Oral History of David (Dave) Ditzel

Interviewed by:
Kevin Krewell

Recorded: July 31, 2015
Mountain View, California

CHM Reference number: X7554.2016

Kevin Krewell: Hi, I'd like to introduce myself. I'm Kevin Krewell. I'm a member of the Semiconductor SIG at the Computer History Museum. Today is July 31st, 2015. We are at the Computer History Museum, and we're about to interview Dave Ditzel, who's probably best known as the founder of Transmeta. But, also an early researcher in RISC processor design at UC Berkeley. He's also worked at ATT Bell Labs, and at Sun Microsystems. Those are probably his most well-known attributes, or his well-known jobs.

Dave Ditzel: And even at Intel. That was a surprise to people.

Krewell: And Intel, but that's probably less well known. Most people were surprised when--

Ditzel: I wasn't allowed to talk about what I was doing there.

Krewell: --I don't know if you still can.

Ditzel: A little bit.

Krewell: So, let's start off with a little background on Dave, and then we'll work into his history, fascinating history actually. But we're going to start off just a little bit about family background. Why don't just give us a little growing up, and where you born and raised, and how you started life.

Ditzel: Generally, I grew up in the Midwest, Missouri and Iowa. My father was a chemical engineer, well trained, university-educated parents, encouraged me to read. I got very interested in science from a young age. I had a big workshop in the house, which is why I was always tinkering with things, and got into doing a lot of electrical types of things. I was very good at wiring stuff up, one way or another. I used to, probably only in maybe fourth, fifth, sixth grade, repair televisions for the neighbors. And, at that point in time, there--

Krewell: Did you have your own tube tester, or did you go down to the store?

Ditzel: No, I'd go to the store, and find tubes. Quite often, what would be wrong with the TV, a tube would be out. You could tell because the filament wasn't lit up, and that was enough. You didn't need the tube tester. And, so, I collected old televisions that didn't work, took them apart, had a big inventory of electronics parts, and got interested in amateur radio. I ended up building a lot of amateur radio gear, but found I was more interested in building the electronics than actually talking on the radio, when I got it done. While I was doing that, I also got introduced to programming. Fifth grade was, probably, the first time I wrote a computer program. It was on something called a Teletype Model 33 teletypewriter. It was fairly slow--

Krewell: You had access--

Ditzel: Our school had a timeshare access tools systems, so I got introduced to the BASIC programming language. And then, in junior high school, I ended up doing a lot more, and got torn between electronics and programming. And so, my whole career has been kind of a trade off between hardware and software and not being willing to give up either one.

Krewell: That's been perfect because it actually has been the nexus of those two areas where you have been so very successful.

Ditzel: It's helped me to be successful and see trade-offs that sometimes software people don't see hardware issues, and hardware people don't know exactly how software works, and so, it's helped me in thinking about processors. Even from a very early age, I was very interested in computers and tried to build my own. From maybe even only 12 years old, I tried to build a card punch reader. I took little three by five cards, punched holes in them, and it would light up a set of ray of lights on the board. I could say, what's 3 times 5? And the answer, 15, would be punched into the card. You put it in the machine, and the little reader would read it, and would light up 15.

I would impress all my parents and friends as to how I built a computer. Now, that did get more sophisticated, and turn into real computers later. About the time I was finishing high school, the world was getting the first eight-bit single chip microprocessors. That was a very exciting time. I ended up building my own home brew computer system. Learned how to build all the pieces, wire my own CPU board, memory board, storage system, paper tape reader. Wired my own keyboard up to it.

Krewell: Where did you get the information for them? Did you get the datasheets--

Ditzel: There was a magazine called Byte Magazine. I remember getting issue number one of Byte Magazine that said, how to build your own computer. As those came out monthly, you learned to read data sheets and put stuff together. A lot of it was just inventing yourself. It wasn't terribly complicated. People had very clever schemes, like how to use your audio tape recorder to plug in the microphone jack and record programs with sounds. There was enough performance in the processors to drive different audio frequencies, and people were pretty ingenious at little home brew pieces to build computers. I had access to timeshared computers in my high school.

I also was very fortunate. I lived in a university town while in high school in Ames, Iowa, the home of Iowa State University. I only lived a few minutes from the University, and I ended up in the summers getting jobs-- I got a job with one of the local computer research labs, working on something called the SYMBOL computer system. I was to be able to get in, and play with a computer that normally only graduate students and professors got to deal with, while I was in high school. That, plus being able to build my own computer, and tinker around, I think I was actually pretty involved for people at that time, to actually do a lot of programming, both hardware and software, still while at a high school age.

Krewell: Especially in that time period, it was extremely unusual to have that kind of access.

Ditzel: So, I would go and spend 10 hours a day in the computer lab. I'd get off school, and go and spend until one or two in the morning, doing programming. And, in the summers, work full time. So, initially helping out, but I got better at what I was doing. I ended up going to school at Iowa State later, and kept that job to help pay my way through school.

But, it really introduced me to some fairly deep concepts in computer architecture, much earlier than most students would have gotten. That helped me form a lot of ideas. The SYMBOL computer was particularly interesting, because it was a very early attempt at machine design by Fairchild Semiconductor in the early 1960s, to try and do a new generation of architecture that reduced the semantic gap between programming languages and hardware. They had almost a one-to-one mapping between elements in the programming language and elements in a binary format that was there.

Krewell: That was custom programming languages--

Ditzel: It was a special programming language that was invented for that machine. All the arithmetic was done in decimal arithmetic. Variable length up to 99 digits of precision. The machine only had basic data types of strings and structures. A number was just a type of string that was held. You could add to it, or you could do string operations on it. This whole machine, what's really amazing about it, is it had basically an operating system, a compiler, timeshared terminals, where you could get access, and all of this was done completely in individual logic gates. There were small, what's called a dual inline package, was invented for that machine. You only got two NAND gates per package roughly, or one flip-flop. There was no microcode ROM at all. There was no software operating system for the machine. You would turn it on, and the machine would sequence through a set of state machines, and run your program that way.

While it was a fascinating concept, it introduced me early to this notion of high level, language computer architecture, and doing very complex operations. Something we later called complex instruction set computers. Working on a machine like that for several years, helped me think about doing the opposite of something we later called reduced instruction set computers.

Krewell: CISC versus RISC.

Ditzel: Dave Patterson at UC Berkeley and I wrote a paper called, The Case for RISC, which reintroduced the notion of building very simple, but fast machines, as a better way to do computing. And somehow, trying to add all this complexity to the hardware, to try and help out the programmer. In the end, the goal is still the same. You want to run the user's program well. But, fundamentally, it doesn't really matter what level the ops are. The question is, how do you do it the best way?

Krewell: But, before you got to RISC, you did a lot of exploration and a lot of research on these very complex instruction set machines.

Ditzel: I got to work on the SYMBOL computer, which was a very complex machine. I did a lot of programming while at the University on IBM 360s, Digital Equipment Corporation PDP-11s, and other machines. I kind of knew the standard machines from the assembly language level as well as where to go. I guess my mission in life at the time, was to figure out how to design better computers. It is something I've had a passion for my whole life, from a very young age, maybe 10, 12 years old, up through today. It's something I just can't let go of.

Krewell: It's a lifelong avocation, or vocation for you, I should say. But, just before we leave the SYMBOL project, that machine never really made it into a large scale commercial--

Ditzel: No it never made it to market. It was started by Fairchild, and before the project was finished, they ended up canceling it. They transferred the 90% finished machine to Iowa State University, who ended up getting it working, and spent the next, maybe 5 to 10 years, trying to get bugs out of the machine. You could imagine, if there's a bug in software, you go and you edit it, you recompile. It is very quick. But, in this machine, you actually, when you found a bug, you actually had to go in and solder wires on the board, and change the logic gates. It really drove home the notion that software programmability is much easier, when you don't have something specified absolutely perfectly at the beginning, rather than completely hard wiring something.

Krewell: But, that machine had some disadvantages as well. In addition to the complexity. In fact, you had to physically change the design to fix a bug. I think you also indicated when we talked earlier, it also ran fairly slowly.

Ditzel: Yeah, I mean it used a lot of chips. There were 20,000 individual chips in there. For me, the impetus for reduced instruction set computing was to say, what if you would use those 20,000 chips in a different way? Instead of doing one instruction, that may take 10,000 clocks serially, then going on to the next one, what if you build a pipelined instruction set that was very simple and had overlap?

A lot of those ideas, coming from SYMBOL and complex instruction sets, and what didn't work well, were formative in some of the ideas that moved forward, early RISC ideas, at least on my part. There were a lot of other people thinking about RISC concepts and those ideas, as well. The whole notion kind of gelled around the 1980 time frame, when Dave Patterson and I wrote, The Case for RISC.

