# Coping with the Complexity of Microprocessor Design at Intel – A CAD History

Patrick Gelsinger, Desmond Kirkpatrick, Avinoam Kolodny and Gadi Singer

***Abstract* — Necessity has driven the evolution of microprocessor design practices and CAD tools at Intel Corporation, as the transistor count has grown by a factor of about 4X each processor generation. In order to cope with the complexity of design tasks, Intel's engineers were early adopters and adapters of innovative CAD research from universities. A unique partnership with Alberto's group in U.C. Berkeley during the 1980's has created one of the first industrial-strength synthesis-based design flows, which became the prevalent paradigm for the whole electronic industry. This paradigm enabled the semiconductor foundry business model, facilitated the proliferation of fab-less semiconductor companies all over the world, while enabling Intel designers to keep pace with Moore's Law.**

## I. Introduction

During the 1980's, Intel Corp. transformed itself from a semiconductor company producing memory chips into a computer company [1]. Intel's transformation was actually a part of a revolution in the whole electronics industry: in the beginning of the decade, microprocessors were considered as toys; 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, which followed Moore's law. Interestingly, in his original 1965 paper [2], Gordon Moore expressed a concern that the growth rate he predicted may not be sustainable, because the requirement to define and design products at such a rapidly-growing complexity may not keep up with his predicted growth rate. However, the highly competitive business environment drove to fully exploit technology scaling. The number of available transistors doubled with every generation of process technology, which occurred roughly every two years. As shown in Table I, major architecture changes in microprocessors were occurring with a 4X increase of transistor count, approximately every second process generation. Intel's microprocessor design teams had to come up with ways to keep pace with the size and scope of every new project.

Table I: Intel processors 1971-1993

| Processor | Intro Date | Process | Transistors | Freq |
|---|---|---|---|---|
| **4004** | 1971 | 10 um | 2,300 | 108KHz |
| **8080** | 1974 | 6 um | 6,000 | 2 MHz |
| **8086** | 1978 | 3 um | 29,000 | 10 MHz |
| **80286** | 1982 | 1.5um | 134,000 | 12 MHz |
| **80386** | 1985 | 1.5 um | 275,000 | 16 MHz |
| **Intel486 DX** | 1989 | 1 um | 1.2 M | 33 MHz |
| **Pentium** | 1993 | 0.8 um | 3.1 M | 60 MHz |

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 of raising 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 evolving, always addressing the most labor-intensive task at hand. Naturally, the evolution of tools occurred bottom-up, from layout tools to circuit, logic, and architecture. Typically, at each abstraction level the verification problem was most painful, hence it was addressed first. The synthesis problem at that level was addressed much later.

This paper is about the co-evolution story of design methodologies, practices and CAD tools in Intel's design environment, as it had to cope with growing complexity since the turbulent 80's and until recent years. It is interesting to note that at the beginning of this process the engineering culture was advocating a *tall, thin designer*. Nowadays, VLSI engineers are highly specialized in different areas of the design discipline, where specialized tools are used in each area. This is similar to the restructuring of the whole computer industry from *vertical* to *horizontal*.

In the 80's, 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 gave little hope from commercial tool offerings at the time. Thus, 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 the time, none was as significant as that of Intel with U.C. Berkeley. In particular, Alberto and his collaborative research team consisting of Prof. Robert

Brayton, Prof. 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 with fundamental breakthroughs in multiple elements of microprocessor logic, synthesis and layout. Many of these early successes resulted in enormous benefit to Intel and eventually made their way into the EDA industry as key enablers of many EDA tools and today's fab-less /ASIC/SoC semiconductor industry.

## II. Design environment for the early x86 processors

### A. Inherited tools from memory chips

Intel's initial design environment was formed to serve the needs of memory chips. During the 70's, 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 on solely paper or Mylar, hence polygon-based layout representations 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, which originated from Don Pederson's group at U.C. Berkeley, and later on refined by Newton, Alberto and students (Intel's version was known as ISPEC). It was possible to simulate and check logic behavior and timing waveforms for small circuits, up to a few hundred transistors.

