The Pentium Chronicles: Introduction

Robert P. Colwell

God tells me how the music should sound, but you stand in the way.

—Arturo Toscanini, to a trumpet player

In June 1990, I joined Intel Corporation's new Oregon microprocessor design division as a senior computer architect on a new project, the P6. This division would eventually grow to thousands of people but at the moment it had a population of exactly one—me. I spent my first day buried in forms, picking primary health-care providers mostly on the basis of how much I liked their names. The second day, my boss stuck his head in my office and said, "Your job is to beat the P5 chip by a factor of two on the same process technology. Any questions?" I replied, "Three. What's a P5? Can you tell me more about Intel's process technology plans? And where's the bathroom?"

P5, as it turned out, was the chip the Intel Santa Clara design team was developing, the team that had created the very successful 386 and 486 chips. P5 would become the original Pentium processor when it debuted in 1993, but in June 1990, my time frame, P5 was little more than a letter and number.

The P6 project was to follow the P5 by two years. Extrapolating from Moore's Law, it appeared that P6 might have as many as eight to 10 million transistors and could become a product as soon as late 1994. My job and the job of the team I was to help recruit was simply to figure out what to do with those transistors and then do it.

In summer 1992, two years after joining the project, I was promoted to architecture manager and served as Intel's lead IA-32 architect from 1992 through 2000.

Somewhat to my surprise, the P6 design project turned out to be a watershed event in the history of the computer industry and the Internet; it could keep up with the industry's fastest chips, especially those from reduced-instruction-set computer (RISC) manufacturers, and it had enough flexibility and headroom to serve as the basis for many future proliferation designs.

It also gave Intel a foothold in the maturing workstation market, and it immediately established them in the server space just as the Internet was driving up demand for inexpensive Web servers.

The P6 project would eventually grow to over 400 design and validation engineers and take 4.5 years to production. But that huge investment paid off—P6 became the Pentium Pro microprocessor, was adapted to become the Pentium II and then the Pentium III, and, most recently, has evolved into the Centrino mobile line. From the basic design have come numerous Xeon and Celeron variants.

In short, the P6 has become the most successful general-purpose processor ever created, with hundreds of millions of chips being shipped. This book is my personal account of that project, with occasional excursions into Pentium 4.

P6 PROJECT CONTEXT

To fully appreciate where the P6 came from, you must first consider the industry and technology context.

The microelectronics industry has been blessed for several decades with an amazing benefaction: The silicon chips on which we place our circuits improve drastically about every two years. New process technology

historically provides twice the number of transistors, makes them fundamentally faster, and generally reduces the power requirement as well.

If we were to do no engineering other than to simply convert an existing design to the newly developed silicon process, the resulting chip would be faster and cheaper without our having done very much work. That sword has two edges, though. The good news is that if I start designing a new CPU chip today toward a production goal that is, say, three years away, I know I can count on having a new process technology. So I design to that new process technology's particular set of design rules, and I am pretty confident that this better technology plus my design team's clever innovations will give my new chip a clear advantage over what is available today.

But the main reason to go through the expense and effort of designing a new CPU is that it will be substantially better than what exists. For microprocessors, "better" generally means higher overall performance that will enable more interesting software applications, such as operating systems with improved user interfaces and shoot-'em-up games with ever-more-realistically rendered bad guys.

My new chip has to deliver higher performance than its predecessors or I have accomplished nothing. My competition will also migrate to the new process technology within my three-year horizon, and its existing chip will have become faster. That's the sword's other edge: The target isn't stationary. A new chip has to beat the competition's new design, as well as any process-migration chips from any source, including my own company.

My and my fellow chip architects' job was, therefore, to find ways of organizing the new microprocessor's internal design so that it would clearly be superior to any others. Naturally, the first step was to identify "any others" and thereby establish the focus of our project goals.

In 1990, Intel was still developing Intel 486 chips—33 MHz, 50 MHz, and 66 MHz, eventually reaching 100 MHz by 1992. P5 was the code name of the design project being done in Santa Clara by (mostly) the same team that had produced the 486 and the 386 before that.