There were great debates that were held for several years at conferences. First, it's a totally stupid idea, and then, well, it kind of makes sense, in some situations. And then later on, eventually takes over, as everybody's saying, well, yes, of course, we all understood this was the right thing to do.

Krewell: Before we jump ahead, let's stick a little more chronologically. After Iowa State, you wound up going to work for AT&T Bell Labs. Isn't that correct?

Ditzel: I did. I took a job at AT&T Bell Labs, and part of the reason I took it there, is they offered to pay for my graduate school education. How I ended up getting connected up there was, there were some recruiters on campus, and I said, look, I'm interested in building a machine for the C programming language. They said, oh, we have a group actually in our research group, and Murray Hill (NJ) was looking at this. We'll connect you up. As an undergraduate, I got an interview for the group that typically only hired very senior PhDs. I ended up being the youngest, I think, professional engineer they'd ever hired into that team.

This is a little group of 30 people at Bell Laboratories, which was maybe 50,000 to 100,000 people-- very large organization. But, this is a small group that invented the UNIX operating system, the C programming language, C++, UNIX networking, and we did early RISC microprocessors together on a team. This was about 30 different people that did a lot of these different things. So it was a great and exciting time to explore a lot of different options, if you could do your study project, with just a couple of people.

Krewell: Those must have pretty much kept you busy, between your graduate work and that project?

Ditzel: Yeah, I ended up, Bell Labs sent me to UC Berkeley, or helped me to go there. I had to get in on my own. I chose Berkeley to work for Dave Patterson, who also studied high level language computer architecture. I had a great time there. Berkeley had accepted a very large number of graduate students those particular years. Therefore, they could put a large number of people on a project. Something, that say companies could do, maybe 50 people on a project. Dave Patterson, and a couple of other professors there, were just extremely good at organizing this large number of students, motivating them into something coherent. That's how the various series of RISC projects at Berkeley got going.

Krewell: Was that DARPA funding, partially, at least?

Ditzel: I'm not sure exactly where all the funding came from. I think it maybe was a combination of NSF, DARPA, a few other places. As students, you don't really worry about that. You just wonder whether you have an office and are being paid and can take classes.

Krewell: And, how was it working with Dave Patterson?

Ditzel: Dave Patterson was absolutely great. Really great visionary leader, very motivational. Great at setting up various events and the students would come. He would bring in outside industry people. It was really a superb place to get an education. He's done just a number of absolutely great things over his career, and it was a pleasure for me to be able to work with him for a while, as well as after I graduated.

After I had been at Bell labs for a while, I moved to Sun Microsystems. One of my first tasks at Sun Microsystems, was to actually hire and work with Dave Patterson, as a consultant. He actually had been working with Sun for a while, and he was helping Sun on SPARC architecture, which was kind of a next evolutionary step from the Berkeley RISC-I and RISC-II architectures.

Krewell: I guess Sun started off using 68000s, I believe.

Ditzel: Yes, Sun started off as a company using the Motorola 68000 processor, but they saw the wave of RISC processors coming. I think they had the brilliant insight, this was done by Bill Joy, Andy Bechtolsheim, and some others, to say, well, if the RISC architecture is so simple, let's open it up, and give it away, and we'll have lots of people help us implement this. When I was at Sun, I was the CTO for the SPARC architecture, as part of the SPARC technology business. It was very exciting, because instead of having one company do one CPU project, there were maybe 10 or 15 companies, all doing different SPARC projects, all hoping to get some of Sun's business.

In the end, this was great for Sun, because they could just pick off the best of whatever resulted from this. They got more innovation that way. Things moved very quickly in the industry. The other thing that's really great about RISC processors-- at Sun, for example, the very first SPARC CPU, was done by just two engineers.

Krewell: So, simple designs?

Ditzel: Fairly simple design. Today, having been at Intel, you would have several hundred engineers, working in one way or another, to try and do a new project like that. It allowed a lot of different companies-- say, RISC, that is, a lot of different companies to go off and to build their own CPUs, to try out different things. There was a lot of innovation in the industry. And I think it really helped propel things forward at a great rate.

Krewell: We've jumped ahead again. I'd like to actually back up again to Berkeley, because there's some other stuff you did at Berkeley-- the X-Tree system, and some of the stuff you were involved with there. I thought maybe you wanted to talk a little about what you did, in addition to--

Ditzel: Well, largely, I was studying and taking classes. There was a research project. What I worked on was called X-Tree I did the processor proposal for X-Tree. It was a tagged architecture. I tried to take some of the ideas that I'd seen in the SYMBOL computer system, and try and do a more modern version of that, to see if we could make it better. I think, in the end, my conclusion was, as much as I had tried to put in advanced concepts of the time, like tagged architectures, and descriptors, and other things in the machine, that the notion of a simple RISC machine, in the end, actually ended up being more appealing.

I did the good thing as a student, which is, I wrote my thesis on the X-Tree processor. But, immediately after that, I wrote up what I call my anti-thesis. Which is, we wrote the Case for RISC, which said, we've been studying this field of high level language computer architecture and as we're working on it, we're not so sure it's really the right path. I think that's really what started the movement back towards simpler architectures. If that hadn't happened, who knows what kind of complex machines we might have today.

Krewell: There were other RISC projects going on at Stanford and IBM. It seemed to be a number of different researchers came to the same conclusion. Mostly because they did share information. It quickly became--

Ditzel: I'll tell you one quick story about IBM. IBM had an early RISC project called the 801. 801 was actually just the building number for IBM T.J. Watson Research, but it was called the 801 machine for that reason. For quite a long time, the management at IBM didn't allow people to publish about this. But rumors were getting out about this particular computer about the same time. I was actually involved in starting a number of conferences. One of the conferences was architectural support for programming languages and operating systems.

The rumors had gotten out, and it certainly helped AT&T and others, that IBM had this incredible RISC processor that was 10 times faster than anything else out there. If you didn't get into RISC now, IBM would totally dominate the industry. And so, I think that actually motivated other people to do RISC.

At one point though, one funny story is that we tried to get IBM to submit a paper for the ASPLOS conference, and they actually submitted it, and then later, they were denied being able to publish it. I wrote a paper to the head of IBM research, and said, look, everybody's doing work in this field, we just want to give IBM credit that it's due for doing this early work. And, I remember, IBM, in the end, did give permission to publish.

Somebody told me the story later, that they had taken my letter, stapled it on another letter at IBM, that said, roughly, it's come to a sad state of affairs when Bell Labs is out to protect the reputation of IBM. Simply because we wanted to be able to cite them appropriately for the great work that they had done, with John Cocke and Marty Hopkins and others, who'd worked on the IBM 801 machine. I think they really had done a very nice job of that, but some of the ideas-- once the basic idea is out, groups like the Berkeley RISC-I Team were able to do equivalent RISC architectures that were equally good, as well as many other companies, Sun, MIPS-- I could name a host of small companies that all had their own RISC architecture, because it was something you could do a small number of engineers.

Krewell: Relatively straightforward design, didn't require microcode. There were a lot of advantages on silicon implementation point. It put a lot more burden on the software side, because the compilers had to be more aware of and handle more of the work.

Ditzel: I think, maybe it's misunderstood. One of the quotes I like best about compilers is, an issue between MIPS and SPARC maybe, was that SPARC had register windows to make it easier on compilers. John Hennesey, who had founded MIPS, said, look, the compiler's only an issue until you have it written. Once it's done, you don't really have an issue with the compiler anymore. In many ways, the flat register file that MIPS used as a model, was fairly easy for compilers to use. They could do a very good job of optimization with that. The register window scheme-- the register windows took a lot of chip area at the time. While there are some good concepts there, depending on the implementation of the technology you're using, it could vary.

But, again, I think once you have compilers, you're pretty much insulated from a lot of the details of the instruction set. Really, it didn't seem to matter so much whether you had a RISC-like instruction set, or a VAX-like instruction set, I think it had less impact on software programmers, than people had initially assumed it might. Particularly, as the world moved to high level programming languages. And, again, you have to remember just slightly earlier than the RISC days, people were still questioning whether you could actually use high level languages effectively for performance. There are still a lot of people writing in assembly language.

Krewell: There was-- even C was considered slow by some people, and C's a very low level--

Ditzel: Complex instruction sets, like you had on the Digital Equipment Corporation VAX, were actually fairly nice for programmers to use, if they were going to write in assembly language. They probably turn out to be less of a benefit for compiler writers than the original designers of that instruction set might have hoped. I think as much as hardware engineers want to help the software, quite often they create more problems than they help in making these trade offs.

Krewell: Actually, I guess, the simplified set of instructions make it, in some ways, easier for a compiler writer. They don't have to worry about very complex instructions to figure out which ones to use at any point.