As Intel started doing logic products, including the first microprocessors (the Intel 4004, 8008, and 8080), design engineers inherited all of those tools and methods which were initially conceived for memory chip design. Some engineers preferred to perform logic design using gate-level schematics, but this encountered some push-back from the layout designers who were familiar with transistor representations, which directly matched the layout. Translation of logic gate symbols into transistor structures was not a trivial task, because the early microprocessors and numeric co-processors (8087, 80387) were designed in NMOS technology. Circuit operation relied on device strength ratios, so each gate symbol had to be accompanied with specific transistor sizes. In addition, the prevailing design style supported many complex gate pull-down devices, pass transistors for clocking structures, dynamic circuits and numerous other clever and often "tricky" structures which could not be cleanly represented by a simple logic gate abstraction. Consequently, even logic design was actually performed by engineers at the transistor level (a.k.a. *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 they inherently carried with them.

### B. 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 a failure: A gate-level logic simulator called LOCIS was developed at Intel in the mid 1970's, and the 8086 design engineers converted their transistor-level schematics into an equivalent logic model using LOCIS gate models. However, the generic gate models of the simulator did not match the tricky MOS logic structures of the 8086 schematics, and its gate-delay models burdened the users with too many irrelevant timing-related messages and glitch warnings.

After this experience, engineers turned to build functional models with general-purpose programming languages. One of the first Register-Transfer Level (RTL) models at Intel was developed for the 8087 numeric co-processor in 1978. It was a FORTRAN program which 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 that derived from Stanford's AI 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 non-overlapping 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 which 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. However, in that era, 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. Separation of concerns like this continues to be a powerful mechanism in design automation.

Taking advantage of MainSail's support of dynamical 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 to 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. 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 and used to program the PLA arrays into the layout. Much of the early automation in PLA synthesis at Intel was enabled by Alberto's U.C. 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.

### *C. The issue of performance verification*

The RTL-based functional design methodology has separated the issue of timing from the issue of functional correctness, assuming that synchronous methodology was enforced, and that the clock is slow enough for all logic paths to settle to a steady logic state within each clock phase. However, during this time critical paths were only modestly considered during the design phase largely due to lack of tools and engineer's knowledge of the design and the 'likely' critical areas. The large majority of critical paths were not fixed until they were discovered on silicon. The clock could be slowed down until no critical path failure existed. Then the clocks frequency was sped up, but specific clock pulses were extended to help isolate the failing circuit. For example, the 49$^{th}$ 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 by a special *clock stretcher* debugging equipment. However, the 286 design had many second sources and very quickly those manufacturers were finding clever ways to speed up their designs to rival Intel's. This led to a minor crisis within Intel as the industry was quickly putting pressure on Intel in the very architecture and design it created, and the tools to dig into this problem were 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, which could discover circuit-levels bugs and resolve device sizing issues. It could also extract the logic functionality of transistor-level circuit structures and represent them by logical expressions. However, the new capabilities were applied in the next generation microprocessor, the 386, which was no longer in NMOS but rather in CMOS.

## III. 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. However, most of the remainder of the design went through radical changes with the move to 32-bit datapath 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 for acceleration of the design process, minimization of manual translation errors and handling of the rapidly increasing design complexity. With these goals in mind, the relationship with U.C. Berkeley and ASV was quickly center to our efforts. Albert Yu (manager of the Microprocessor Division) and Pat Gelsinger (leading new design methods in Corporate CAD at the time) visited Berkeley to explore some of Alberto's research work and affinity toward our problems as well as the ability to partner on these challenges.

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

other research groups in U.C. Berkeley. The design manager of the project, Gene Hill, put Alberto on a consulting contract to facilitate the above topics as well as reviewing the overall 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 2000 decade, NMOS power was increasing at a near exponential rate. CMOS brought with it a reasonable P device and a strong bias towards 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 CMOS, there was much less benefit to gain 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 individually-sized and customized gate structures which were common in the days of NMOS.