Task 1 was, therefore, to scope out the P5, analyze its performance potential, investigate the techniques the Santa Clara team was using, and then come up with something that would be twice as fast.

Betting on CISC

Other Intel chips were not the only competition. Throughout the 1980s, the RISC/CISC debate was boiling.

RISC's general premise was that computer instruction sets such as Digital Equipment Corporation's VAX instruction set had become unnecessarily complicated and counterproductively large and arcane.

In engineering, all other things being equal, simpler is always better, and sometimes much better. All other things are never equal, of course, and commercial designers kept adding to the already large steaming pile of VAX instructions in the hope of continuing to innovate while maintaining backward compatibility with the existing software code base.

RISC researchers promised large performance increases, easier engineering, and many other benefits from their design style. A substantial part of the computer engineering community believed that Complex Instruction Set Computers (CISCs) such as the VAX and Intel's x86s would be pushed aside by sheer force of RISC's technical advantages.

The chip architects' job was to find ways of organizing the new microprocessor's internal design so that it would clearly be superior to any others.

In 1990, it was still not clear how the RISC/CISC battle would end. Some of my engineering friends thought I was either masochistic or irrational. Having just swum ashore from the sinking of the Multiflow ship (Multiflow was a computer systems startup that folded in 1990), I immediately signed on to a "doomed" x86 design project. In their eyes, no matter how clever my design team was, we were inevitably going to be swept aside by superior technology.

But my own analysis of the RISC/ CISC debates was that we could, in fact, import nearly all of RISC's technical advantages to a CISC design. The rest we could overcome with extra engineering, a somewhat larger die size, and the sheer economics of large product shipment volume.

Although larger die sizes are generally not desirable because they typically imply higher production cost and higher power dissipation, in the early 1990s, power dissipation was low enough that fairly easy cooling solutions were adequate. And although production costs were a factor of die size, they were much, much more dependent on volume being shipped, and in that arena, CISCs had an enormous advantage over their RISC challengers.

In joining Intel's new x86 design team, I was betting heavily that my understanding was right. P6 would have to beat Intel's previous chips, AMD's best competitive effort, and at least keep the most promising RISC chips within range.

Proliferation thinking

We quickly realized we were not just "designing a chip" with the P6 project.

Intel's modus operandi is for a flagship team (like the P6) to start with a blank sheet, conceive a new microar-

chitecture, design a chip around it, and produce relatively limited production volumes.

Why is that a good plan, in an industry where large economies prevail? There are several reasons. The first is that the architects must be fairly aggressive in their new design; they will want to spend every transistor they can get, because they know how to translate additional transistors into additional performance, and performance sells. This means that the first instantiation of their concept will fill the die, making it large.

The physics of manufacturing silicon chips is such that a larger die is much less economical than a smaller one, since fewer such chips fit onto a silicon wafer, and also because random manufacturing defects are much more likely to ruin a large chip than a small one. Because of this large-die issue, the first version of a new microarchitecture will be expensive, which automatically limits its sales volume.

32-bit code would be the battleground for the RISC/CISC conflict, and also the future of general software development.

But the second, third, and *n*th proliferations of the original chip are the moneymakers. These follow-on designs convert the design to a new silicon process technology, thereby gaining all the traditional Moore's Law benefits. The chip gets smaller because its transistors and wires are smaller. It gets faster because smaller transistors are faster. Smaller is also cheaper—more silicon die fit on a given silicon wafer, and there will be more good die per wafer, with less die area exposed to potential contamination. Moreover, the team is much smaller than the original design team, and it only takes about a year instead of 3 to 5 years for the flagship process.

Henry Petroski points out that this flagship/proliferation paradigm is not unique to the microprocessor industry: "All innovative designs can be expected to be somewhat uneconomical in the sense that they require a degree of research, development, demonstration, and conservatism that their technology descendants can take for granted."[1]

When it became clear that P6's real importance to Intel was not so much its first instantiation (which Intel eventually marketed as the Pentium Pro), but in its "proliferability," we began to include proliferation thinking in our design decisions. Early in the project, proliferability figured prominently in discussions about the P6's frontside bus, the interconnect structure by which the CPU and its chip set would communicate. Some of the marketing folks pointed out that if the P6 had the same frontside bus as the P5 (Pentium) project, then our new CPU would have ready-made motherboards when silicon arrived. If the P6's chip set was delayed for some reason, we could debug our new CPU on the P5's chip set.