Ditzel: Quite often, the actual compiler writers prefer simple, orthogonal, regular instruction sets-- is perfectly fine for what they're doing. As long as you can process the instructions at a sufficient rate. The theory was, if you could build a RISC chip that processed instructions maybe at twice the rate, if you used twice as many instructions, it didn't matter in the end. You'd still come out ahead.

Krewell: In the end, you and Professor Patterson wrote the Case for the Reduced Instruction Set Computer, which became the seminal paper in that time frame. Any interesting stories behind that? What was the reception?

Ditzel: There was a huge amount of skepticism. And I think what made the paper interesting is it came out at roughly the same time as a paper called The Case Against RISC. We had pre-circulated the paper. And, I remember Doug Clark at Digital Equipment Corporation, who was one of the designers of the

VAXs and the earlier computers at DEC, didn't believe in this at all. He said, you just going back to building a PDP-8, or something. There's people had a little knock offs on why they didn't think this was going to work without really fully understanding the concepts. And, what I thought was really interesting, was three or four years later, Doug Clark, himself, wrote a paper called, basically, we were wrong, this stuff actually does work pretty well.

Krewell: Was that before or after Alpha?

Ditzel: That was before Alpha. But, I think, along the way, they had a RISC architecture, called Prism at DEC, that was a step along that path. Even today, though, you could say, well, maybe RISC has won, because most of the complex architectures like the x86. Both, Intel and AMD, for example, describe their implementation as breaking the complex x86 ops into simpler, RISC-like ops. AMD calls them ROPS, and Intel calls them UOPS. But, they're basically RISC-like operations. The question is, did RISC actually win inside? Well, you could say, well maybe it did, from that point of view.

Krewell: From an architecture point of view, it did win. I think Mike Johnson's book, the blue book (Superscalar Microprocessor Design, 1991), was the first one that actually sort of dropped that in his textbook. Here, you could do it x86 implementation, by translating the instructions into simple ops.

Ditzel: I think that really makes the key point, that you can separate the notion of the instruction set the programmer uses from how you implement it. What they were really doing in translating into RISC ops was a form of binary translation. They were doing binary translation, with hardware, of one instruction to two or three RISC-like operations. But that whole notion of binary translation can also be done by software. That led into a whole other set of fields of research.

Krewell: That leads us into Transmeta. Maybe before we do Transmeta, let's talk a little bit about your time at Sun. That built up to what became Transmeta.

Ditzel: Yeah. I done some early RISC work at AT&T Bell Labs for about 10 years. AT&T had both gotten into and gotten out of the computer business within a few years. I wanted to go work for a company that was really dedicated to RISC processors. So I ended up changing jobs, moving to California, and joining Sun Microsystems. It was about maybe 1,000 people or so.

I'd gone to school with Bill Joy and Eric Schmidt at Berkeley, and I actually tried to hire Andy Bechtolsheim from Stanford to come work at Bell Labs. I was working on bitmap displays at the time like he was working with a Sun terminal. In the end, it turned out to be the opposite, they ended up hiring me. I had a great time, basically helping push forward a lot of different SPARC designs, both within Sun and outside. A lot of my focus was on the instruction set architecture, enhancing the 32-bit instruction set architecture they had when SPARC was first announced.

Then I lead a team that developed the 64-bit extensions to SPARC. That was done in a fairly interesting way, because when you did something like a 64-bit machine, the tendency in the past was, you throw everything away, you do something new. When you move from a 16-bit PDP-11 to a VAX, it was a totally new machine architecture. What was different about what was done with SPARC was we did that extension in a totally upwards compatible way.

I think a lot of the lessons on that probably were perceived by AMD when they did the 64-bit extensions for x86. To do it in upwards compatible way rather than throw it away, and go to a new architecture like DEC did with the Alpha. While the Alpha was a really great architecture, very simple, very cleanly designed, the fact that it had to start from zero again with binary compatibility, gave it a bit of a challenge there. Extending an architecture tends to be the easier thing that companies tend to do, even if it doesn't result in the nicest machine in the end. Just overall, there's a lesson there-- a lot of the complexity and cruft we see in instruction sets that we don't like, is there because of long periods of evolution.

Krewell: Intel tried going to Itanium for the 64-bit solution, and that has completely, pretty much flopped at this point.

Ditzel: I think Itanium had some of its own challenges-- one, by not maintaining compatibility. Something that you would think they would have realized was important. But also--

Krewell: They sort of did. They had the x86 emulation mode that really ran very, very poorly, but it was there.

Ditzel: Not enough to make users happy. I think there were also some things designed into the instruction set of the machine, like defining the load use latency as part of the instruction set. It might have been great for the first implementation, but for the next couple of implementations, they were locked into things that, in the end, meant that they, on average, I think, lost about a factor of 2 in megahertz, compared to what the x86 implementation teams were able to do. Although the Itanium architecture had some really nice benefits in parallelism, it's really hard to make up a factor of two that you've lost in megahertz.

Krewell: Then also, the predication they built into it-- I've talked to people that have indicated that when they look back at it, that the extra cost of the predication was a wash with the advantages of a predicated operation.

Ditzel: I've looked a lot of new architecture designs. I have a rule that anytime somebody does a completely fresh generation, you'll tend to find lots of new ideas. If you take maybe 100 new ideas in Itanium, or 100 new ideas in the i432, or again, any other major architecture, what you tend to find is, maybe 10 of those 100 ideas turn out to be good ones. The other 90 are ones that are just weighing you down. I think that the trick over time is to learn how to evolve, and pick the good things. I still do not think, even today, the industry really understands well how to do brand new machines, how to take some risks in terms of new innovations, but how--

Krewell: Risk with a k--

Ditzel: Risk with a k-- yeah-- how to figure out how to come up with, on the whole, really good things. I think there's so much demand now, that the first product you have out the door, has to be a great success, that people tend to do more evolutionary things. I think that's why we see a real slow down in innovation in computer architecture.

Krewell: Yeah, certainly anybody trying to build a brand new chip from the ground up, it's about $100 million complex project. By the time you've got that $100 million done, you've got to start on the next one.

Ditzel: In some sense, those economics were the impetus for starting Transmeta. When I was with Sun Microsystems, we had a number of SPARC projects, but I realized, at some point, that if we looked at binary translation techniques, we could actually separate the notion of the physical implementation, the hardware, from the instruction set you were running. The SPARC CPUs, as a RISC chip, felt they could build the hardware to be faster. But, with the notion of binary translation, we could take better hardware and move compatibility onto it.

That was really the notion for starting Transmeta. I think there's also a view that big companies find it very hard to make big changes. By doing a start up, we knew that Transmeta would either succeed or fail, whether we made binary translation work. Transmeta did binary translation of the x86 code onto what was, essentially, a VLIW style architecture, with a completely RISC-like instruction set. You could put a number of 32-bit RISC-like ops together in one longer instruction word, and then the binary translator would figure out how to do exactly the same semantics the user had intended on x86, to do it on that hardware. Exactly how you design that instruction set to do that with high efficiency, I think, is really the key. But, it's fairly clear now, something that wasn't clear then, that this could be done with high efficiency.

Krewell: Now, did you have this idea when you were still at Sun, or do these ideas about the VLIW and emulation layer on top of that?

Ditzel: The general notion of VLIW being the way to get good performance was around at the time. There were various companies, like Multiflow, Cydrome, and various DSP chips. While SPARC started out as a nice RISC machine, and one of the fundamental principles of early RISC machines, is you do one instruction per clock. But, with the quest for more performance, soon people want to do two instructions or three instructions, so you were ending up doing variable length. One of the things about VLIW is just grouping those encodings together. There's multiple kinds of VLIW. Some with giant, fixed length, like Multiflow, you had to have at least 32 instructions at a time.

Krewell: We should, probably just for some people-- VLIW stands for very--

Ditzel: Very long instruction word.

Krewell: Where you take multiple instructions and pack them into one operation.

Ditzel: My view is a little more liberal in the definition of VLIW, which is to take 32-bit RISC instructions and putting a variable number there, but putting coding in the actual instructions to say how many of those instructions are put together. Because, if you don't do that, and have to do what's called super scaler execution. It takes a lot of extra hardware to look at all the instructions coming up, decide how many you can issue at the time, where are the dependencies. All of this can actually be checked very simply in software.

Krewell: You can even have slots there aren't being used, so you to put in no op basic functions?

Ditzel: At times you can. When we started Transmeta, nobody really knew if you could do that. Most of the binary translation work that was done prior to that, was what was called static translation. I think what was new at Transmeta was the notion that you could use dynamic translation, and still be fast enough to actually outperform the pure hardware implemented processors.

Krewell: By static, the compilers had to do all the code optimization ahead of time, schedule the instructions, a priori, and the problem, I think, that other people ran into, is that that doesn't work well in environments where there's a lot of dynamicism, like say, JAVA and such. You don't have all that static knowledge at compile time.

