no. 6, pp. 1062–1081, Nov. 1987.

[15] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proc. IEEE Int. Conf. Computer-Aided Design ICCAD-87*, Nov. 1987, pp. 116–119.

[16] D. Braun, J. Burns, S. Devadas, K. H. Ma, K. Mayaram, F. Romeo, and A. L. Sangiovanni-Vincentelli, "Chameleon: A new multilayer channel router," in *Proc. 23rd Design Automation Conf.*, 1986, pp. 495–502.

[17] M. Rose, M. Wiesel, D. Kirkpatrick, and N. Nettleton, "Dense, performance directed, auto place and route," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1988, pp. 11.1.1–11.1.4.

[18] M. Rose, N. Papakonstantinou, G. Wellington, D. Kirkpatrick, and M. Wiesel, "Synthesis for high performance random layout," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1990, pp. 885–889.

[19] G. Hill, "Design and development of the Intel 80386 microprocessor," Video recording, Univ. Video Commun., Stanford, CA, 1988.

[20] D. Fischer, Y. Levhari, and G. Singer, "NETHDL: Abstraction of schematics to high-level HDL," in *Proc. Conf. European Design Automation*, 1990, pp. 90–96.

[21] Y. Chen, E. M. Clarke, P. Ho, Y. V. Hoskote, T. Kam, M. Khaira, J. W. O'Leary, and X. Zhao, "Verification of all circuits in a floating-point unit using word-level model checking," in *Proc. Formal Methods in Computer-Aided Design*, 1996, pp. 19–33.

About the Authors

Patrick Gelsinger has been president and COO for EMC Corporation's infrastructure products since 2009. In his 30 years with Intel, he has held numerous positions, including senior vice president and general manager of the Digital Enterprise Group, first-ever Intel CTO, CTO for Intel Architecture Group, general manager of desktop products, design manager for the Pentium Pro and 80486 processors, architect of the 80486 processor, and CAD logic methodology manager and designer for the 80386 processor. He has a master's degree in E.E.C.S. from Stanford, a bachelor's degree in E.E.C.S. from Santa Clara, and an honorary doctorate from William Jessup University. He has received a variety of industry awards, published several books and many papers, and is an IEEE Fellow. He is married with four adult children.

Desmond Kirkpatrick is a principal engineer responsible for Intel's research road map in design efficiency. From 1991 to 1999, he was a member of the Pentium Pro and Pentium 4 microprocessor design teams. In 1999, he became Intel's first technical liaison to the Gigascale Silicon Research Center at Berkeley. In 1986, he joined Intel, where he contributed to hierarchical, full-chip timing analysis, floor planning, layout synthesis, and extraction, earning two Intel Achievement Awards. He received an S.B. degree in electrical engineering from MIT in 1986 and a Ph.D. degree in electrical engineering and computer sciences from Berkeley in 1997.

Avinoam Kolodny is an associate professor of electrical engineering at Technion–Israel Institute of Technology. He joined Intel after completing his doctorate in microelectronics at the Technion in 1980. During 20 years with the company, he was engaged in diverse areas, including nonvolatile memory device physics, EDA, and organizational development. He served as Intel's corporate CAD system architect in California during the codevelopment of the RLS system and the 486 processor, and was manager of Intel's performance verification CAD group in Israel. He has been a member of the faculty of Electrical Engineering at the Technion since 2000. His current research is focused primarily on interconnect issues in VLSI systems, covering all levels from physical design of wires to network-on-chip and multicore systems.

Gadi Singer is vice president of the Intel Architecture Group and general manager of Intel's SoC Enabling Group. He joined Intel in 1983; among other accomplishments, he was appointed vice president and CTO of Intel Communications Group in 1999 and 2004, respectively. From 2005 through 2007, Singer served as general manager of the Ultra Mobility Group. Among his prior roles, Singer was general manager of Intel's Design Technology Division, cogeneral manager of the IA-64 Processor Division, and general manager of the Enterprise Processors Division. He has received three Intel Achievement Awards for his technical contributions. In 1983, Singer received his bachelor's degree in electrical engineering from Technion–Israel Institute of Technology, where he also pursued graduate studies from 1986 to 1988. SSC