These arguments were absolutely correct on the surface, but they overlooked the bigger picture: Long-term, the P5 bus was woefully inadequate for the much higher system performance levels we believed we would get from the P6's proliferations. We had also begun considering whether a multiprocessor design was feasible, and the P5 bus was very inappropriate for such systems. We could do a lot better with the new packaging and bus driver circuits that were becoming available.

Another design decision that proliferation thinking heavily influenced was the relative performance of 16- and 32-bit code. 16-bit code was legacy code from the DOS era. We knew P6 would have to run all Intel Architecture x86 code to be considered compatible, but we believed that as the years rolled by, 16-bit code would become increasingly irrelevant. (The ascendancy of 32-bit code over 16-bit probably seems perfectly obvious today, and the trend was indeed unmistakable. But as in music, politics, and electronics, timing is everything.) 32-bit code would be the battleground for the RISC/CISC conflict, and also the future of general software development, and we intended to make a good showing there. So we concentrated on designing the P6 core for great 32-bit performance, and with 16-bit performance, it would be "you get what you get."

The gauntlet

That was pretty much the environment of the P6 project. We were designing a product we hoped would be immediately profitable, but we were willing to compromise it to some extent on behalf of future proliferations. P6 would have competition within the company from the P5 chip and outside the company from other x86 vendors and the RISC competitors.

Although some of us were very experienced in computer systems and silicon chip design, a team as large as the one we were envisioning would have to have a large percentage of novice "RCGs" (recent college graduates), and we were still a brand new division, with no x86 track record. Over the next 5 years, Intel would bet several hundred million dollars that we would find answers to these challenges.

We not only found the answers, but we also came up with a microarchitecture that propelled Intel into volume servers, fundamentally changing the server space by making servers cheap enough that every business could afford one. (Such servers were, of course, essential for the explosive growth of the worldwide Web in the mid 1990s, but we hesitate to take credit for the Internet. That goes to Al Gore. Ask him. He'll tell you.) Intel also realized a handsome profit from the three million Pentium Pro microprocessors it sold, so we hit that goal too.

But at the beginning of those 5 years, about all we had were some big ideas and a short time in which to cultivate them.

DEVELOPING BIG IDEAS

The first step in growing an idea is not to forget it when it comes to you. Composers, writers, and engineers routinely work hard at simply remembering their glimpses of brilliance as they arise. Then they try to incorporate their brainchild into the current project and move on to the next challenge. For small ideas, those that affect primarily your own work, any number of techniques will allow those good ideas to flourish.

Not so with big ideas. Big ideas involve a lot of people, time, and money, all of which are necessary but not sufficient conditions for success.

Good engineers would much rather use a known-good method to accomplish some task than reinvent everything.

Engineering projects begin with a perceived need or opportunity, which spawns an idea, some realizable way to fill that need. Even if your boss just tells you to do something, you still "need" to do it. So ideas start with "Wouldn't it be great if we had a bridge spanning San Francisco harbor to Marin County?"; or, "What if we placed towers every so often along busy highways, and used them to relay radiotelephone traffic?" (I know! We could call them "cell phones!); or, "We could put up satellites, time their movement and transmissions, and then use them to determine someone's exact position on the Earth's surface," and so on.

In 1961, President John F. Kennedy committed the United States to placing a man on the moon by the end of the 1960s and returning him safely to Earth.[2] That was the perceived need or opportunity. NASA engineers had to conceive ways to realize that vision. Could they make booster rockets safe enough to carry humans into space? What were the feasible ways of landing a craft on the lunar surface such that it could later take off again? How could that hardware be transported from Earth to the moon? Should it be launched directly as a straight shot, or should the lunar attempt launch from the Earth's orbit?