Ditzel: Yeah, the static compiler would try to take a binary for some machine, maybe it was x86, and translate it to say, a lot of people want to say, DEC had an Alpha translator.

Krewell: IBM was rumored to have it.

Ditzel: Various people tried this, but again, you try and convert the whole program to a new program that you would run. Again, there are some things you just can't figure out at compile time. You don't know what two pointers point to the same thing or not. You have to have some runtime support. I think, what we learned in thinking about this at Transmeta, was a little bit of extra hardware support, to help the dynamic translator, could make a huge difference. On a machine not designed for binary translation I think you can get within about a factor two with negative performance, is about as good as people do, maybe a little bit better, but not much. But with hardware support, you can actually come up to being equivalent, or actually higher performance, than the machine of similar hardware capability could be. I think that's the trick, is learning what are those special hardware support operations you need in order to make binary translation go fast.

Krewell: How did Transmeta come together? Where did the funding come from? Who were all the people, the team that got put together, to start the project?

Ditzel: Transmeta came together with some people who were maybe a little disillusioned at various companies about how the projects weren't as exciting as they once were, and wanted to go off and do something different. Along, kind of, with the generalized notion that we had that you could virtualize the instruction set, and so actually implementing a particular instruction set didn't have to be the same in hardware as what you were going to run in the user program.

Also, I think, what was becoming very clear at the time, was the rise in popularity of the PC architecture and the Microsoft software together meant that there was this install base of software out there, that the RISC world just hadn't kept up with. There was clearly a large market there, deserving to be served. We felt that the very complex architectures implementing x86, purely in hardware at the time, could be beaten by building VLIW techniques with binary translation. It was a way to build a better mouse trap.

We thought we could do that. We had people with general expertise in the area. It just kind of came together all at once. So we decided, hey, let's go off and start a company. There was, I think, a group of eight to ten people or something who got together. We got some initial funding from DARPA to actually go off and study some unrelated topic. We actually were studying cryogenic electronics at the beginning, and that gave us enough money to kind of get started, to get some initial concepts together, some data to prove our point, and then we started off going and seeing venture capitalists. This was at a time when Silicon Valley, you could raise money for new microprocessors reasonably well. We got Transmeta funded that way.

Krewell: As the concepts were put together, was it right from the very beginning of VLIW where code morphing was your initial concept?

Ditzel: No, it actually did not evolve very much. We knew at the beginning we wanted to do a VLIW-style architecture. There were multiple piece parts. We had a concept of how to build a new type of binary translator, a new type of a dynamic translator, in multiple stages that hadn't been done before.

Krewell: Had there been any research papers on it before you guys did the work? Was there was any research that led into the binary dynamic translation?

Ditzel: We had seen some other work where people had just done dynamic translators. Let's say, there was one experiment we had done trying to run MIPS instructions on the SPARC architecture, merely for comparing the two architectures together. You know you typically have these benchmarking wars, where you want to know how you're doing versus your competition. It was helpful from a tool analysis point of view to measure dynamic instruction frequencies, not only on your own machine, but what does your competitor do by measuring the same tool suite.

What we found is that by using a little bit of the dynamic translation techniques, that we could do, within a factor of two, the native speed of the SPARC while running the MIPS instruction set. The question is, well, where's that factor of two being lost? As we thought about it, we realized that it was the mismatch between what you're translating from and to, and if that mismatch is too big, you could do it, but you would have to have a longer sequence of instructions. Our notion was, look, if we went and did a custom designed instruction set, it knew it was going to be translating from x86, we could do a RISC-like instruction set to translate to, then we could translate to with very high efficiency. We ended up having to do the implementations from scratch after we started Transmeta.

But, this general notion that, if we could get special purpose hardware support, that we thought we could do a much better job. We knew we could already get within a factor of two of the base hardware, even without the hardware support. Then we felt like we could build a computing engine that was more than a factor two faster as well.

Overall, the combination of those things led us to believe that we could build a substantially faster, simpler, lower-power processor architecture by using a simple machine with binary translation on the top. In the end, although we had to refine the technique somewhat, the basic technique was pretty much the same. We learned that in binary translation, we needed to move from a two gear translation system to a four gear system.

Krewell: You jumped ahead--maybe you could explain--

Ditzel: I'll explain that a little bit. If you simply take a program and dynamically start to translate instructions one at a time, that's basically interpreting each instruction one at a time. That could be very slow. You could take 100 machine cycles to one native machine cycle. So, 100 to 1 slow down-- that wouldn't be good. But, if you try to binary translate large parts of the program, do a lot of optimization on it, you could find also you're taking too much time there.

What we learned is to step up the amount of time taken during translation with the amount of optimization. What you'll tend to find in programs is that, maybe 80%, 90% of the instructions being run, run very infrequently. If you take a lot of cycles to run, it doesn't really matter, because you don't run them very often. But, maybe 10% of the code is where you spend 90% of your time. If you could optimize that code very well, you could afford to take a lot of CPU cycles to do the optimization, and then amortize that across all of the execution. Balancing out how much time you take to translate versus what the quality of the code as you go there, was one of the techniques Transmeta developed into what we call a four gear binary translation system.

Krewell: As you do that optimization, you store that optimized code in, basically, a cache, so rather than executing the original--

Ditzel: So the next time you come back to it, you already have pre-translated code and you don't have to do the translation again.

Krewell: And, that's where you get your real speed--

Ditzel: Now these are techniques also, again in 1995, when we started, people didn't know whether this was going to work very well. But, in fact, another place where this is being used very successfully is in the implementation of Java. Java started off as an interpreted language. Later, they turned to dynamic translators, and got very good performance. Those set techniques tended to run in parallel. And again, translating from Java to the underlying machine, worked pretty well. The job we took on at Transmeta was harder, because we needed to get very good efficiency, because we had to compete against hardware solutions. Plus, we wanted to run the operating system level instructions, which typically you don't have to do on a Java style dynamic translation system. Again, having the right hardware support underneath was what made the difference there.

Krewell: You're getting away from the idea of trying to do a Java machine, Java or something like that.

Ditzel: I think because it crosses the boundaries of both hardware and software, it's a little difficult for people to necessarily accept. If you're hardware person, you're skeptical of the software. If you're a software person, you're not really sure what hardware to get. But, slowly people are becoming more accustomed to this, and the techniques have gone on to be used elsewhere. NVIDIA has a CPU now called the Denver CPU, which uses techniques very similar to what Transmeta used. They actually licensed some of the Transmeta technology, and hired some of the Transmeta people. There's a learning curve one gets on of learning how to do this better and better.

I went off to Intel, while the NVIDIA guys, ex-Transmeta guys, were there. I had a different set of ex-Transmeta guys at Intel. And we were studying similar concepts, but maybe along a slightly different direction than they had done. We tried to advance the state of the art, in particular, to try and understand what is the trade off between doing binary translation on VLIW versus doing binary translation on out-of-order hardware. I think that was not well understood, and probably, today is not well understood.

I think one of things we learned at Intel is that binary translation on top of out-of-order hardware actually worked far better than anybody expected. There was great controversy about that even while we studied the project. We had people on our own team that, when we wanted to move from a VLIW in-order issue machine to out-of-order, simply said, look, it's impossible, you won't get more than about 5% more performance, I'll stake my reputation on that. These were people who were very well respected in the architecture community.

Krewell: Another concern was that the out of order machine is much more complex.

Ditzel: The feeling was, and this is something that's still said, for example, by the people at NVIDIA, that by rescheduling in the binary translator, on top of in-order hardware, you get many of the benefits of out of order, and you're very close there. We initially believed that at Intel and we pushed the envelope as hard as we could there. But, in the end, we actually had the opportunity to do a nice experiment of also building an out-of-order engine, and comparing those two, and to see, would we get more than 5% or not, and would the hardware complexity explode?

The surprising thing, and I'll say this in very generic terms. I'm not going to give away any real secret details that aren't very generally known in the industry now, but what happened, as a result of that experiment, was the hardware complexity only made the chip bigger by about 5%. OK? The performance almost doubled. I will say I was wrong about how much we could gain by having out-of-order. I think the difference is that they're just certain things that are dynamic, like cache misses, TLB misses, other issues, that are very difficult for a software scheduler to do before the program executes. I think it's an important notion to be able to handle these dynamically different events from things that are statically schedulable. There's still a lot to be done in that area that, I think, will take the next 10 years, before we see it to fruition.

Krewell: There's still a lot of interesting architecture work you think that should be done.