Working with a cell library, we could employ U.C. 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 more optimal layout results. Post global placement by TimberWolf, specific cell placement occurred in standardized rows of standard cells and routing channels with 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.

With layout of standard cell blocks automatically generated, the layout and circuit designers could myopically focus on the highly optimized blocks like the datapath and I/O ring where their creativity could yield much greater impact. Further, these few large blocks greatly simplified the overall global chip floor planning effort allowing a much more rapid final chip assembly with far fewer errors. Verification of final connectivity was performed by an in-house program called CVS written by Todd Wagner [13]. 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.

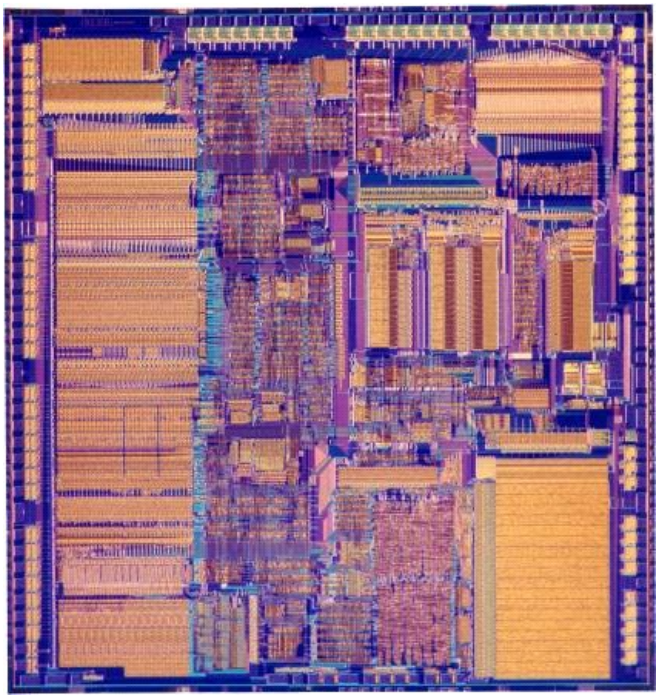


Figure 1: Intel 80386 Processor – Taking a clockwise path around the chip: The upper right was bus interface and instruction decode, lower right was test and control logic and the large microcode ROM, the lower left was the data path for primary instruction executive. Moving up the data stack on the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

## IV. THE 486 DESIGN ENVIRONMENT

### *A. The challenge of logic design effort in the 486*

While the 386 design heavily leveraged the logic design of the 286, the 486 was a more radical departure with 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 8K byte cache which was a write through cache used for both code and data. Given that substantially less of the design was leveraged from prior designs and with the 4X 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 of the ability to afford them, find them, train them and then effectively manage a team that would have needed to be much greater than 100 people that eventually made up the 486 design team.

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

- A fully automated translation from RTL to layout (we called it RLS: RTL to Layout Synthesis)
- No manual schematic design (direct synthesis of gate-level netlists from RTL, without graphical schematics of the circuits)

- Multi-level logic synthesis for the control functions
- Automated gate sizing and optimization
- Inclusion of parasitic elements estimation
- Full chip layout and floor planning tools

For executing this visionary design flow, we needed to put together a CAD system which did not exist yet. We traveled one more time to our now good friend Alberto at U.C. Berkeley to extend our previous collaboration with new tool development. A liaison person 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 U.C. Berkeley. The original research in CAD there was being combined into the "Berkeley Synthesis Project" with focus on merging logic synthesis and layout generation efforts. After collaboration with Alberto at IBM in 1980-1982 and a Berkeley sabbatical in 1985, Dr. Robert Brayton joined the U.C. Berkeley faculty full-time in 1987 and the three main CAD professors, Alberto, Brayton, and Newton joined forces to build what became a highly prolific period in CAD. Alberto coined this era as the "age of the heroes", a "vibrant era of creativity and expansion" in his tour de force DAC 2003 keynote speech. In hindsight, Alberto and his colleagues fostered strong industrial collaboration by their decision to make the results of U.C. Berkeley research (including software systems) freely available to everyone. Through this arrangement, the close technical collaboration between Intel and the U.C. Berkeley CAD group was able to benefit academia and industry, which in turn fueled even more research advances.