The process NASA followed was to identify several promising ideas and then attack each one to see if they could find a showstopper flaw in it. They systematically eliminated the plans that would not work and increased their focus on the ones that survived. In the end, they settled on a compound plan that included the orbit around Earth, the trip to the moon, a lunar orbit, and a landing craft with two pieces, one for landing (which would be left behind) and one for takeoff and return to lunar orbit.

At every step of the Apollo program, this overall concept determined the engineering and research. The two-stage lunar lander could be accurately estimated as to weight and size, which set the thrust requirement for the lander's engines. The overall thrust required to get the rocket, its fuel, and the lander into Earth orbit in turn guided the development of the huge Saturn V booster. The various docking and undocking maneuvers implied a need for airtight docking seals and maneuvering thrusters.

Our approach to the P6 project was a lot like NASA's approach to the moon shot. We tried as much as possible to reuse existing solutions, tools, and methods, but we knew at the outset that existing technology, tools, and project management methods would not suffice to get our project where we wanted it to go. So we purposefully arranged the design team to address special technology challenges in circuits, buses, validation, and multiprocessing.

Defining success and failure

Engineers generally recognize the validity of the saying, "Creativity is a poor substitute for knowing what you're doing." (Ignore what Albert Einstein is reputed to have said: "Imagination is more important than knowledge." That might be valid for a scientist, but as an engineer, I know that I can't simply imagine my bridge tolerating a certain load.) Good engineers would much rather use a known-good method to accomplish some task than reinvent everything. In this way, they are free to concentrate their creativity on the parts of the design that really need it, and they reduce overall risk.

On the other hand, if we apply this thinking to overall engineering project management, we are in trouble. Our instinct to exhaustively research project management methods, pick the best one, and implement it will lead us to an infinite loop because new project management methods are being written faster than we can read them. Worse, there's no apparent consensus among these learned treatises, so there's no easy way to synthesize a "best-of" project management methodology. Moreover, methods that work on one project may fail badly on the next because the reward for succeeding on one design project is that you get to do it again, except that the next project will be at least twice as difficult. That's the dark cloud around the Moore's Law silver lining.

Large, successful engineering companies must constantly struggle to balance their corporate learning (as embodied in their codex of Best Known Methods) against the need of each new project to innovate around problems that no one has faced before. In a very important sense, the P6 project was blessed with a team whose members either had never worked on Intel's x86 chips or had never worked at Intel. This helped enormously in getting the right balance of historical answers and new challenges.

Senior wisdom

In most cases, a company will present the "new project" need or opportunity to a few senior engineers who then have the daunting job of thoroughly evaluating project requirements and answering a two-pronged question: What constitutes success for this project and what constitutes failure? They must identify possible avenues for the project to pursue that will lead to the promised land.

The path is not easily recognizable. Nature is biased against success: For every plan that works, thousands fail, many of them initially quite plausible. And the success criteria are not simply the logical converse of failure conditions. For the P6, success criteria included performance above a certain level and failure criteria included power dissipation above some threshold.

Large projects can be outrageously inefficient if not managed properly.

In essence, a few senior people are making choices that will implicitly or explicitly guide the efforts of hundreds (or in NASA's case, tens of thousands) of others over the project's life. It is, therefore, crucial that project leadership be staffed correctly and get this phase right, or it will be extremely hard for the project to recover. Do not begin the project until the right leadership is in place.

Occasionally, you will see articles about computer programmers who are wildly talented at cranking out good code. Such people do exist. We don't really know where they come from, and we don't know how to make more of them, but we know them when we see them. To try to put these superprogrammers into perspective, their output is usually compared to that of their less sensational compatriots—"one superprogrammer can turn out as much code in a day as three of her coworkers could in a week."

As with senior project leadership, this kind of comparison misses the point: You can't substitute higher numbers of less gifted people for the efforts of these chosen few. Quantity cannot replace quality. Guard these folks when you find them, because you cannot replace them, and their intuitions and insights are essential to getting a project onto the right track and keeping it there through production.