Ditzel: I think there's a lot that could be done. My dismay is a lot of it is not being done, because so many of the companies that are still in business are doing small, incremental steps. It used to be, maybe 10 years ago, Intel had a tick and a tock scheme. Whereas, a tock would be a completely new machine architecture. And then when you moved to a new process generation, they do a tick, which moved that well-established, known, working architecture to a new semiconductor process technology. That way you weren't taking the risk of process technology and architecture at the same time. But, what's happened is, because it's so hard to make changes, the notion these days, this is my personal view, is a tick is maybe only a 5% change in the architecture, and a tock, instead of being 100% change, is maybe a 10% change, at most.

Krewell: It's more refinement than it is any kind of big change.

Ditzel: Exactly right. People learned, I can improve this little thing here or there, but I think the pace of innovation has slowed down tremendously, because people aren't willing to take the risks, and the amount of learning curve it takes to get a new generation architecture going. I still think there's some huge potential out there. There's factors of 2 to 4x, and there's people exploring various options. I'm not sure I'm a believer on all the different techniques out there, but I'm very delighted to see people looking at these different techniques.

Because, the other thing that's happening in the industry is there are many fewer companies actually doing new CPU designs. I'd say in the early days of RISC, there were maybe 100 companies doing their own RISC designs, because you only needed two engineers to do a RISC CPU, right? Now, it takes several hundred engineers, and we're finding it's really down to a small number of companies. Apple and NVIDIA, ARM, and a few others--

Krewell: They're doing it under an instruction set license from ARM. And ARM itself is really conservative.

Ditzel: It's not the same as a brand new computer architecture. It's very tough for universities to do, because in order to beat the current machines, which are pretty good, the amount of effort and infrastructure you have to build is such that it's really not very practical at universities to look for a new breakthrough in computer architecture that way.

Krewell: What about your alma mater? They have their RISC-V, I guess it is.

Ditzel: Berkeley is trying to help people do research in using processors by having a standardized instruction set, and making it very open. I don't think that the computer architecture work I've seen there is very innovative. It's more establishing a standard architecture that people want to put in an internet of things device, and have free and open access. I'm really a big supporter of that project from that standpoint. But, I don't know that it really is helping us study instruction sets, or new pipeline architectures, or different methods of computation. In a way, it's really it's attacking a different problem.

Krewell: Actually, in a sense though, maybe it's hurting more than helping, because if I come in with a fixed research architecture, where's the research in new architecture?

Ditzel: Yeah, I'll give you a particular point of view here, which I can say is one of the people who helped start RISC, is, I think, the RISC guys have gone off the tracks a little bit here, in two ways. One, and this is something I felt when I was at Sun Microsystems, that a lot of engineers felt because they were given a RISC architecture, RISC instruction set, to work with that they would inherently be better than any complex instruction set architecture. That just wasn't the case.

When Intel doubled down, and really put a lot of effort on the implementation, they found they could mimic and get around some the inefficiency of CISC, and build pretty good machines. Having a RISC instruction set is not a birthright that you will go faster. Maybe it was a temporary advantage. I think people had viewed it as a permanent advantage. I think at the time, also, RISC was being done, the machines we were competing against, machines like the VAX architecture and others, that might have taken tens or hundreds of cycles per instruction. And by doing the RISC instruction set that was pipelined, at one instruction per clock, we were making a conscious trade off to get that constant throughput even though it may take two or three times as many RISC instructions to do a program than one using a more complex instruction set. I think that's been forgotten by the world as well.

If I look at the new 64-bit ARM instruction set out there, it's a very generic RISC. And I think it's a really excellent RISC architecture, but it could've been done in the 1980s. So, other than being 64-bits wide, and having lots of SIMD instructions, or something else, there's nothing terribly innovative there that couldn't have been done 20 years prior. I think that's not the be all and end all of things. I think there will be another wave in the future with more factors of 2 or 4x available just in basic instruction execution, which are available. I think there's new opportunities for doing new styles of architecture.

Something, I'm personally excited about is, how would you design computer architecture, generally, if you could think in 3D? I think we're getting to the point where stacking chips in three dimensions is going to make the trade offs differently. Whereas, it used to be anywhere you could go on a chip, was 100 of times faster than going across chip boundaries to another chip. Well, now with 3D stacking, it's going to turn out that actually you could go to a chip on top of you maybe 5 to 10 times faster than you can go across your own chip. That means we ought to be rethinking computer architecture very fundamentally in the regime of having technology which can do 3D stacking.

Krewell: In the case where you do 3D stacking, often its stacking memory on top of a processor, so you get very high bandwidth access from the processor to the memory. Memory bandwidth has always been a battle, and that was always one of the arguments, CISC was an advantage. It had less pressure on the memory bandwidth than RISC did.

Ditzel: Sure, but as technology shrinks, wire resistance is going up proportionately so that it takes you longer to get at some units. If you're in the main CPU, and you want to talk to your SIMD floating point unit, it may take a lot of clocks to go over across the chip. Maybe putting a SIMD unit on top of the main execution unit, you could actually get to it faster. So maybe we ought to be stacking our floating point units rather than just thinking of stacking memory. Stacking memories is something we should be doing. But, there's, I think, more exciting alternatives that could be contemplated, as well.

Krewell: Then again, it's thinking about stacking internal 3D, so, in a sense, do we need an architecture that can handle the interconnection of all the elements in 3D? That's an interesting--

Ditzel: I think it will open some new opportunities for how we partition things. I think one of the other things that has to happen still, we don't quite yet have today, is another generation to a lower power electronics. If we can get new materials that help us have transistors that operate at substantially lower supply voltages, that will get the power on the chips down, such that, if you stack the chips, the cube of chips you stack doesn't melt. That's still a little bit of an issue here. But, I think as technology changes in the future, I have every expectation that the power per die will go down such that you can stack chips without cooling problems.

Krewell: That would indicate we need to start really thinking about some radically new ways of building processors and building computers.

Ditzel: I think there's some big opportunities there, so for any students out there who want to know what to study in computer architecture, thinking about, how do you think in 3D, might be a place to start.

Krewell: Actually, now we're starting to get a point where it's not just software and hardware and chip access, it's also packaging access.

Ditzel: Yeah, well, the entire system will get integrated, right? Today you've got your SoC, and your DRAM, and your flash memory and disk, or whatever, well, we just saw, even this week, people make a major new announcement, a new memory type from Micron and Intel, for storage that's going to be maybe 10 times as dense as what we'd seen before. If we package that, you'll get the equivalent of all your CPUs, all of your memory, in one small cube in the future, right? If it's got radio built into that, OK, so you apply power, and it could wirelessly communicate with anything.

Krewell: Maybe the power would be wireless, too.

Ditzel: That's possible.

Krewell: Actually, something we skipped over earlier, and this was one of your heroes, was Nikola Tesla, who was big into wireless.

Ditzel: Big into wireless power transmission. Of all the great things he did, that's the one, I perhaps, am least convinced of. But, it depends on the amount of power you need. If the power levels are extremely small, than wireless transmission is pretty reasonable. With the amount of power we want to consume in our devices today, I think we're still pretty far off in being able to do that in any widespread, practical use.

Krewell: So, we still need to get those little cubicle processors in computers in the future-- super dense and super interconnected.

Ditzel: I think, particularly with the push toward Internet-of-Things devices, you're going to want devices that are so low powered, you could power them off a little hearing aid cell size small battery for five years. When we can build our computing elements, there are these little, small squares that we can stick on the wall anywhere, and they've got video cameras, and radio, and are communicating with us. Perhaps we just need to take a giant box, throw a million chips in the box, and ask them to compute something.

Krewell: That's like the compute motes that are just scattered about, and they interconnect with each other. They're a mesh.

Ditzel: There is interesting work to be done in computer architecture. But, I see less and less going on in the main CPU core each year. I think it's a little less predictable when we're going to get a next breakthrough.

Krewell: Any idea who or where that would come from, if it did come from somewhere?

Ditzel: Right now, very hard to see, because very few companies are doing, again, the kind of mainstream computer research. I think people think that the computer design is a commodity. It doesn't matter what your instruction set is, it's all the same. I don't really buy into that quite as much yet, but certainly, I think there's less and less funding for doing new and innovative things.

Krewell: I'd like to jump back first for a little bit. I got to know you most when you started Transmeta. You were a very visible representative of Transmeta. How would you describe that whole experience of starting the company? Getting very well known, becoming the center of attention. And becoming the center of attention that Intel, from a negative point of view, you were enemy number one at Intel for awhile. And then, the product while was very fundamentally successful, in that it functioned well. Ultimately, the company failed. How would you describe that whole experience?