As the 486 project was starting in 1986, Gene Hill (Director of microprocessor development) was deliberating whether to take the full risk, or work on a conservative plan in parallel. Gary 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 to Corporate CAD department. There was agreement by Gene with Albert Yu and Mike Aymar (who headed Corporate CAD) to form a central methodology development group under Rafi Nave 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 US and Israel and with U.C. Berkeley and Alberto. He did not expect this at the time, but his next assignment would be managing the 486 design, so he quickly became the customer for the very tool chain he was driving.

### *B. 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 or general 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. A language definition specifically with the formalism required for synthesis with clear semantics for items like busses, native algebraic and Boolean logic functions and the basic control flow mechanisms that a logic design required. The iHDL language defined by Tzvi Ben Tzur, Randy Steck, Gadi Singer and Pat Gelsinger met the bill. In a series of summits between Israel, Oregon and Santa Clara in 1985 and 1986 we converged on a language definition while 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. U.C. Berkeley's adoption of standard intermediate format for logic representation was a key enabler for Intel (and others) to develop higher-level description languages. Amazingly, Intel didn't replace iHDL until 2005 with Verilog simply because of its expressive completeness and effectiveness for synthesis, i.e. a 20 year life to the language.

### *C. Intel's first standard Cell Library*

The vision of automatic conversion of RTL to layout hinged also upon 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 gain productivity due to the large-scale reuse and modularity. Given the long history of individual transistor optimization at Intel, getting agreement on standard cells was no small assignment.

Jim Nadir in Corp 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 political 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.

### *D. Intel's adaptation of logic synthesis from. Berkeley*

We decided that our RLS system would be based on a tool called MIS (Multi-level Logic Interactive Synthesis System) [14], which was actually an experimental workbench which was being developed by the graduate students at Berkeley for executing various restructuring operations on combinational logic blocks. Gary, our liaison person, was regularly sending software releases of MIS from Berkeley, CA 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, which could be handed over to the layout synthesis tools. The library mapping step was developed by U.C. Berkeley 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 amount 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 as well as during place-and-route. As design debate raged then (as it does 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 tool called CLCD [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 the synthesis, some would be globally determined by external requirements and some were highly optimized design signals such as critical datapath signals that were highly optimized by 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 Israeli’s 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 together, between Oregon, California and Israel. 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 email. 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!"

### *E. Physical design automation in RLS*

With the advent of multi-layer metal process technologies, layout synthesis became competitive with manual layout artwork. The complexity of creating dense designs now made automation more suitable and acceptable to engineers. At the time, Intel still used manual effort to generate more regular structures such as memories and datapath, but control logic was synthesized both at the logic and layout levels. Place and route algorithms came from Alberto’s students at U.C. 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] multi-layer 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 produced a series of capabilities led by Manfred Wiesel. The DAPR [17][18] standard-cell tool 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 first time on the 486, we had developed a full-chip floorplanning and assembly tool called ChPPR [19] which used a ½ 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] which was orders of magnitude faster by eliminating layout extraction. The ChPPR hierarchical tool was actively used on mainstream microprocessors at Intel until about 2005, almost twenty years.

### *F. 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, while the highly regular data path was done manually for achieving 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 (providing a convenient input mechanism to capture 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 all of the ASIC industry, and the common interface for the fab-less semiconductor industry.