FOUR PROJECT PHASES

Small projects involving only a few engineers can succeed on a seat-of-the-pants, just-do-whatever-needs-doing basis. As long as an experienced engineer is in charge—one who can recognize when the team has found a workable product concept and when to drive the project forward—the project can succeed. But large projects suffer from this ad hoc treatment. Large projects can be outrageously inefficient if not managed properly and might even implode if allowed to stall long enough. Large projects require structure and scheduling.

Although we certainly had structure and a schedule, we did not start with the conceptual framework that forms the backbone of this book. Rather, the framework presented is a product of my attempt to impose order and labels on what we actually did, with the benefit of hindsight and the passage of time.

The four major phases I've been able to distill are

1. Concept
2. Refinement
3. Realization
4. Production

In the *concept* phase, senior engineers consider the request or opportunity and try to brainstorm ways to satisfy it. If the need was "a better way to get from downtown San Francisco to Marin County," they would create a set of possible solutions that might include ferries, tunnels, bridges, trained dolphins, blimps, water wings, submarines, inflated inner tubes, human cannonballs, and jet-skis. (Remember, this is the anything-goes brainstorming phase.)

The *refinement* phase weeds out the implausible solutions and prioritizes the rest so that the project's limited engineering effort concentrates on the ideas that are most likely to pan out. Of the initial set of, say, 10 or 20 ideas that exit the concept phase, only two or three are likely to survive the refinement phase. One of these will be designated the plan-of-record and will receive the most attention at the beginning of the realization phase.

Realization is the actual engineering process. The design team takes the best idea that has emerged from the refinement phase (and may even have been the instrument by which refinement occurs) and implements the prototype product.

The last phase of the engineering process—*production*, driving what you've created into solid high volume—is often overlooked by design teams. Design teams must shepherd their creation all the way through volume production, not simply transfer responsibility to some production engineering group at the first sale.

As in any project framework, the four project phases overlap. The project as a whole may be transitioning from concept to refinement over a few weeks or months. Any design engineer in the project might be at some point in this transition, substantially lagging or leading the rest of the project. One part of a design team might be finishing a previous design and thus be unable to join a new effort until most of the concept phase is over.

This four-stage model can be extremely useful as a management tool, as well as a way to coordinate the design team. (I wish we had recognized it as such.) The team should superimpose the four stages on the overall project schedule, so that everybody knows how to best

make their local decisions. Ideas that are worth chasing down when the project is in the concept phase might have to be triaged at later phases, for example.

THE BUSINESS OF EXCELLENCE

I would be remiss if I did not emphasize the role of the P6 team.

In the 1970s, the Pittsburgh Steelers football team won four Super Bowls. It wasn't just that the Steelers had dominating players at so many positions. It wasn't just that they were well trained and executed brilliantly much of the time and at least competently the rest. It wasn't even that the Steelers were underdogs for the first half of the 1970s. It was that the Steelers were determined to win. There was a palpable sense about that team that they would face and subdue any challenge that turned up. They would do whatever it took to succeed, and their definition of success was to be the best in the world at what they did.

The P6 team had those qualities. We relied heavily on Randy Steck's excellent innate managerial instincts and constant drive to improve and a very experienced senior technical leadership who didn't have to be told what to do next. We also had an entrepreneurial slay-the-drag-ons-as-they-appear resilience and a willingness to try new things. In the end, P6 turned out the way it did because an incredibly talented design and architecture team was fervently committed to making it a success.

I am not implying that we necessarily knew more than other project managers, nor am I suggesting that ours was a better approach than any project management book would propose (although I am sure it was better than some). I simply want to relate what we did, why we did it that way, and how it turned out. …

References

1. H. Petroski, *Design Paradigms*, Cambridge Univ. Press, 1994
2. R. Turnill, *The Moonlandings: An Eyewitness Account*, Cambridge Univ. Press, 2003.

***Robert P. Colwell**, the 2005 recipient of the IEEE Computer Society/ACM Eckert-Mauchly Award, was Intel's chief IA32 architect through the Pentium II, III, and 4 microprocessors. He is now an independent consultant. Contact him at bob.colwell@comcast.net.*