Ditzel: A fascinating experience. It was great to go through. I think it was a real opportunity to make bigger steps in architectural approaches than might have been possible at a larger company, because people just didn't want to take the risk. And for venture capital people, unless you make a big step, you're not going to get a big return. They actually do want to support you taking a big risk there. But even at the time, it was difficult to raise money. We raised on the order $250 million to get our first product out the door. That's not an easy thing to do necessarily. There weren't a lot of companies doing that. It was great to be able to represent doing innovation. That helped us to go out and hire great other engineers. We assembled a great team of people.

Krewell: —Linus Torvalds

Ditzel: Various-- Linus and various others. I heard Linus say once that he liked working at Transmeta because he could feel like one of the average programmers. Sometimes he didn't like being treated special, which he gets treated specially sometimes for having done the Linux work, and which he deserves. But, I think it was enough of a challenge, and other people thought it was a great challenge. We weren't sure whether it could be done, or how well we could do it. I think it was great unexplored territory. .

In the end, Transmeta didn't make it, and I'll just say here I think it was not because of the technologies. It actually turned out to be manufacturing problems. We switched fabs. The fab we switched to actually had a problem, and couldn't actually make our chips for a year. We had to prove where the issue was in the fab, and what was happening. They're not happy if I speak too much about those details. But, the technology itself worked fine.

In fact, I was able to go to Intel. We started another team. We had, at one point over 200 people working on a project there, and they were all convinced that this was a good project to do, as well as many others, along with a fair number skeptics as well. But, certainly convinced in the notion that this could be something really new and interesting that could provide some substantial benefits.

Now, in any project you do at any big company, it's not just a technology issue, you realize. There's a survival issue. As you go through and get approvals, you might have to go through 1,000 meetings, where you'd get approval to go to the next step. It only takes one meeting to say no. In my career, I've done probably 20 different processor designs and pipelines. A lot of which have been canceled. The first couple times that happens, it's extremely painful. But, it's really part of the ecosystem you have to live in. Companies are going, conditions change, they need a slightly different product. I remember one project we had at Sun Microsystems. We were doing a gallium arsenide SPARC CPU. It was going to have an amazingly fast cycle time for the time. We were doing--

Krewell: Wasn't (Seymour) Cray doing their gallium arsenide?

Ditzel: We were using similar technology to Cray. In fact, a little bit too similar. Cray and Sun were both using Gigabit technology. And what had happened to the project, at one point is, Cray actually went and bought Gigabit, therefore, making the technology unavailable to other people. We looked at switching to a different gallium arsenide maker. And we could have done that. In the end, the decision came down at Sun, that they were having challenges with another CPU project, and they needed to move the people on the gallium arsenide project over to a CMOS project, that was going to be higher volume, and needed to be done quicker. We ended up canceling something again. Could have been fantastic. There was nothing technically wrong with the project. But just business conditions are such that things change.

In a career as a computer architect, it may take 5 to 10 years to get a new project out the door. In fact, in my experience, generally, a lot of brand new projects are on about a 10-year curve. So if I look at what happened with the RISC microprocessors, for example. We published first paper on RISC in 1980. It was ridiculed for the first two years. Then as people started to do actual implementations and bring it out, it did better. Then there was kind of a crest of RISC, and it seemed like it was going to take over the world. And then the other people fight back a little bit.

In some sense, some of the techniques for RISC that were appropriate for the technology of the day, for example, having a 32-bit fixed length instruction, was really great for five micron CMOS. But, we're nowhere near five micron CMOS any more. Are those still the right trade offs? Much less clear, and so things tend to go down a little bit. We've seen the number of RISC architectures declining. Whereas, it's basically just ARM that's left and a little bit of SPARC on the side maybe, and a little bit of MIPS.

Krewell: MIPS, still--

Ditzel: A tiny bit, but they're not certainly in the glory days if they were before.

Krewell: The PowerPC is still around as well. I know PowerPC is a very complex RISC processor--

Ditzel: Transmeta took about 10 years. It takes a year or two to get the ideas together, to go out, to get funding, to get momentum, to build a team, to do an implementation takes three or four years. It takes a couple of years to bring it to market. What you'll find is you had to make trade offs for expediency in the first generation, that you're going to fix in the second generation. By the time you get the second or third out, it's kind of a 10-year period.

In doing projects at Intel as well, what I saw was, their official methodology as such, it will take 10 to 15 years to do a new project, because of all the steps of pathfinding, and getting approvals, and going to the next level, and budgeting to get something to be a product. No senior manager would readily think that it should take 10 to 15 years. But, in fact, if you look at what it takes to step through the processes, that's what history has shown for going through that.

That's, again, very challenging if you want a career as a computer architect. To say, hey, I want to sign up and do something, but I'm not going to see the result for 10 to 15 years. And that's also a disincentive for doing brand new computer architectures.

Krewell: Well, yeah-- Intel's last brand new architecture was Itanium. As we mentioned earlier, it wasn't a very successful experiment. I think that's probably why they are a bit gun shy. It's pretty much x86 everywhere these days.

Ditzel: Hard to explain why big companies do things sometimes.

Krewell: We've talked about a lot of different places you've worked over the years, and we've talked a little bit about what you see coming in the market right now in terms of lack of innovation. Overall, looking back at your career, what were your high points? What were the things that you felt were the most interesting, most challenging? Was it Transmeta? Was it the early work?

Ditzel: Transmeta was certainly the most interesting, because to do a software-defined processor had really never been done to that extent. People had done little bits of emulation of user level code or something, but to really take on the entire architecture, and say you could do it, was something nobody knew whether that was really reasonable or not. And now when NVIDIA ships the Denver CPU, nobody thinks twice about that as a technique. It's just commonly accepted.

Certainly there was a lot of excitement about it at the time, and we had the Internet bubble, and things were, everybody was very interested in processors. But, I think a lot of the work on SPARC, at the early days the Sun, also had similar interest. The whole issue with getting RISC started. Most of the major computer conferences around 1980 to '85, would have a RISC versus CISC debate panel session in the evening. Where we would yell horrible names at the other side about how stupid they were, and not understanding our ideas, and it was entertaining.

Krewell: It was great fodder in the '90s for the Microprocessor Forum, and such. It kept Michael Slater busy.

Ditzel: I'd say a lot of the things that are very interesting to me in the past are also things which have not been visible to the rest of the world. When you work on a new microprocessor, quite often these are secret projects. You only knew about the public side, but there are a lot of other secret projects.

One of the projects that was very exciting was a project we started about 1991 or so, when I was at Sun Microsystems. We went off and we hired the Soviet supercomputer design team. This is a time when the Soviet Union was not doing well, it was breaking apart, people didn't have budgets anymore, and were looking for jobs. What we were able to do at Sun was to go off and actually hire about 300 people who would build the unique Soviet supercomputers under the direction of Professor Boris Babayan.

In the Soviet Union, there were a lot of clones. There were a few groups that did new design. They had done some very interesting work in stack machines, tagged architectures, VLIW, and different kinds of binary translation, as well. We had a large project at Sun Microsystems doing a VLIW SPARC. It turns out that never came out. We got reasonably far, but that was about the time that I quit Sun, and went to Transmeta. After I left, I don't know why, but the project fell apart.

Projects like that were very interesting. Certainly the project I did at Intel was extremely interesting, but difficult, in the sense, that it was tough to get people to listen to enough of the details to understand why what you're doing was better, as opposed to, wait a minute, you're not in the geographic area that I'm in, and so, therefore, you're a threat to my business if you succeed.

Krewell: It got parochial interest, within a large corporation.

Ditzel: Trying to do something new was always very exciting. Without saying what it was exactly, the opportunity we had at Intel was to replace all three of Intel's architectures-- the Atom, the Xeon desktop, and the high end Xeon Phi supercomputer, with one brand new architecture.

Krewell: Oh, that's pretty radical.

Ditzel: It was extremely challenging. That was the reason I gave it a try. And people said, why would you go to Intel? Everybody thinks you don't like them. Well, it was nothing personal. But, if we could have made a change like that, that could have really changed the industry in a big way. That's why I went there for a while.

Krewell: I mean that certainly would have been a big change in the industry and would have had a large impact. But then, you also learned the hard part of moving a big giant company that has a lot of inertia.

Ditzel: Things are hard to predict. In particular, things are hard to predict, the lessons I would say is, what you can do with your technology is actually far more predictable than the management structure you currently have in place. As an example, when we did Transmeta, we were trying to sign up IBM as a fab and do a deal with IBM. Which we managed to do. But, between the time we started the negotiation, and the time we finished, almost five years later, we had gone through five different vice presidents at IBM who were in charge of the deal.

Because constant reorganizations keep changing things. I think, in terms of probably at least half the projects I've been on that have been canceled, they had been canceled because of management-associated changes, one way or the other. Where the management team that was supporting you, suddenly has left and gone to another company, and now new management comes in. They have their own ideas of what they want to do, and they don't want to give credit to the prior managers. Projects suddenly change around, and that'll just happen on big expensive projects.