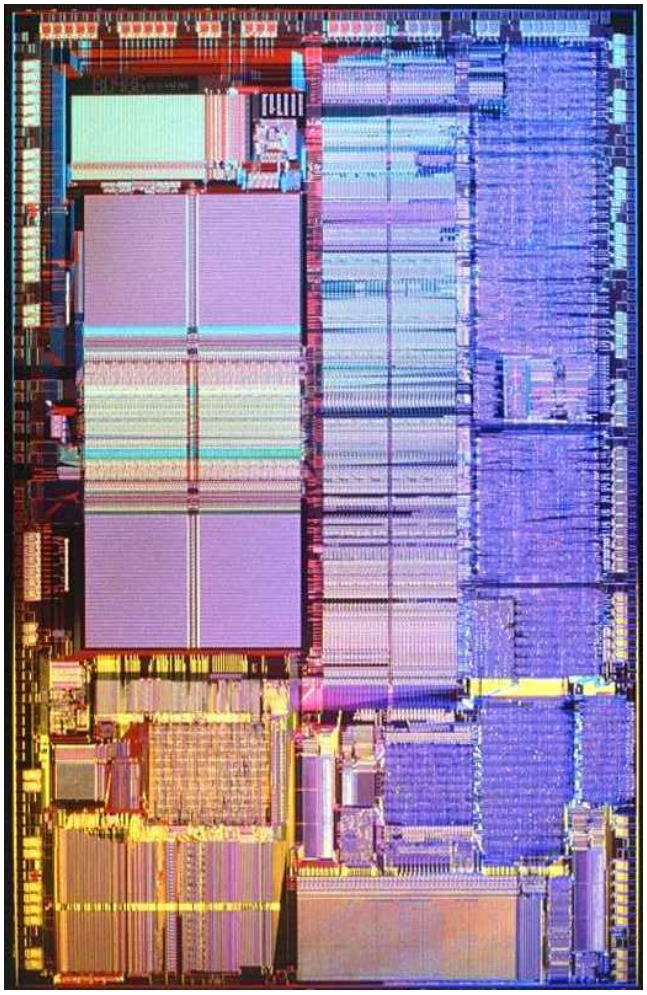


Figure 2: Intel486 Processor -- counter-clockwise from top-left: memory interface, 8k unified cache, floating point unit. At the bottom right is the decode logic, microcode ROM at bottom all mostly hand-craft, then going up split into data-path on left (hand-craft) and control on right (all synthesis with a small hand-craft section in middle of die), crossing control signals handled by full-chip assembly. Photo courtesy of Intel Corporation.

At the end of the design, Pat, Gene and Alberto were featured in a video that Intel distributed worldwide to universities [20]. 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 delivering stunning acceleration in the 486 program. Our commercial to 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". Being a Stanford graduate, a partnership with U.C. Berkeley might be a bit unexpected. However, with our US to Israel, U.C. Berkeley and commercial to university collaborations, we had created an extraordinary sense of teamwork crossing numerous unwritten barriers to diversity and creativity.

## V. Design environment of Pentium processors

The 486 processor was followed by Pentium, Pentium Pro and 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 has remained in effect through all of those generations, while the initial set of tools were replaced by more robust and better integrated tool sets. As the EDA industry has matured, some of the in-house tools were replaced by commercial tools. Starting at 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.

The first Pentium was a superscalar microprocessor design and the micro-architecture included new features like microcode-based instructions, 64-bit fast external data-bus and a completely revamped Floating Point Unit with unprecedented levels of performance (e.g., the FMUL was about 15 times higher throughput than in the 486). At 3.1 million transistors, Pentium required stronger EDA capabilities. Avtar Saini, the Pentium design manager, met Gadi Singer who relocated from Israel to California in the summer of 1990, designated to be the next Intel liaison person in Alberto's group. Avtar talked to Gadi at Intel's Santa Clara cafeteria on the evening before he drove to Berkeley, and convinced him to retarget his stay and become the Pentium DA manager. That shift did not end up a total negative for the Intel-Berkeley interaction as the Pentium DA team continued a very deep and effective interaction with Alberto, Newton, and the rest of U.C. 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 datapath 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 due to well entrenched practices 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, that the human factor is a major aspect in getting value out of new capabilities.