Krewell: At Transmeta, you ran into a glitch trying to use IBM as a fab--

Ditzel: IBM was fine as a fab. We never had any trouble there. When we switched off of IBM, we had some challenges.

Krewell: Yeah, that was when you had your problems, yes. And actually, only if you had stayed at IBM, maybe the part wouldn't have been as competitive in terms of process geometries, but at least would have yielded parts.

Ditzel: There are always big company challenges, and one of the challenges we had with IBM, is they had decided they were going to stop making bulk CMOS and only make SOI wafers in the future. That decision didn't last very long. It lasted long enough for us to have to consider the implications of, if our wafer prices doubled versus moving somewhere else for a fab.

Krewell: You also had back biasing technology. Would that have worked on the SOI?

Ditzel: We hadn't used that at the time. No, it probably would not have worked that way. Basically, by moving and staying on a standard CMOS, that was something we could continue to use.

Krewell: Now, actually SOI is trying to make another comeback, as an interim step before we get to FinFET.

Ditzel: Could be. It will be interesting to see where technology develops, and what really catches on. But, it's been very hard to beat standard CMOS, just because the volume production in the constant treadmill you are on for improvement.

Krewell: What do you think about Moore's Law slowing down? Even Intel has admitted now that it's not two years between nodes, it is going to be two and a half years.

Ditzel: Yeah, certainly things are slowing down in terms of technology development. Particularly, people are slowing down how they utilize transistors. For example, if you spend a lot of effort to get a particular clock speed, if your transistor of the next generation is 20% faster, maybe you'll want to take 10% of that, and just use it as efficiency benefits, and you won't worry about getting the maximum speed. Clock rates have basically slowed down.

The other thing that had helped the industry for a lot of the years, as every technology shrink, your voltage dropped by about a factor of 0.7. That's now slowed down as well. That was important because, as you doubled the number of transistors, that voltage drop meant that you could keep the power levels reasonable. Because your power is related to your voltage squared, so that 0.7 drop gave you roughly half the power. You could double the number of transistors, and end up with more or less at the same power. Now that we're trying to double the number of transistors, but not dropping the supply voltage, it is creating real power challenges. People are having to work on a lot of issues there.

Krewell: There's a fair amount of process work being done on sub-threshold logic to try to get it down below 0.8, 0.7 volts.

Ditzel: Yeah, but things run very slow with sub-threshold logic. The question is, can you use a lot of really slow processors to be as good as one really fast processor? It's been much more difficult for that to happen. In fact, a lot of the promise of multi-core, maybe 10 years ago, was, oh, we'll just use 8 or 16 CPUs. In the future, you're going to have notebook computers with 128 CPUs in them. So we don't need to build faster individual CPUs. That turned out not to work at all. Basically, and for most computers, two or four cores is about as much as most people will need on the desktop. And they have a tough time keeping those busy. Having a fast individual core is still a very good thing. And the dream that we can take individual programs, and split them up the run across multiple cores, is still something that has not been realistically demonstrated, in my view.

Krewell: At least on the wide, wide range of applications. There are some obviously parallel-izable tasks, like graphics.

Ditzel: Sure, if the tasks can be split up, that's one thing. And there's a lot of parallelism there. But, taking an individual single thread program, and dividing in across multiple cores, has turned out to be much harder in practice. Getting people to break up their programs has also been harder in practice. I think if there's a desire out there, people are able to explore as many cores as possible, at some point you actually need each core to be faster. So there's a desire for that as well.

Krewell: Amdahl's Law is still part of the equation. It is still needed.

Ditzel: I think we still needed better CPU cores.

Krewell: We're not done yet there, I guess.

Ditzel: I hope not.

Krewell: Actually, to go back to the Russian connection there, for a second-- that was interesting. It did fall apart, you said, after you left. Do you follow what they're doing these days?

Ditzel: Sure, quite a bit. So, what happened was much of that team got picked up by Intel. They went on to do-- actually, probably, the single most innovative new architecture at Intel. When I joined Intel, I went and thought perhaps I could help them with that. I got assigned to do a different project, which I thought was also innovative, even maybe a little more short term focused here. But, I fully support what they were doing as being extremely innovative there. I was able to keep a number of those connections. In terms of where there's interesting work going on, Russia is a place that is perhaps underestimated, particularly in software strength. There's a lot of good stuff going on there still.

Krewell: We don't hear much about them in the mainstream press or media.

Ditzel: Mostly it's what they do other behalf of other people. You'll see it on behalf of other companies.

Krewell: We've talked about your career. We've talked about what you thought were interesting projects and your biggest challenges over your career. What are you working on now?

Ditzel: I left Intel about two years ago. Taking a little bit of a break, but what I've decided to do is to spend my time on things I think are both personally interesting, and which could maybe make a big change in the industry. One of the things I've been looking at is 3D stacking of chips. I think it's the obvious place to go, and the question is going to be, why hasn't 3D stacking taken off? In my mind, the simple explanation is about cost. If you take four chips and you stack them on top of each other, and you use what seems to be an obvious way of interconnecting. What's called through silicon vias. For that, you drill a hole in the piece of silicon, you plate up like a printed circuit board, solder in between there, and you connect the chips with little solder bumps. It tends to about double the cost per die of those chips.

If you're building memories, the question is, well, who would pay twice as much cost per bit for memory in order to stack in 3D with this particular technique? People have used other stacking techniques with wire bond and other things, which aren't as technologically advanced. But then that limits or their usage. Cost has been really big issue there.

One of things I've been doing is helping a professor friend of mine, Professor Kuroda, from Keio University in Japan, who has a very simple idea, which is to say, look, rather than having a special, expensive process you have to implement with drilling holes in the silicon, let's just lay the chips on top of each other, and we'll have the chips communicate wirelessly. And it just seems like such an obvious idea. He has a way of using very simple antennas. It's really just inductive coupling. You make a couple of turns of wire, maybe only 100 microns on a side or so. You put these coils on top of each other, and there's so much coupling between these two, it's almost as good as having a solid wire going between the two. You can run these at many gigabits per second. They're extremely low power, maybe 100 times lower power than other forms of communication, that I think this could--

Krewell: There is not much loss between using the capacitive or inductive coupling?

Ditzel: No, there's no loss because the signals transmitted and you have basically a cross-coupled latch at the output, and when the voltage pulse comes in, it'll transmit because they just got the standard digital signal there. There's a little bit of analog circuitry to go through the analog coil, but fundamentally, it just looks like you're driving one gate on one layer, and the output comes out another gate on another die.

I think this will enable people to think about how to design systems in 3D, and will offer great opportunities for more density, smaller packaging, lower power, lower cost. I think maybe that will be the enabler that 3D has been waiting for. That all of the previous approaches, which are too expensive, couldn't deal with.

I've been trying to be an advocate for that technology. I'm serving as advisor on with a couple of other companies. And, maybe even looking at getting some new things started. I'm still tinkering in the back of my mind with some new ideas in processor architecture, and we'll see if those take off.

Krewell: You certainly haven't retired.

Ditzel: Just taking things at my pace, and I'm not working for any other big company full time right now. Trying to explore the opportunities and see where things will go.

Krewell: That's good. You have always been involved in very interesting projects. I look forward to seeing what comes out in the future.

Ditzel: One of the challenges of working at Intel was I had to be very isolated. Didn't talk to contacts on the outside because I couldn't talk about what I was doing. I think, since I've left, I'm using this opportunity to renew some contacts, and see what other people are doing, and trying to understand what the next wave of opportunities are.

Krewell: And then, looking at those next wave opportunities, we talked a little bit about there is still a need for better computer architectures. Do you see yourself getting back to that side of the business?

Ditzel: Yeah, it's possible. I'm working on some some ideas currently. I think there's at least another factor 2 to 10x possible out there by, I'll just say, doing more simply designed architectures, that are more matched to the application and combining that with how you take advantage of 3D, I think, will be the big wave of opportunity in the future. So, that's one of things I'm exploring.

Krewell: Well, it sounds a little bit like specialized architectures that you could combine the various groups of the specialized architectures to do different things well, and then somehow connect those together.

Ditzel: I think, particularly in the mobile world for tablets, phones, and internet of things. I think instruction sets and implementation techniques do matter still. I think there's some big gains to be had. That's not in the best interest of the vested interests, who want to convince you that their instruction set is the best. But, I think for particular applications, it may not matter so much of you have an internet of things device what instruction set it is, or even in phones and tablets.

How we distribute applications these days has changed a lot from the last 20 years. I think that itself could enable a new generation of architectures, once we stop having the applications being fixated on a particular implementation technique. If you look at the Apple Watch, for example, I think they've mandated they're going to use more independent form of submitting applications to the App Store using LLVM bitstream. Maybe techniques like that will allow us to innovate new architectures without being tied in to some backwards legacy instruction set.

I think the people who develop software take that in mind. That will help future hardware architects. In one of the notions though is that there may be a new chip every year. But software lasts forever. What we need the software companies to do, the Googles and the Apples of the world, is to think about how they design apps. So, if that app is done, and it's going to last for 20 or 30 years, how could you use that to implement something differently that's not so tied specifically to one particular machine instruction set or computer architecture? I think liberalizing those, and a lot of that it is actually happening out there, I used the example of the Apple Watch, it's actually happening. I think what Google is doing with the Android apps is extremely interesting. It's not tied to any one particular architecture.

Krewell: As long as it's a well-behaved app and you use it in a way--

Ditzel: I think these will open up opportunities for future computer architecture, where they could be drastically more efficient then we see in current chips today.

Krewell: Even Microsoft tried to do that with their modern apps where the same program could run on either an x86 or an ARM-based device seamlessly. And that was one of the goals. Unfortunately, Surface wasn't so successful at doing that. The execution may not have been the problem. I think there's still some issues with legacy. Still a lot of legacy in the Microsoft world that has to be overcome. Actually, so we've talked a little bit about what you think the best next big thing is. We've talked a little bit about the future. What else would you like to say about where things are going? What your vision of the future is in computing? Is there any additional point you want to add?

Ditzel: I think how we do computing in the cloud, for high performance computing applications, is probably where there is going to be more innovation. For what I do on my phone, I expect this to be fairly evolutionary for a while. But, I think, the challenges of the size of the data centers, where you're limited to maybe 20 megawatts of power, how do you get more computing done there, is a big enough problem that people can afford to try specific new architectures to help out those type of data center computing environments. So I'm hoping to see some innovation there, in what happens.

I'd like to see more innovation in processor architectures than is happening now. I think we're kind of at a lull just last year or two. It's not looking particularly good for me, in terms of what's happening. We've seen people like IBM move their semiconductor business off to global foundries. Which is maybe good from a fab point of view, but it's probably not so good in terms of people who are studying architectures, who need to be closely tied with the fab. I think that's going to become tougher and tougher to do.

And, I think there is going to be a concentration particularly between, say, phones and tablets, and maybe just the top three vendors or so who are driving all the designs. And the question is, how much will any of them be willing to invest? Some people are doing their own architectures, others just pick up the latest thing from ARM. And ARM will supply the generic thing for everybody, so maybe ARM will do some innovative work.

Krewell: We'd like to see that. What about the very popular topic of Internet of Things right now. These very small, connected devices that will be pervasive in our world, what--

Ditzel: I believe in pervasive devices that are all connected to a network by some radio. I want to be able to talk to my device and know what's the state of its battery on any sensor. The last thing I want is 50 sensors planted around my house, and not knowing which ones have a dead battery in them, because they can't talk to me. That's really good. I think that's very interesting.

I think a lot of companies are chasing this though without perhaps knowing how to monetize it. I think that's more the question I have, if you can make internet of things chips very, very small, maybe you could make a few million chips just with a small number of wafer lots. It's not going to keep the fabs full. I think there's going to be some disruptive economics coming.

As Moore's Law continues to some extent and die sizes continue to shrink, it's hard for people to spend enough money to fill up all those transistors in the way that we used to. I think what's going to happen instead is, they'll actually take in terms of cost reductions. So, hopefully we'll get cost reductions there. But that will mean that there's less money coming in, maybe to innovate on another new designs.

Krewell: Well, in a sense though, with internet of things, a lot of the devices may stop at a certain process note. 28 nm seems to be the sort of sweet spot in terms of cost--

Ditzel: It's going to be cost. A cost-drive basis, maybe a little bit power.

Krewell: But, if everything comes to a large volume of these guys, say 28[nm], and say, OK, I'm done, I'm not going to go to 20[nm] or 16[nm] FinFET or anything else because it's too expensive, who's going to fund the next generation?

Ditzel: That will be the challenge in economics here, yes. And while you're in growth mode, it's really great to fund new fabs and do that, but again I think you're absolutely right. It's going to be one of the challenges in electronics. I think we're going to see big slowing down in interest of how rapid the advances are.

I think there will still be advances, but people aren't going to move to the next node unless there's a significant cost advantage. It used to be that cost advantage was very clear. And then a few years ago, I remember one large graphics maker complaining that from their fab, they were paying more dollar per transistor doing a shrink than they were just staying where they were. And that's just not economically interesting to too many people.

Krewell: But for higher performance, you do need new process. Servers are an important part of that. 3D stacking maybe one of those alternatives where people mix and match various process nodes, depending on what they need, and then they convert it to stack to get the density.

Ditzel: What we've seen in NAND Flash, is people actually going backwards in terms of lithography so they could stack higher. So some the 3D NAND Flash from Samsung, for example, or Toshiba or others, are doing that. We are actually going in a coarser metal geometry, so they can actually grow in the vertical direction without having to stack multiple chips. They just do this by going to lots and lots of metal layers. I've seen people to maybe up to 100 layers of metal. I think could be coming reasonably soon as a technique. I think that's, in part, because the cost of stacking multiple individual chips is too high. I think we'll see lots more in terms of density. But, it's not clear to me, that we're going to be holding handheld devices that will be 10 or 100 times faster in terms of running single threaded apps for us.

Krewell: It's a single thread it seems to be a big challenge. We haven't really solved that challenge in architecture.

Ditzel: We don't teach people to program differently perhaps that doesn't seem-- to be on the near horizon

Krewell: They've been trying to do that for a long time now. It's been 10 years, and it still hasn't changed significantly yet. So, one last thing-- do you have any career advice for young people who might be getting into this industry, this crazy industry that we've been working in?

Ditzel: Yeah, engineering's always good. Software, in particular, is still, I think, pretty strong.

Krewell: We need to do chip architecture, too.

Ditzel: What we're doing in chip architecture, I'd say, is pretty sobering, in terms of, a lot of the jobs are not going to be in the US. And I think that's a real challenge people need to get used to. I think it's moving to Asia. A lot of the traditional electrical engineering VLSI types of jobs, I think, you need to look pretty hard at exactly what is it you want to do, and where you're willing to live in order to do that.

And, I think we can see that a little bit in Silicon Valley here, by how little money is going in a new chip start ups. And more money that's going into software companies, social networking, a little bit of internet of things, not so much. But, just generally, software-oriented approaches. I think there's a lot to be done in software. And that's the great part about software, you get powerful hardware underneath, the software can do lots of different things with it.

Krewell: But the software is going to expect that the hardware is going to continue to get more powerful and smaller and cooler over time. If that starts stalling, that may have a negative impact on the whole ecosystem.

Ditzel: Then we're going to need more trained engineers to solve that problem and come up with some breakthroughs.

Krewell: Exactly, we need to breakthrough this architecture barrier we're in.

Ditzel: But, generally, just good science engineering math, helping engineering move forward, would be the advice I'd give for young people these days. There are always going to be very good jobs there. You may have to look around for exactly where you're going to work physically, but there will be things available.

Krewell: Any final notes, I've gone through--

Ditzel: No, just great. I appreciate the chance to be able to talk a little bit. There's so many interesting projects. And I hope that the engineers for the next 50 years are going to find as many interesting new CPU designs, and find it as exciting as we found it in the past 20 years.

Krewell: How many of your old projects are in the Computer History Museum?

Ditzel: I think there's a fair number of pieces in the Computer History Museum. Right now, there's a wearable computing exhibit. I've certainly seen some of the companies I've been involved in many early wearable devices, which were very clunky to today's standards here. You can see people wanted to do it.

I think one of the challenges Transmeta had is when having very low power x86 CPUs, we enabled people to do lots and lots of prototypes, and we spent a lot of effort there. But people would build demonstration units, and they'd make maybe a hundred of them. And you've got to have somebody sell a million to 10 million units of something to make a business at it. Lots of demonstrations.

People are still trying to figure out what are we going to do with it. We hear about internet of things. I don't think we know exactly what's actually going to happen. I think we're going to look back in 10 years, and laugh and say, that's what we thought they were going to do. Well, they did something completely different.

But I think the pace of innovation is exciting, and I certainly hope it continues. I think having devices get continually small. Particularly, 3D stacking is absolutely going to happen. As we get the lower power of electronics, we'll be able to do more in smaller spaces. Will each device though be substantially more powerful? That's less clear. I think is more of a cost reduction, power reduction, with some increase in features, is more likely to be what happens in the predictable short term.

Krewell: We've got that down our record now of you saying that, so you should come back 10 years from now, and revisit all this, and see if it has changed much.

Ditzel: I'd be happy to do that. Thanks very much, Kevin.

Krewell: Thanks, Dave.

END OF INTERVIEW