Manually-designed datapath 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 cover all functionality branches with high coverage. Simulating the schematics at 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 datapath circuit schematics were automatically analyzed for their logic expressions and translated into RTL representation [10], [21]. Then, the extracted logic models (in iHDL) were fed into the logic synthesis programs co-developed 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 reside between latch/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 duration, reducing test development for Schematic Verification to zero, reducing the run time to a fraction of the previous dynamic verification, and increasing the quality level towards zero schematic mismatches.

Still, functional verification of the full chip RTL model has grown non-linearly 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 some mathematical calculation has created a business crisis. The technical challenge then was to formally verify floating-point arithmetic logic as well as all associated micro-code to be functionally correct. This spawned another phase of Intel's close collaboration with academic researchers [22] (though not with a U.C. Berkeley emphasis) 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 which examines the design without simulating its behavior over time and does not require test inputs. However, 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. However, dynamic verification remains the main way to address the functional verification problem. Formal techniques were helpful for property checking by simulation instrumentation (tracking violation 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.

Yet another area which became critical in the Pentium era was full chip timing and modeling of interconnects 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. However, this was insufficient to support the aggressive timing requirements and the new cross-unit interdependencies that introduced many long-haul signals. Distributed RC extraction and modeling was introduced for the Pentium, as well as the power grid analysis.

It is important to note that since the 1990s a very significant productivity gain was achieved by increasing the computational power available to design teams. It is important to consider the computing environment at Intel design centers. Interestingly, beginning with the 386 development Intel began employing UNIX as its primary engineering development environment given its more flexible and engineering oriented environment. In fact, Pat's entry to the design team was because of their zeal for UNIX. The 386 design team was fed up with the DEC 20 and the IBM CMS environment, and was highly attracted to the flexibility of the UNIX environment. However, the only machine big enough at the time (and available) was the IBM370-168, later replaced with 3081. Given Pat was a bit of a UNIX hacker at the time, he set up the entire design team inside of 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. "Live Free or Die" UNIX license plates commonly adorned design member's offices.

Late in the 486 design and entirely for the Pentium generations 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, Pentium on 486 machines and so on. The functional simulations could be easily partitioned into different jobs for running on different workstations. A major invention at Intel was called NetBatch. The idea was to utilize all of the engineering workstations at Intel world-wide 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 have become commercially available several years later.

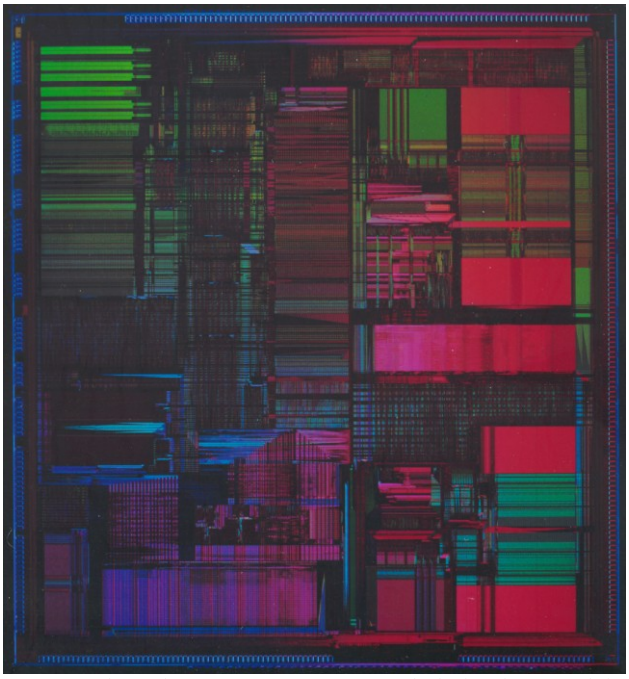


Figure 3: Intel Pentium Processor – counter clockwise from top-left: floating point unit (hand crafted datapath on the left and synthesized controls on the right). The middle of the die consists of the primary datapath (handcraft on the right) with a control section on the left (all synthesis) with a channel for chip assembly. The top right consists of an 8K data cache. The bus interface logic resides below the data cache. The 8K 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.

## VI. Discussion and Trends

At each step of the CAD evolution, higher productivity was enabled by increased automation, which leveraged increasing compute 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. separation of logic verification from timing verification, separation of logic synthesis from library mapping). This decomposition led to specialization in the expertise of engineers: 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, static CMOS, all involve some self-imposed restrictions as part of the engineering discipline. Disciplined restrictions are essential in every methodology. However, 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 wild creative "rights". They did not want to accept the standards/restrictions of new methodologies (which were chosen in order to save verification and allow automation). This kind of conservatism goes together with risk-avoidance, as people stick to their 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 roadmap and a strategic guideline, the evolution of CAD and design methodology happened bottom-up and via numerous controversies.

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

Collaboration between Intel, Alberto and U.C. Berkeley continues to this day in a broad range of areas of computer architecture and in particular in the area of 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 once 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 from across Intel, U.C. Berkeley. Such friends and memories stand as some of the finest of our collective careers. Further, such an overview paper 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 better chronicled for a better 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, Doubleday 1996.

[2] G. Moore, "Cramming more components onto integrated circuits," Electronics, Vol. 38, No. 8, April 19, 1965.

[3] T. Quarles, A. R. Newton, D. 0. Pederson, and A. Sangiovanni-Vincentelli, "SPICE 3BI User's Guide", Univ. of Calif., Berkeley, Apr. 1987.

[4] C.R. Wilcox, M.L. Dagcforde, G. A. Jirak, "Mainsail Implementation Overview," Stanford Computer Systems Laboratory Report No CSL TR-167, March 1980.

[5] C. Mead, L. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980. (Out-of-print: pre-print drafts available: http://ai.eecs.umich.edu/people/conway/VLSI/VLSIText/VLSIText.html)

[6] K. Tham, R. Willoner, D. Wimp, "Functional Design Verification by Multi-Level Simulation," Proceedings 21st Design Automation Conference, 1984, pp 473-478.

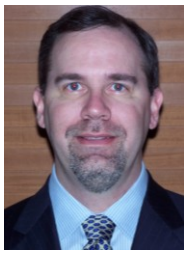
[7] R. E. Bryant, "MOSSIM: A switch-level simulator for MOS LSI," Proceedings of the 18th Design Automation Conference, 1981, pp 786-790.

[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 Academic Publishers, 1984.

[9] G. D. Hachtel, A. L. Sangiovanni-Vincentelli, and A. R. Newton, "Some results in optimal PLA folding (Invited Paper)," in Proc. IEEE Intl. Conf. on Circuits and Computers (ICCC '80), Vol. 2, New York, NY: IEEE, 1980, pp 1023-1027.

[10] A. Kolodny, R. Friedman, and T. Ben-Tzur, "Rule-based Static Debugger and Simulation Compiler for VLSI Schematics," Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD), Santa Clara, CA, Nov. 1985.
[11] J. Reed, A. L. Sangiovanni-Vincentelli, and M. Santomauro: "A New Symbolic Channel Router: YACR2," IEEE Transactions on CAD of Integrated Circuits and Systems, 1985, pp 208-219.
[12] C. Sechen, A. L. Sangiovanni-Vincentelli. "TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package," Proceedings 23rd Design Automation Conference. 1986, pp. 432-439.
[13] T. J. Wagner, "Hierarchical Layout Verification," Proceedings 21st Design Automation Conference, 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 of Integrated Circuits and Systems, vol. CAD-6, no. 6, Nov. 1987, pp 1062-1081.
[15] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," Proceedings of the IEEE International Conference on Computer-Aided Design ICCAD-87, November 1987, pp 116-119.
[16] D. Braun, J. Burns, S. Devadas, K. H. Ma, K. Mayaram, F. Romeo, A. L. Sangiovanni-Vincentelli, "Chameleon: A New Multi-Layer Channel Router," Proceedings 23rd Design Automation Conference, 1986, pp 495-502.
[17] M. Rose, M. Wiesel, D. Kirkpatrick, and N. Nettleton, "Dense, Performance Directed, Auto Place and Route," IEEE Custom Integrated Circuits Conference, 1988, pp 11.1.1-4.
[18] M. Rose, N. Papakonstantinou, G. Wellington, D. Kirkpatrick, and M. Wiesel, "Synthesis for High Performance Random Layout," Proceedings IEEE International Symposium on Circuits and Systems, 1990, pp 885-889.
[19] S. Meier, N. Nettleton, D. Kirkpatrick, and D. Braun. "ChPPR - Chip Planning, Placement and Routing," IEEE Custom Integrated Circuits Conference, Section 2, 1990.
[20] G. Hill, "Design and Development of the Intel 80386 Microprocessor," (video-recording) University Video Communications, Stanford, CA, 1988.
[21] D. Fischer, Y. Levhari, and G. Singer, "NETHDL: abstraction of schematics to high-level HDL," Proceedings of the Conference on European Design Automation, 1990, pp 90-96.
[22] Y. Chen, E. M. Clarke, P. Ho, Y. V. Hoskote, T. Kam, M. Khaira, J. W. O'Leary, X. Zhao, "Verification of All Circuits in a Floating-Point Unit Using Word-Level Model Checking," Proceedings of Formal Methods in Computer-Aided Design, 1996, pp 19-33.

**Pat Gelsinger** is President and COO for EMC Corporation's Infrastructure Products since 2009. Pat had numerous roles for Intel in his near 30 years with the company including, Sr. VP and GM of Digital Enterprise Group, First ever CTO for Intel, CTO for Intel Architecture Group, GM of Desktop products, Design manager for the Pentium Pro and the 80486, Architect of the 80486, CAD Logic Methodology manager and designer on the 80386. Pat has a Masters in EECS from Stanford, a BS in EECS from Santa Clara and an honorary doctorate from William Jessup University. He has received a variety of industry recognition 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 roadmap in design efficiency. From 1991–1999, he was a member of the Pentium Pro and Pentium 4 microprocessor design teams, contributing to full-chip assembly and interconnect performance management as well as to the specification of Intel's 130 nm process technology. In 1999, he became Intel's first Technical Liaison to the Gigascale Silicon Research Center at U.C. 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 the S.B. degree in electrical engineering from Massachusetts Institute of Technology in 1986 and the Ph.D. degree in electrical engineering and computer sciences from U.C. 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 twenty years with the company he was engaged in diverse areas including non-volatile memory device physics, electronic design automation and organizational development. He pioneered static timing analysis of processors as the lead developer of the CLCD tool, served as Intel's corporate CAD system architect in California during the co-development 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 networks on chip and multi-core systems.

**Gadi Singer** is vice president of the Intel Architecture Group and general manager, SoC Enabling Group for Intel Corporation. Singer joined Intel in 1983, holding a variety of senior technical and management positions. He was appointed VP in 1999 and CTO of Intel Communications Group in 2004, among other accomplishments. From 2005 through 2007, Singer served as general manager of the Ultra Mobility Group. Among his prior roles, Singer was GM of Intel's Design Technology Division, co-GM of the IA-64 Processor Division and GM of Enterprise Processors Division. Singer received three Intel Achievement Awards for his technical contributions. Singer received his bachelor's degree in electrical engineering from Technion University, Israel, in 1983 where he also pursued graduate studies from 1986 to 1988.